Brad Dayley
Brendan Dayley

Sams **Teach Yourself**

AngularJS, JavaScript, and jQuery All in One

SAMS

About This eBook

Sams Teach Yourself AngularJS, JavaScript, and jQuery All in One in 24 Hours

Brad Dayley
Brendan Dayley

SAMS 800 East 96th Street Indianapolis, IN 46240 USA

Sams Teach Yourself AngularJS, JavaScript, and jQuery All in One

ISBN-13: 978-0-672-33742-0
ISBN-10: 0-672-33742-8

Library of Congress Control Number: 2015907445

Printed in the United States of America

First Printing August 2015

Mark Shirar

Compositor
Nonie Ratcliff

Contents at a Glance

Table of Contents

About the Authors

Brad Dayley is a senior software engineer with more than 20 years of experience developing enterprise applications and web interfaces. He has used JavaScript, jQuery, and AngularJS to develop a wide array of feature-rich web applications. He has a passion for new technologies, especially ones that really make a difference in the software industry. He is the author of *Node.js, MongoDB, and AngularJS Web Development*, *Learning AngularJS*, *jQuery, and JavaScript Phrasebook,* and *Sams Teach Yourself jQuery and JavaScript in 24 Hours*.

Brendan Dayley is a university student majoring in computer science. He is an avid web application developer who loves learning and implementing the latest and greatest technologies. He recently attended Dev-Mountain's Immersive Web Development program, specializing in web application development and AngularJS in particular. He has written a number of web applications using JavaScript, jQuery, and AngularJS and is excited about the future of these technologies.

Dedication

For D!

A & F

Jessie

My one and only

Acknowledgments

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

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

We Want to Hear from You!

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

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

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

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

Email: feedback@samspublishing.com

Mail: Sams Publishing
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Introduction

Welcome to *AngularJS, JavaScript, and jQuery All in One*. This book is designed to jumpstart you into the world of dynamic web application development using JavaScript, jQuery, and AngularJS. The book covers the basics of the JavaScript language, jQuery library, and AngularJS framework and how to use them to build well-designed, reusable components for web applications.

With billions of people using the Internet today, there is a rapidly growing trend to replace traditional websites, where one page links to another page and so on, with single page applications that have richly interactive elements.

The main reason is that users have become less patient with clicking, waiting, and then having to navigate back and forth between web pages. Instead, they want websites to behave more like the applications they are used to on their computers and mobile devices.

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

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

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

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

Who Should Read This Book

This book is aimed at readers who already have an understanding of the basics of HTML and have done some programming in a modern programming language. Having an understanding of JavaScript will make this book easier to digest, but it is not required because the basics of JavaScript are covered.

Why You Should Read This Book

This book will teach you how to create powerful, interactive web applications that have a well-structured, easy-to-reuse code base that will be easy to maintain. The typical readers of this book want to learn JavaScript, jQuery, and AngularJS for the purpose of building highly interactive web applications. The typical reader will also want to leverage the innovative Model View Controller (MVC) approach of AngularJS to extend HTML and implement well-designed and structured web pages and web applications.

What You Will Learn from This Book

Reading this book will enable you to build rich, dynamic interactions into your web pages and applications. Websites are no longer simple static content that consist of HTML pages with integrated images and formatted text. Instead, websites have become much more dynamic, with a single page providing a wide array of functionality and interactions.

Using AngularJS, jQuery, and JavaScript enables you to build logic directly into your web applications that allows you to interact with the user from your client-side application. These technologies also allow you to interact with back-end web services on the web server to create a comprehensive client-side web application. The following are a few of the things you will learn while reading this book:

- The basics of the JavaScript language
- How to implement JavaScript, jQuery, and AngularJS in your web pages
- How to dynamically modify page elements in the browser
- How to use browser events to interact with the user directly
- How to implement client-side services that can interact with the web server
- How to implement rich user interface (UI) components, such as zoomable images and expandable lists
- How to quickly build AngularJS templates with built-in directives that enhance the user experience
- How to bind UI elements to the data model so that when the model changes, the UI changes, and vice versa
- How to bind mouse and keyboard events directly to the data model and back-end functionality to provide robust user interactions
- How to define your own custom AngularJS directives that extend the HTML language
- How to build dynamic browser views that provide rich user interaction
- How to create custom services that can be easily reused in other AngularJS

applications

Why AngularJS, jQuery, and JavaScript in the Same Book?

The reason we decided to put AngularJS, jQuery, and JavaScript in the same book is that they all relate to each other. We've been asked questions like "Should I use AngularJS or jQuery?" or "Should I use JavaScript or jQuery?" many times. We see them as a stack that works together very well.

JavaScript is the base language that is supported by the browser. jQuery extends JavaScript with a syntax that is much more powerful and user friendly. AngularJS is an extension of jQuery (or at least a stripped-down version of jQuery) that provides an extremely powerful MVC framework with robust data binding functionality.

Understanding all three of these technologies and how they work together will make you a better web developer, even if you use another JavaScript framework or library to develop, because they provide the fundamental functionality that all good web applications need. You may decide that simple JavaScript fits the needs in one area, or jQuery/jQueryUI provides the perfect functionality for some web forms, or that you need the robust functionality of AngularJS for your web application. Either way, you will have the skills and understanding to be able to choose and implement the right technology.

What Is JavaScript?

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

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

What Is jQuery?

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

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

DOM elements.

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

What Is AngularJS?

AngularJS is a client-side framework developed by Google. It is written in JavaScript with a reduced jQuery library called jQuery lite. The entire ideology behind AngularJS is to provide a framework that makes it easy to implement well-designed and well-structured web pages and applications using an MVC framework.

AngularJS provides all that functionality to handle user input in the browser, manipulate data on the client side, and control how elements are displayed in the browser view. Here are some of the benefits AngularJS provides:

- **Data Binding**—AngularJS has a very clean method to bind data to HTML elements using its powerful scope mechanism.
- **Extensibility**—The AngularJS architecture enables you to easily extend almost every aspect of the language to provide your own custom implementations.
- **Clean**—AngularJS forces you to write clean, logical code.
- **Reusable Code**—The combination of extensibility and clean code makes it very easy to write reusable code in AngularJS. In fact, the language often forces you to do so when you're creating custom services.
- **Support**—Google is investing a lot in this project, which gives it an advantage where other similar initiatives have failed.
- **Compatibility**—AngularJS is based on JavaScript and has a close relationship with jQuery. That makes it easier to begin integrating AngularJS into your environment and reuse pieces of your existing code within the structure of the AngularJS framework.

Beyond AngularJS, jQuery, and JavaScript

This book covers more than jQuery and JavaScript because you need to know more than the language structure to create truly useful web applications. The goal of this book is to give you the fundamental skills needed to create fully functional and interactive web applications in just 29 short, easy lessons. This book covers the following key skills and technologies:

- HTML is the most current recommendation for web page creation. Every example in this book is validated HTML5, the most recent recommended version.
- CSS is the standard method for formatting web elements. You not only learn how

to write CSS and CSS3, but also how to dynamically modify it on-the-fly using jQuery and JavaScript.

- JavaScript is the best method to provide interactions in web pages without the need to load a new page from the server. This is the standard language on which most decent web applications are built.
- jQuery and jQueryUI are some of the most popular and robust libraries for JavaScript. jQuery provides very quick access to web page elements and a robust set of features for web application interaction. jQuery provides additional UI libraries that provide rich UI components for web applications.
- AJAX is the standard method that web applications use to interact with web servers and other services. The book includes several examples of using AJAX to interact with web servers, Google, Facebook, and other popular web services.

Code Examples

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

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

The titles for the listing blocks include a filename of the file that contains the source. These files can be obtained from the book's website (follow the directions on the back cover of this book).

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

Development Web Server

I chose Node.js with Express as the web server for the development environment for this book. You will get a chance to set up Node.js as the web server in Lesson 1. There are several reasons I chose Node.js over a more traditional web server like Apache or IIS, including the following:

- Node.js is extremely easy to install and set up.
- You can use Node.js to test your JavaScript snippets without having to use a web browser.

- There is a great Node.js plug-in for Eclipse that allows you to easily debug JavaScript.
- You do not need to understand a back-end scripting language such as PHP, Python, or Ruby because you can write your server-side script for Node.js in JavaScript.

Special Elements

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

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

Tip

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

Note

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

Caution

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

Q&A, Quizzes, and Exercises

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

Finally

We hope you enjoy this book and enjoy learning about JavaScript, jQuery, and AngularJS as much we did. These are great, innovative technologies that are really fun to use. Soon you'll be able to join the many other web developers who use them to build rich, dynamic, and interactive websites and web applications.

Part I: Introduction to AngularJS, jQuery, and JavaScript Development

Lesson 1. Introduction to Dynamic Web Programming

What You'll Learn in This Lesson:

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

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

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

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

Understanding the Web Server/Browser Paradigm

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

Looking at Web Server to Browser Communication Terms

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

Although the concept is simple, several steps must take place for the data to be requested from the server and displayed in the browser. The following sections define the components involved, their interactions with each other, and how JavaScript, jQuery, and AngularJS are involved.

Web Server

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

Browser

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

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

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

DOM

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

Click here to view code image

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

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

Browser Events

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

Browser Window

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

URL

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

HTML/HTML5

Hypertext Markup Language, or HTML, provides the basic building blocks of a web page. HTML defines a set of elements representing content that is placed on the web page. These element tags are used to create objects in the DOM. Each element tag pair is represented as an object in the DOM. Each element is enclosed in a pair of tags denoted by the following syntax:

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

For example:

Click here to view code image

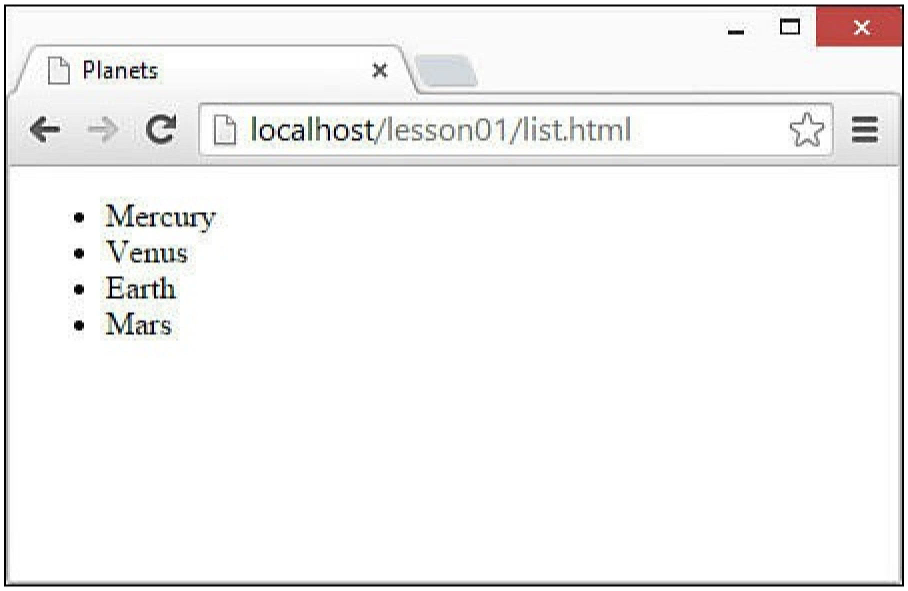
```
<p>This is an HTML paragraph.</p>.
```

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

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

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

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



- Mercury
- Venus
- Earth
- Mars

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

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

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Server Side Script</title>
    <meta charset="utf-8"/>
  </head>
  <body>
    <ul>
      <li>Mercury</li>
      <li>Venus</li>
      <li>Earth</li>
      <li>Mars</li>
    </ul>
  </body>
</html>
```

CSS/CSS3

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

Cascading Style Sheets, or CSS, provide a way to easily define how the browser renders HTML elements. CSS can be used to define the layout as well as the look and feel of individual elements on a web page.

CSS3, or Cascading Style Sheets level 3, is the next generation of CSS that incorporates more special effects, such as transformations and animations. It also provides rich additions for borders, backgrounds, and text.

To illustrate CSS, we’ve added some CSS code to our example from Listing 1.1. Listing 1.2 uses CSS to modify several attributes of the list items, including the text alignment, font style, and changing the list bullet from a dot to a check-mark image. Notice how the CSS style changes how the list is rendered in Figure 1.2.

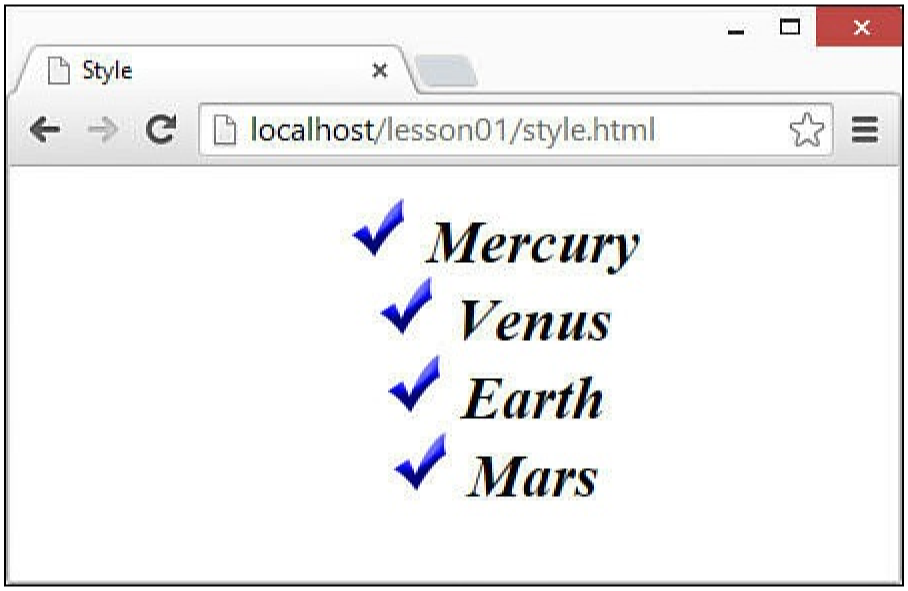


FIGURE 1.2 The CSS code dramatically changes the look of the list in the browser.

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

[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Style</title>
    <meta charset="utf-8" />
    <style>
      li {
        text-align: center;
        font-family: "Times New Roman", Times, serif;
        font-size: 30px;
        font-style: italic;
        font-weight: bold;
        list-style-image: url('/images/check.png');
        list-style-position: inside;
      }
    </style>
  </head>
  <body>
    <ul>
      <li>Mercury</li>
      <li>Venus</li>
      <li>Earth</li>
      <li>Mars</li>
    </ul>
  </body>
</html>
```

HTTP/HTTPS Protocols

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

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

The following sections discuss HTTP headers and the two most common types of HTTP request, GET and PUT.

HTTP Headers

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

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

Following are a few of the more common ones:

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

request or response.

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

GET Request

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

Click here to view code image

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

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

Click here to view code image

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

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

Click here to view code image

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

POST Request

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

Web Server and Client-Side Scripting

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

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

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

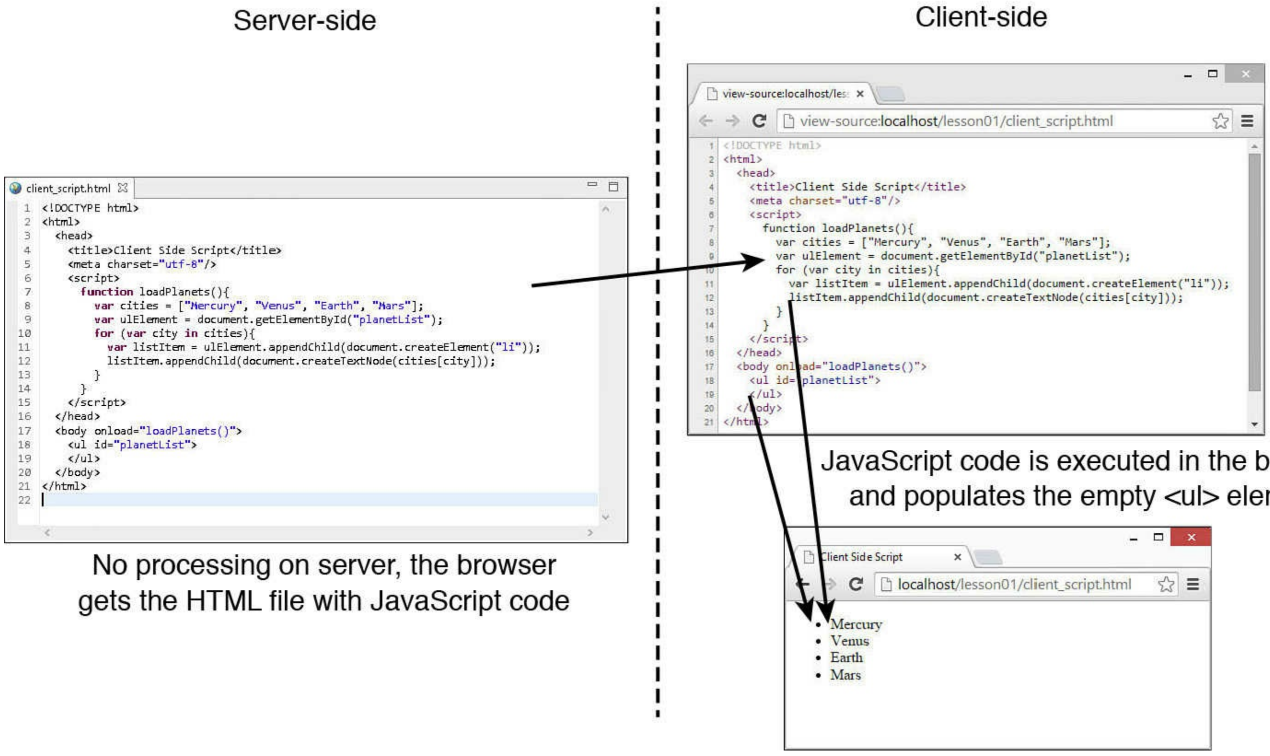
Client-Side Scripting

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

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

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

Figure 1.3 diagrams the flow of data between the web server and the browser for a simple client-side script that uses JavaScript to populate an empty `<ul>` element with a list of planets. Notice that the file located on the server is the same one sent to the browser, but in the browser, the JavaScript adds `<li>` elements for each planet. You do not need to fully understand the JavaScript code yet, just that the HTML is dynamically changed on the client and not the server.


Server-side
Client-side
client_script.html
view-source:localhost/les
view-source:localhost/lesson01/client_script.html
Client Side Script
localhost/lesson01/client_script.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Client Side Script</title>
    <meta charset="utf-8"/>
    <script>
      function loadPlanets(){
        var cities = ["Mercury", "Venus", "Earth", "Mars"];
        var ulElement = document.getElementById("planetList");
        for (var city in cities){
          var listItem = ulElement.appendChild(document.createElement("li"));
          listItem.appendChild(document.createTextNode(cities[city]));
        }
      }
    </script>
  </head>
  <body onload="loadPlanets()">
    <ul id="planetList">
    </ul>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Client Side Script</title>
    <meta charset="utf-8"/>
    <script>
      function loadPlanets(){
        var cities = ["Mercury", "Venus", "Earth", "Mars"];
        var ulElement = document.getElementById("planetList");
        for (var city in cities){
          var listItem = ulElement.appendChild(document.createElement("li"));
          listItem.appendChild(document.createTextNode(cities[city]));
        }
      }
    </script>
  </head>
  <body onload="loadPlanets()">
    <ul id="planetList">
    </ul>
  </body>
</html>
```

- Mercury
- Venus
- Earth
- Mars

No processing on server, the browser gets the HTML file with JavaScript code
JavaScript code is executed in the browser and populates the empty <ul> element

FIGURE 1.3 The JavaScript is executed in the browser, and so the HTML document rendered by the browser is different from the one that was originally sent.

Server-Side Scripting

There are two major types of server-side scripting. These are server-side templates and AJAX request handlers. Each of these methods requires that code be written on the server to either dynamically generate an HTML document before it is sent to the browser or to dynamically generate data that can be consumed by a client-side application.

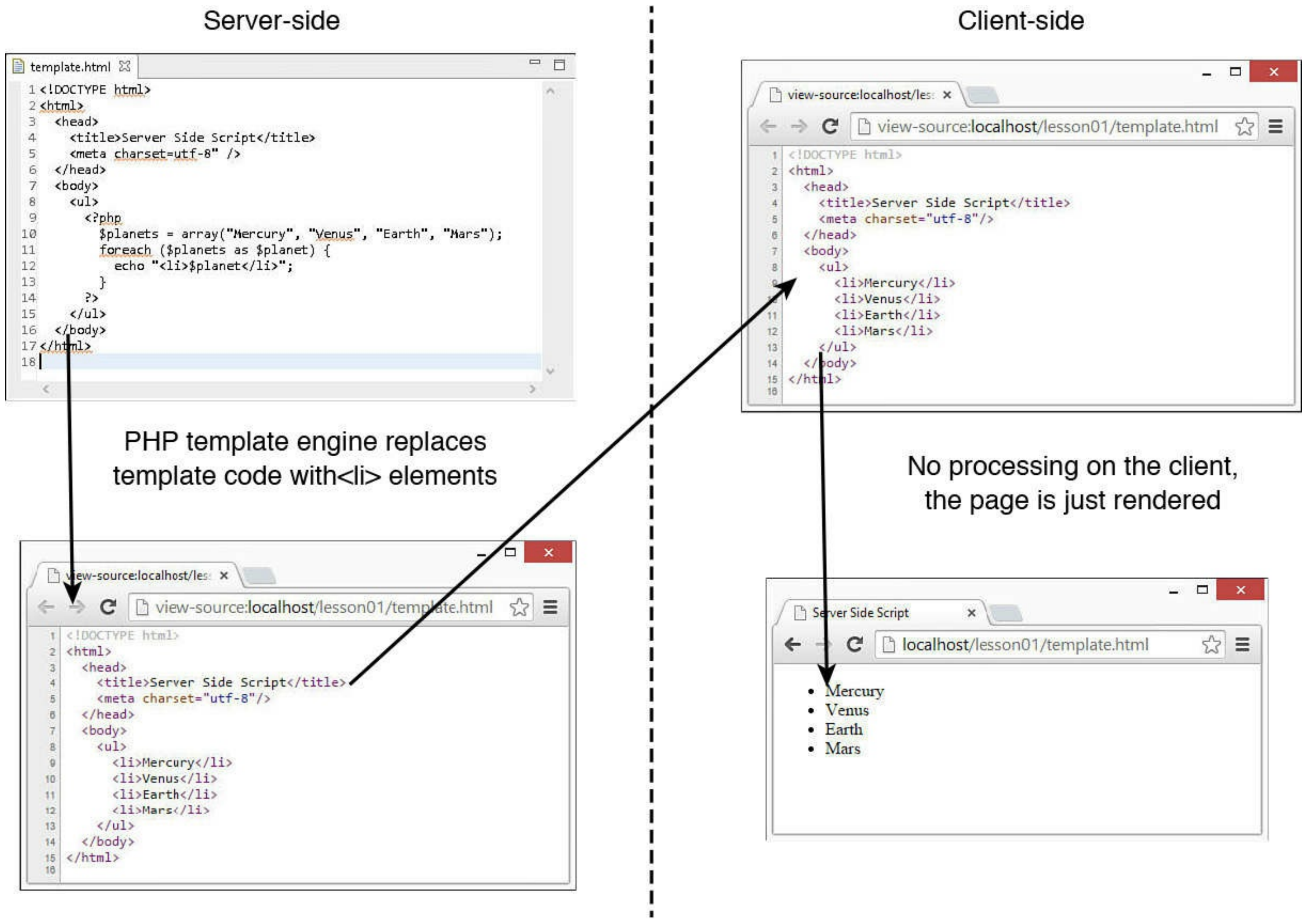
Server-Side Templates

The first type is to use a PHP, .Net, Java, or other type of application that is run on the server that generates the HTML page, or at least parts of the HTML page, dynamically as they are requested by the client.

The main advantages of this type of server-side scripting is that data processing is done completely on the server side and the raw data is never transferred across the Internet; also, problems and data fix-ups can be done locally within the server processing.

The disadvantage of this type of server-side scripting is that it requires more processing on the server side, which can reduce the scalability of some applications.

Figure 1.4 illustrates using a simple Node.js application on the server that will dynamically create an HTML document that populates a list of planets. In the example in Figure 1.3, PHP code is used, and the web server's PHP engine will replace the code in the <?php> tag with the output generated by the PHP script.


Server-side
Client-side
template.html
PHP template engine replaces template code with<li> elements
view-source:localhost/lesson01/template.html
No processing on the client, the page is just rendered
Server Side Script
localhost/lesson01/template.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Server Side Script</title>
    <meta charset=utf-8" />
  </head>
  <body>
    <ul>
      <?php
        $planets = array("Mercury", "Venus", "Earth", "Mars");
        foreach ($planets as $planet) {
          echo "<li>$planet</li>";
        }
      ?>
    </ul>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Server Side Script</title>
    <meta charset="utf-8"/>
  </head>
  <body>
    <ul>
      <li>Mercury</li>
      <li>Venus</li>
      <li>Earth</li>
      <li>Mars</li>
    </ul>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Server Side Script</title>
    <meta charset="utf-8"/>
  </head>
  <body>
    <ul>
      <li>Mercury</li>
      <li>Venus</li>
      <li>Earth</li>
      <li>Mars</li>
    </ul>
  </body>
</html>
```

- Mercury
- Venus
- Earth
- Mars

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

You don't necessarily need to understand how the code works at this point; you only need to understand that the HTML document is dynamically generated on the server and not the client.

Note

There are numerous methods of using HTML templates on the server-side. We do not cover those here because they are out of the scope of the book. If you would like to learn more about server-side scripting, you might investigate PHP, Ruby on Rails, and Node.js more fully.

AJAX Handlers

The second major type of server-side scripts are applications that return raw data in the form of raw JSON or XML to the browser in response to an Asynchronous JavaScript plus XML or AJAX request. AJAX requests are designed to allow JavaScript running in the browser client to get raw data from the server.

AJAX reduces the need to reload the web page or load other web pages as the user interacts. This reduces the amount of data that needs to be sent with the initial web server response and also allows web pages to be more interactive.

For a simple example of AJAX, we've constructed two scripts—Listing 1.3 and Listing 1.4. Listing 1.3 is an HTML document with JavaScript that runs on the client after the page is loaded. The JavaScript makes an AJAX request back to the server to retrieve the list of planets via a server-side script. Listing 1.4 simulates the JSON data that could be returned by the server-side script. The list of planets returned is then used to populate the HTML list element with items.

LISTING 1.3 ajax.html A Simple JavaScript Client-Side Script Executes an AJAX Request to the Server to Retrieve a List of Planets to Use When Building the HTML List Element

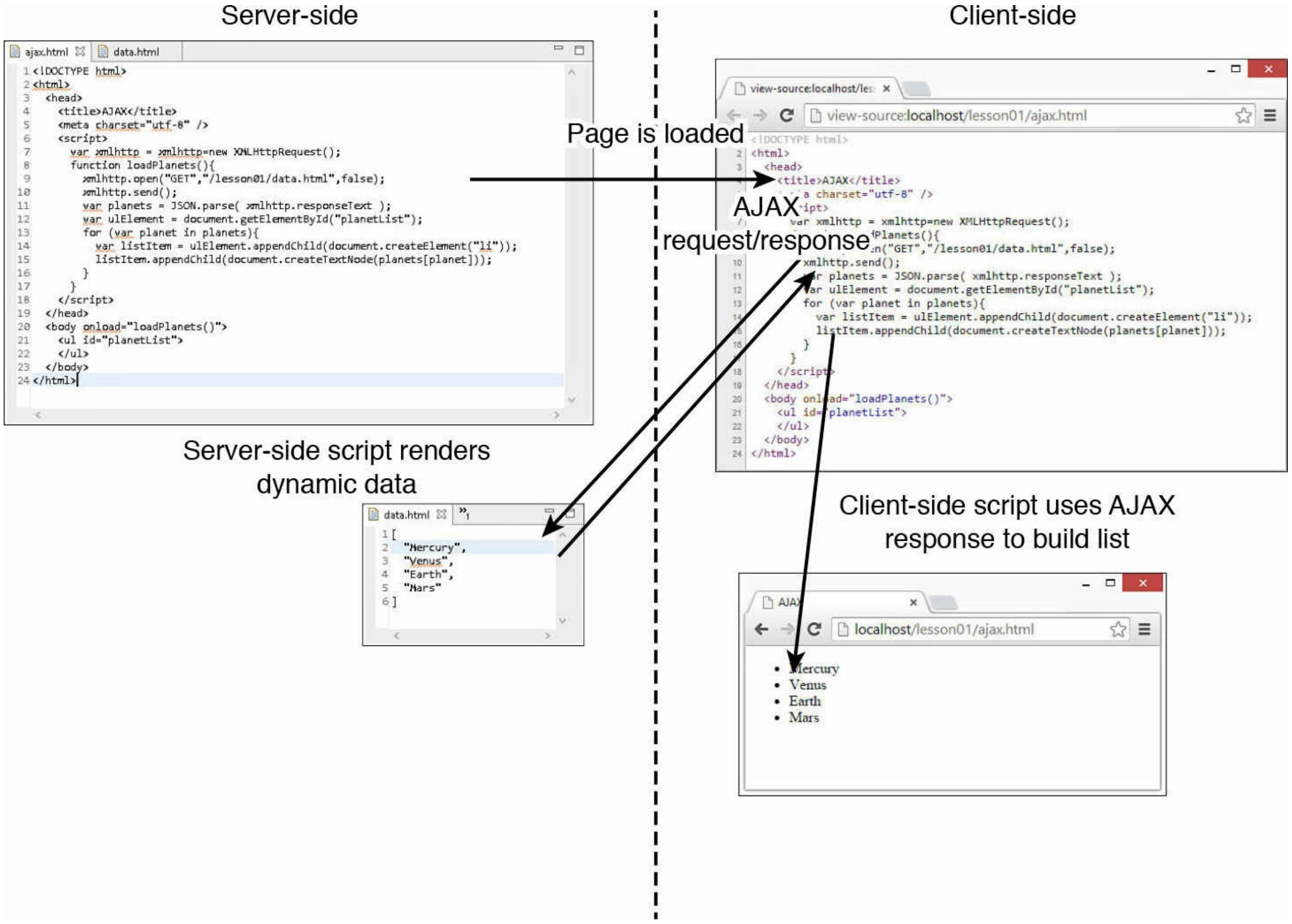
Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>AJAX</title>
    <meta charset="utf-8" />
    <script>
      var xmlhttp = new XMLHttpRequest();
      function loadPlanets(){
        xmlhttp.open("GET","/lesson01/data.html",false);
        xmlhttp.send();
        var planets = JSON.parse( xmlhttp.responseText );
        var ulElement = document.getElementById("planetList");
        for (var planet in planets){
          var listItem = ulElement.appendChild(document.createElement("li"));
          listItem.appendChild(document.createTextNode(planets[planet]))
        }
      }
    </script>
  </head>
  <body onload="loadPlanets()">
    <ul id="planetList">
    </ul>
  </body>
</html>
```

LISTING 1.4 data.html Dynamic JSON Data Generated by a Server-Side Script

```
[
  "Mercury",
  "Venus",
  "Earth",
  "Mars"
]
```

Figure 1.5 illustrates the flow of communication that happens during the AJAX request/response. Notice that a second request is made to the server to retrieve the list of cities.


Server-side
Client-side
Page is loaded
AJAX request/response
Server-side script renders dynamic data
Client-side script uses AJAX response to build list
ajax.html
data.html
view-source:localhost/lesson01/ajax.html
localhost/lesson01/ajax.html
AJAX

```
<!DOCTYPE html>
<html>
  <head>
    <title>AJAX</title>
    <meta charset="utf-8" />
    <script>
      var xmlhttp = xmlhttp=new XMLHttpRequest();
      function loadPlanets(){
        xmlhttp.open("GET","/lesson01/data.html",false);
        xmlhttp.send();
        var planets = JSON.parse( xmlhttp.responseText );
        var ulElement = document.getElementById("planetList");
        for (var planet in planets){
          var listItem = ulElement.appendChild(document.createElement("li"));
          listItem.appendChild(document.createTextNode(planets[planet]));
        }
      }
    </script>
  </head>
  <body onload="loadPlanets()">
    <ul id="planetList">
    </ul>
  </body>
</html>
```

```
[
  "Mercury",
  "Venus",
  "Earth",
  "Mars"
]
```

- Mercury
- Venus
- Earth
- Mars

FIGURE 1.5 Using an AJAX request, JavaScript can send an additional request to the server to retrieve additional information that can be used to populate the web page.

Setting Up a Web Development Environment

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

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

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

development web server is required to test out scripts and interactions.

- **Development Web Browser(s)**—Again, you should initially develop to the browser that you are most comfortable with or that will be the most commonly used.

For the purposes of this book, we have chosen to use Eclipse for the IDE and Node.js for the development web server. These technologies are very easy to set up, configure, and get going with. They also integrate well with each other and are easily extended. The following sections take you through the process of setting up Node.js and Eclipse for JavaScript development.

Setting Up Node.js

Node.js is a JavaScript platform based on Google Chrome's V8 engine that enables you to run JavaScript applications outside of a web browser. It is an extremely powerful tool, but this book covers only the basics of using it as the web server to support your web application examples.

To install and use Node.js, you need to perform the following steps:

1. Go to the following URL and click INSTALL. This will download an installable package to your system. For Windows boxes, you will get an .MSI file; for Macs, you will get a .PKG file; and for Linux boxes, you can get a .tar.gz file.

   `http://nodejs.org`

2. Install the package. For Windows and Macs, simply install the package file. For Linux, go to the following location for instructions on installing using a package manager:

   `https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager`

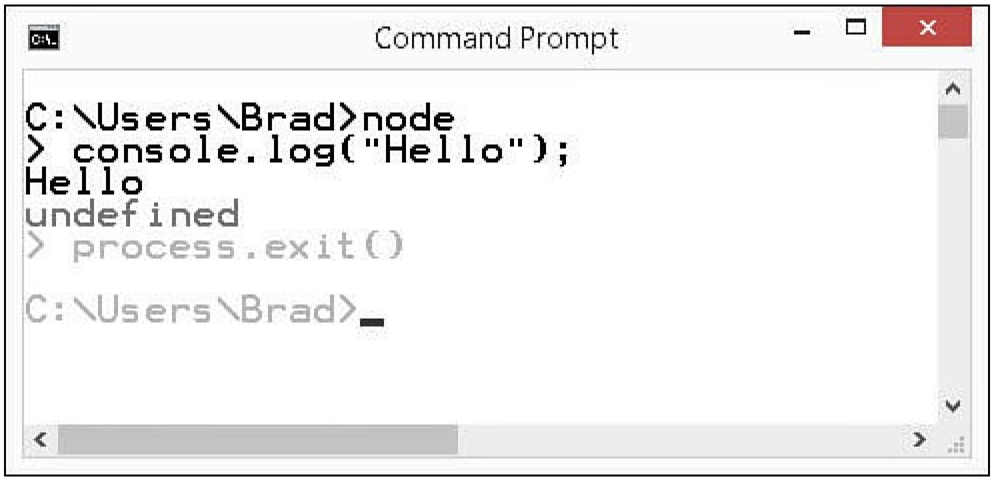
3. Open a terminal or console window.

4. Type `node` to launch the Node.js JavaScript shell, and you should see a > prompt. The Node.js shell provides the capability to execute JavaScript commands directly on the underlying JavaScript engine.

5. If the node application is not found, you need to add the path to the node binary directory to the PATH for your development system (this process is different for each different platform). The binary directory is typically /usr/local/bin/ on Macs and Linux boxes. On Windows, the binary directory will be in the <install>/bin folder, where <install> is the location you specified during the installation process.

6. Then you get to the > prompt. Type the following command and verify that Hello is printed on the screen, as shown in Figure 1.6:

```
console.log("Hello");
```



```
C:\Users\Brad>node
> console.log("Hello");
Hello
undefined
> process.exit()

C:\Users\Brad>_
```

FIGURE 1.6 Starting and using the Node.js command prompt.

7. Use the following command to exit the Node.js prompt:

```
process.exit();
```

You have now successfully installed and configured Node.js.

Configuring Eclipse as a Web Development IDE

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

We chose Eclipse for this book because it is a great general IDE that is easy to configure and set up. You can use your own IDE if you would rather; however, this might be a good chance to try a different IDE if you are unfamiliar with Eclipse.

Note

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

Use the following steps to download, install, and configure Eclipse:

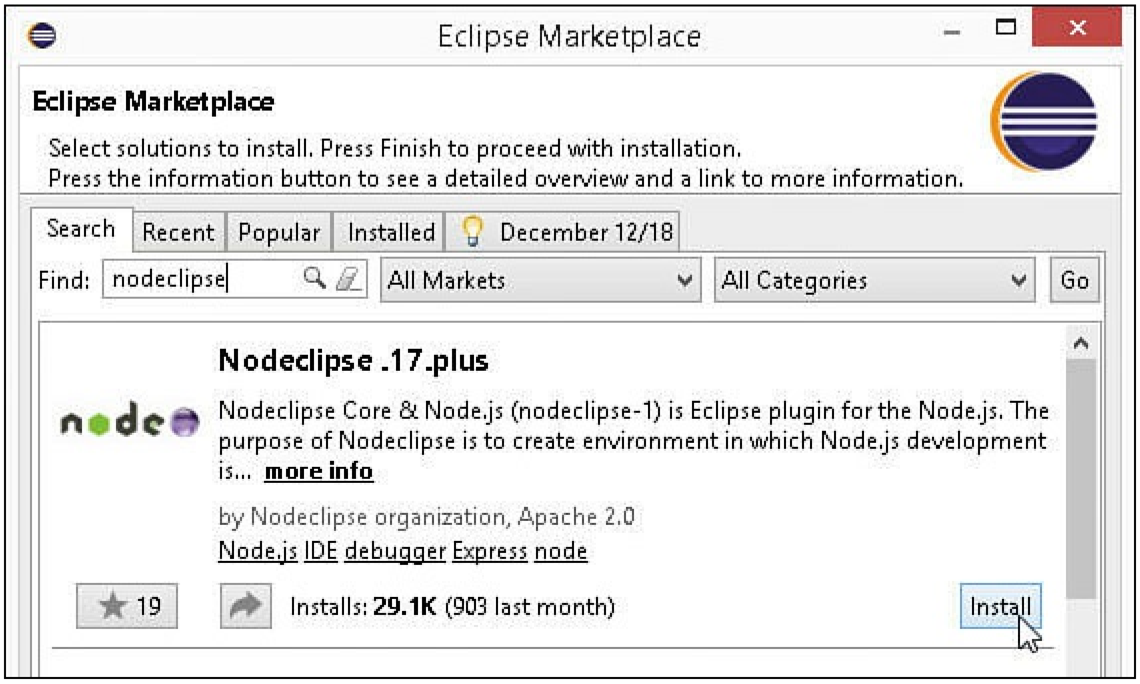
1. Install a Java JRE or JDK. For this book, we downloaded and installed the Java SE Development Kit 8 from the following location:

   http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

2. Download and extract Eclipse. The location you extract the Eclipse files to will be the installation location. For this book, we installed the Luna version Eclipse

IDE for Java Developers from the following location:
http://www.eclipse.org/downloads/

3. Start Eclipse by double-clicking the Eclipse executable file.
4. After Eclipse has loaded, install the Node.js plug-in for Eclipse by selecting Help, Eclipse Marketplace from the main menu. Then type **nodeclipse** into the Find box and click Install to install the package, as shown in Figure 1.7. You will need to accept the license agreement as part of the install process.



Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Nodeclipse .17.plus

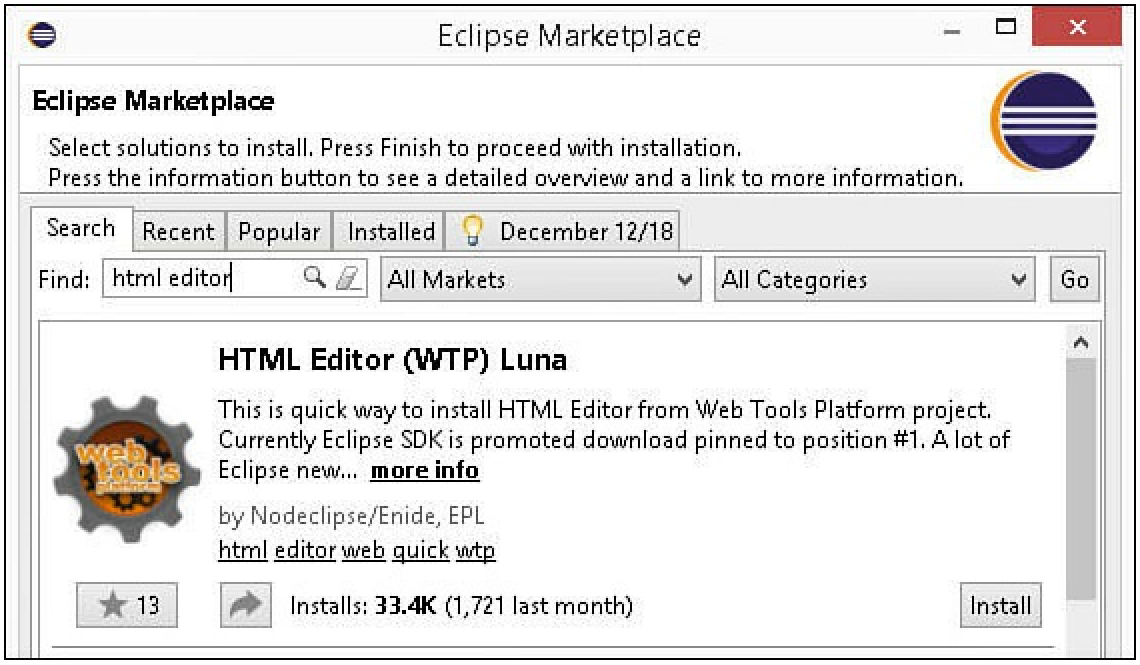
Nodeclipse Core & Node.js (nodeclipse-1) is Eclipse plugin for the Node.js. The purpose of Nodeclipse is to create environment in which Node.js development is... more info

by Nodeclipse organization, Apache 2.0
Node.js IDE debugger Express node

19 Installs: 29.1K (903 last month)

FIGURE 1.7 Installing the Nodeclipse plug-in for Eclipse.

5. After the Nodeclipese plug-in is installed, install the HTML Editor plug-in by selecting Help, Eclipse Marketplace from the main menu. Then type **html editor** into the Find box and click Install to install the package, as shown in Figure 1.8. You will need to accept the license agreement as part of the install process.


Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Finish to proceed with installation.
Press the information button to see a detailed overview and a link to more information.

Search | Recent | Popular | Installed | December 12/18

Find: html editor | All Markets | All Categories | Go

HTML Editor (WTP) Luna

This is quick way to install HTML Editor from Web Tools Platform project. Currently Eclipse SDK is promoted download pinned to position #1. A lot of Eclipse new... **more info**

by Nodeclipse/Enide, EPL

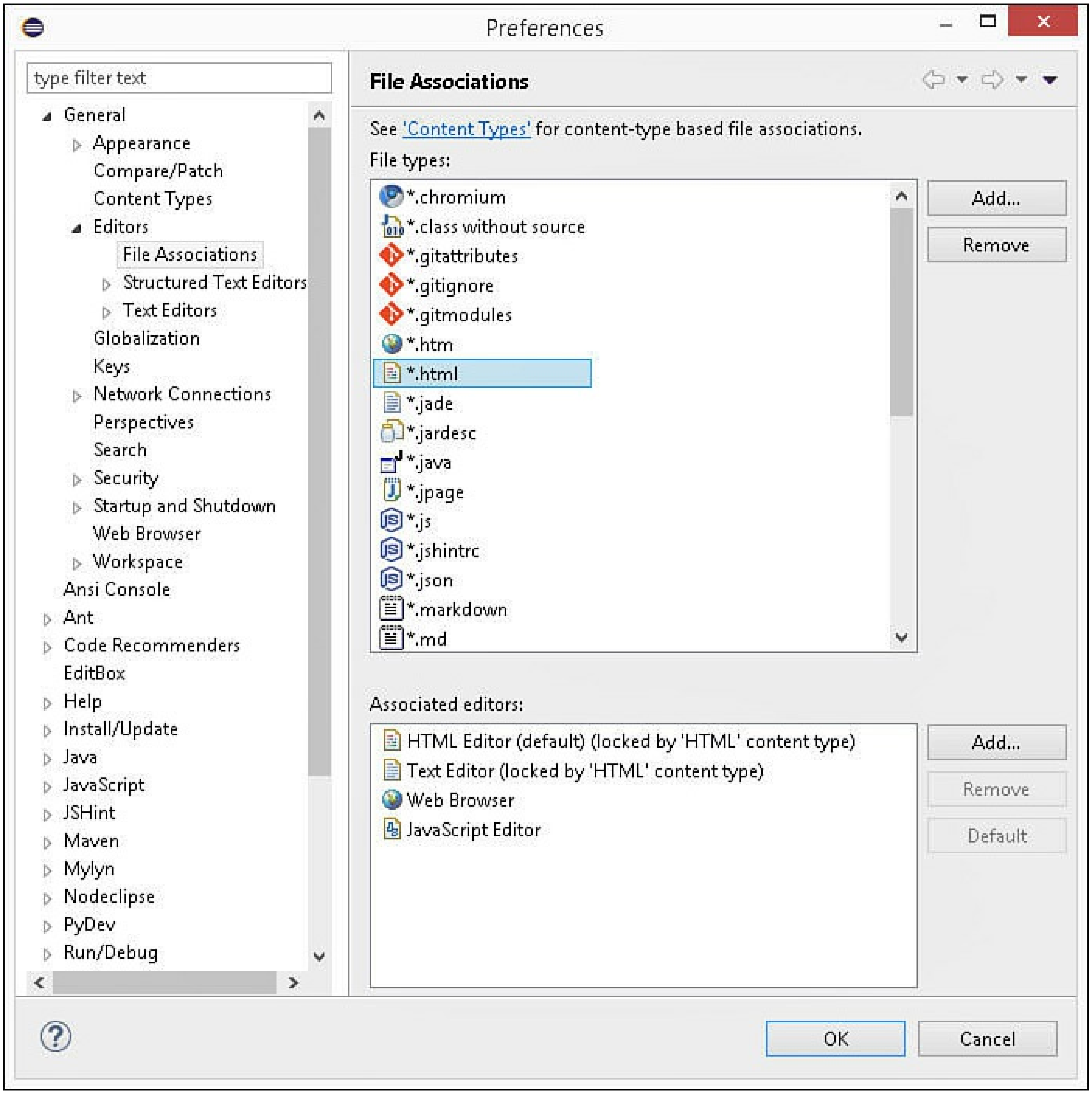
html editor web quick wtp

13 | Installs: **33.4K** (1,721 last month) | Install

FIGURE 1.8 Installing the HTML Editor plug-in for Eclipse.

6. Restart Eclipse to enable the new plug-ins.

7. Verify that the .js extensions uses Nodeclipse and .html extensions use HTML Editor as their default editors. To do this, select Window, Preferences, and then select General, Editors, File Associations and click the file types to see the associated editors, as shown in Figure 1.9.



File Associations

See 'Content Types' for content-type based file associations.

File types:

*.chromium
*.class without source
*.gitattributes
*.gitignore
*.gitmodules
*.htm
*.html
*.jade
*.jardesc
*.java
*.jpage
*.js
*.jshintrc
*.json
*.markdown
*.md

Associated editors:

HTML Editor (default) (locked by 'HTML' content type)
Text Editor (locked by 'HTML' content type)
Web Browser
JavaScript Editor

FIGURE 1.9 Setting default editors for file types in Eclipse.

8. Set the path to the Node.js executable by selecting Window, Preferences and then selecting Nodeclipse in the navigation pane. The Node.js path option is toward the top of the options.
9. Create a project for this book by selecting File, New, Project to launch the New Project Wizard. Then select Node, Node.js Project. Click Next and type in the name of the project; for example, LearningJavaScript. Then click Finish to create the project.
10. Now we'll validate that things work by creating and running a JavaScript

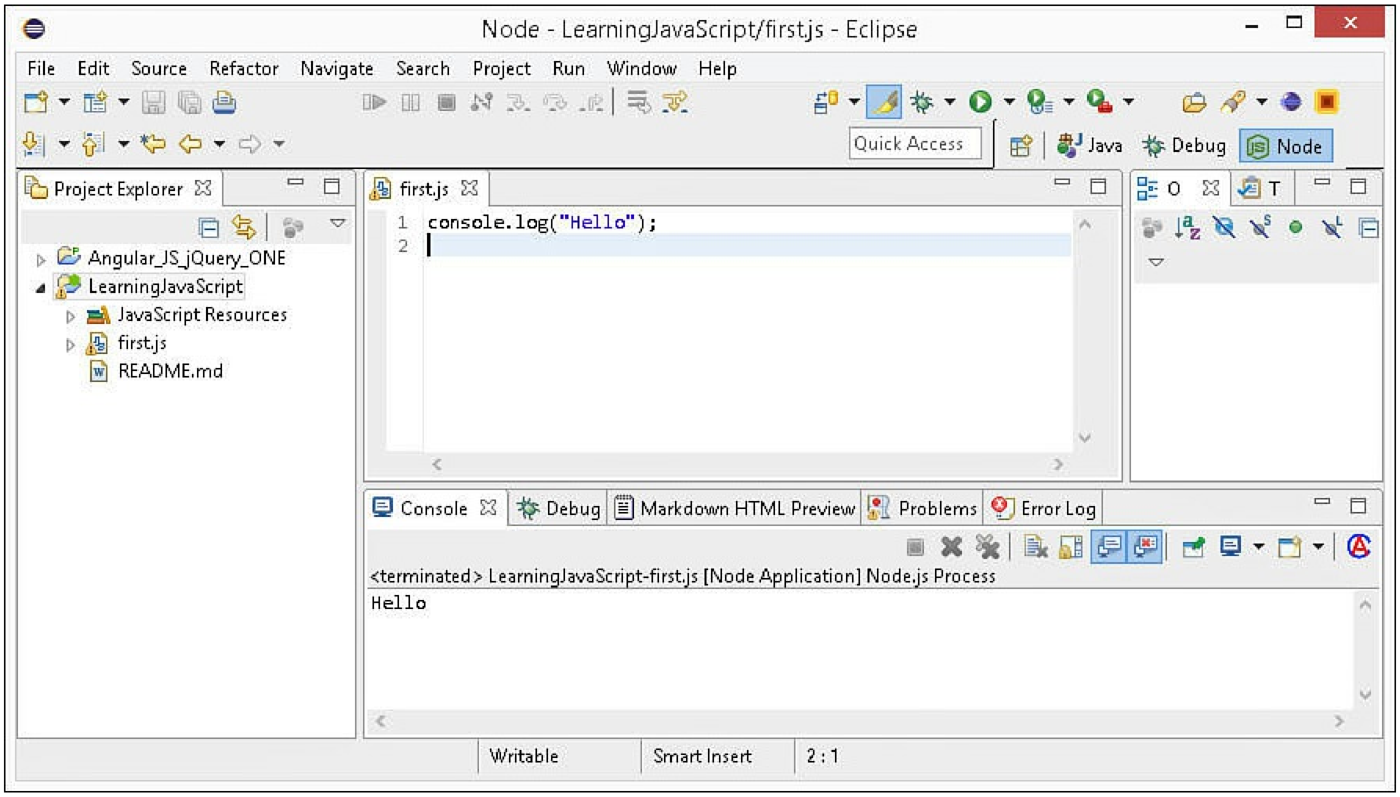
Application from Eclipse. Select the new project and then select File, New JavaScript File from the main menu. Name the file **first.js** and click Finish to create the file.

11. Type the following line of code into the file and save it:

```
console.log("Hello");
```

12. Double-click to the left of line number 2 so that a small circle appears, noting that a breakpoint has been set.

13. Select Run, Run As, Node Application. You should see the word "Hello" printed to the Console window, as shown in Figure 1.10.



```
console.log("Hello");
```

```
<terminated> LearningJavaScript-first.js [Node Application] Node.js Process
Hello
```

FIGURE 1.10 Running a JavaScript application in Eclipse.

Eclipse is now set up and ready for you to begin developing JavaScript.

Note

You can also run JavaScript applications from a console prompt by typing in the command:

node <path_to_JavaScript_file>

Creating an Express Web Server Using Node.js

Node.js is a very modular platform, meaning that Node.js itself provides a very efficient and extensible framework, and external modules are utilized for much of the needed functionality. Consequently, Node.js provides a very nice interface to add and manage these external modules.

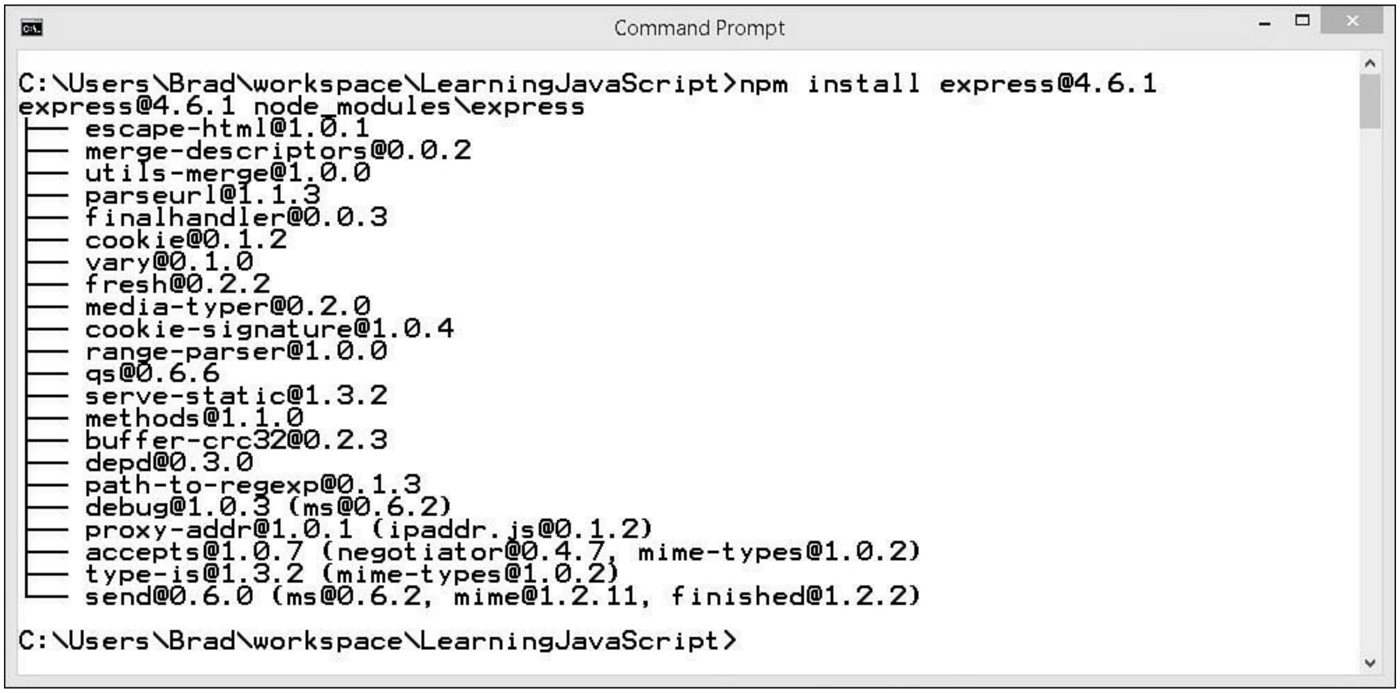
Express is one of these modules. The Express module provides a simple-to-implement web server with a robust feature set, such as static files, routes, cookies, request parsing, and error handling.

The best way to use Node.js as the web server for your web development is to utilize the Express module. In the following exercise, you build a Node.js/Express web server and use it to serve static files.

Use the following steps to build and test a Node.js/Express web server capable of supporting static files and server-side scripting:

1. Open a console prompt and navigate to the location where you created the project folder for this book. If you don't know the path, right-click the project in Eclipse and select Properties from the menu. Then select Resource, and the full path to the project folder is shown in the Location field to the right.
2. From a console prompt in the project folder, execute the following command, as shown in Figure 1.11. This command will install the Express module version 4.6.1 for Node.js into a subfolder named node_modules:

   npm install express@4.6.1



```
C:\Users\Brad\workspace\LearningJavaScript>npm install express@4.6.1
express@4.6.1 node_modules\express
├── escape-html@1.0.1
├── merge-descriptors@0.0.2
├── utils-merge@1.0.0
├── parseurl@1.1.3
├── finalhandler@0.0.3
├── cookie@0.1.2
├── vary@0.1.0
├── fresh@0.2.2
├── media-typer@0.2.0
├── cookie-signature@1.0.4
├── range-parser@1.0.0
├── qs@0.6.6
├── serve-static@1.3.2
├── methods@1.1.0
├── buffer-crc32@0.2.3
├── depd@0.3.0
├── path-to-regexp@0.1.3
├── debug@1.0.3 (ms@0.6.2)
├── proxy-addr@1.0.1 (ipaddr.js@0.1.2)
├── accepts@1.0.7 (negotiator@0.4.7, mime-types@1.0.2)
├── type-is@1.3.2 (mime-types@1.0.2)
└── send@0.6.0 (ms@0.6.2, mime@1.2.11, finished@1.2.2)

C:\Users\Brad\workspace\LearningJavaScript>
```

FIGURE 1.11 Installing the Express npm module for Node.js from a console prompt.

Note

Node occasionally has an issue on some systems where it cannot automatically create a folder to store modules in. You may see an error message similar to the following. If you do, create the npm folder in the path specified, and the npm install command should work:

Click here to view code image

```
Error: ENOENT, s≠tat 'C:\Users\Brad\AppData\Roaming\npm'
node <path_to_JavaScript_file>
```

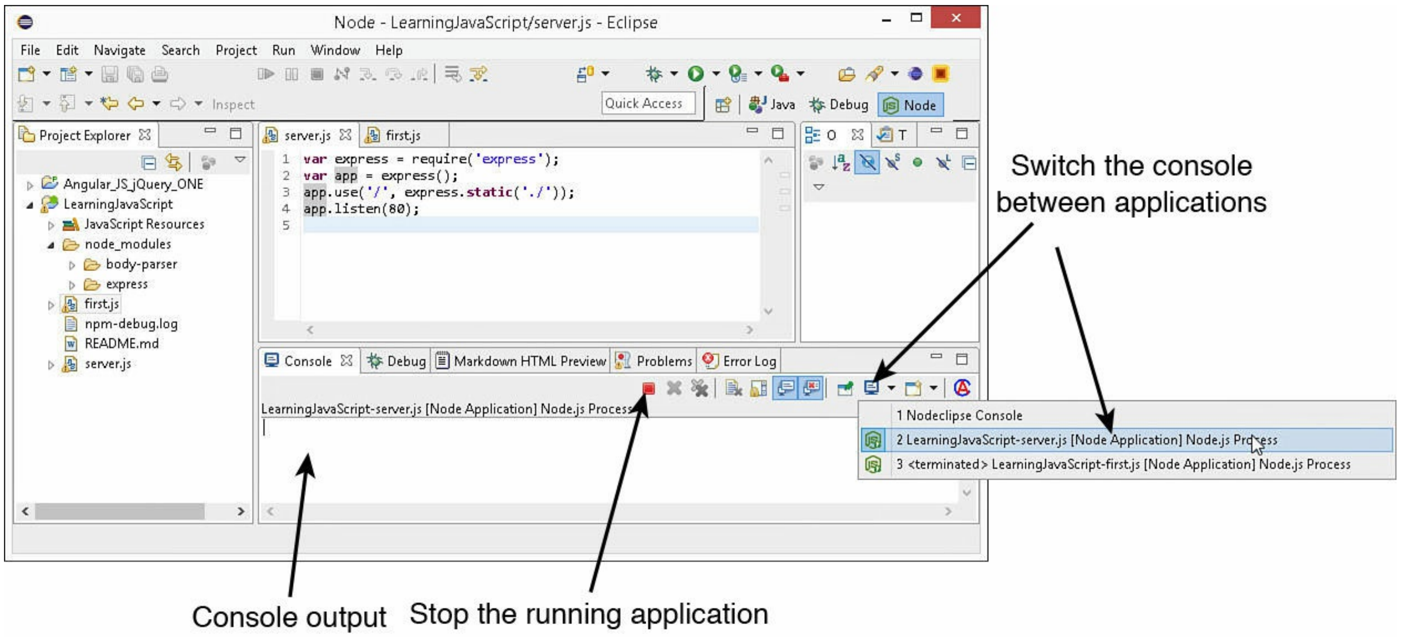
3. Execute the following command to install the body-parser module for Node.js. This module makes it possible to parse the query parameters and body from HTTP GET and POST requests. This command will install the body-parser module version 1.6.5 for Node.js into a subfolder named node_modules:

npm install body-parser@1.6.5

4. Go back to Eclipse, right-click the project, and select Refresh. You should see the node_modules folder with body-parser and express subfolders.

5. Create a file named server.js in the root of your project directory, place the contents from Listing 1.5 inside of it, and save it. This is a basic Node.js/Express web server that will service static files using the root of your project directory as the website root location.

6. Verify that your Node.js web server will run correctly. Start the web server by right-clicking the server.js file and selecting Run As, Node Application from the menu. The Console window, shown in Figure 1.12, should show that the server.js file is running and provide a red box to stop the server. If you are running multiple applications in Eclipse, you can click the Console Select button to select a specific console, as shown in Figure 1.12.



```
var express = require('express');
var app = express();
app.use('/', express.static('./'));
app.listen(80);
```

LearningJavaScript-server.js [Node Application] Node.js Process

1 Nodeclipse Console
2 LearningJavaScript-server.js [Node Application] Node.js Process
3 <terminated> LearningJavaScript-first.js [Node Application] Node.js Process

FIGURE 1.12 Running a JavaScript application in Eclipse.

7. Hit the server from a web browser at the following address. Because the web server is servicing static files using the ./ path, the actual contents of the server.js file should be displayed in the browser:

   localhost/server.js

8. Stop the web server by clicking the red box in the Console window.

Note

If you are not familiar with Eclipse, you should take a minute to practice starting and stopping the server and running the first.js application at the same time to understand starting and stopping applications and how to navigate between them in the console window.

You have now successfully set up a Node.js/Express web server in Eclipse. You will use this web server for most of the examples in the book. A few of the lessons require AJAX interaction, and a separate server will be created for those lessons.

LISTING 1.5 server.js Creating a Basic Node.js/Express Web Server

Click here to view code image

```
var express = require('express');
var app = express();
app.use('/', express.static('./'));
app.listen(80);
```

Try it Yourself: Creating a Dynamic Web Page with jQuery and JavaScript

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

Note

The images and code files for this and all the examples throughout this book can be downloaded from the code archive on Github.

Adding HTML

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

1. Create a folder named lesson01 in your project.
2. Right-click the lesson01 folder that you created.
3. Select New, File from the pop-up menu.
4. Name the file first.html and click OK. A blank document should be opened up for you.
5. Type in the following HTML code. Don't worry if you are not too familiar with HTML; you'll learn enough to use it a bit later in the book:

Click here to view code image

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

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

Click here to view code image

```
http://localhost/lesson01/first.html
```

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

Adding CSS

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

Use the following steps to add the CSS style to the `<span>` element. For reference, the style changes you make in these steps are shown in the final script in Listing 1.6:

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

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

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

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

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

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

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

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

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

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

6. Save the file.

7. Open the following URL in your web browser, and you should see the stylized text Click Me appear, as shown in Figure 1.13:

Click here to view code image

```
http://localhost/lesson01/first.html
```

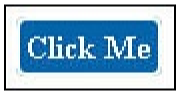


FIGURE 1.13 `<span>` element stylized to look like a button.

Writing a Dynamic Script

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

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

1. Change the `<span>` element to include an ID so that you can reference it, and also add a handler for the `mouseover` event, as shown in line 30 of Listing 1.6:

Click here to view code image

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

2. Add the following line of code to the `<head>` tag, as shown in line 6 of Listing 1.6. This loads the jQuery library from a hosted source:

Click here to view code image

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
```

3. Add the following JavaScript function to the `<head>`, as shown in lines 6–13 of Listing 1.6. This function creates an array of coordinate values from 10 to 350, then randomly sets the top and left CSS properties of the span element each time the mouse is moved over it:

Click here to view code image

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

4. Save the file.
5. Open the following URL in your web browser, and you should see the stylized text Click Me appear, as shown in Figure 1.13:

Click here to view code image

```
http://localhost/lesson01/first.html
```

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

LISTING 1.6 A Simple Interactive jQuery and JavaScript Web Page

Click here to view code image

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <meta charset="utf-8" />
05     <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
06     <script>
07       function moveIt(){
    var coords = new Array(10,50,100,130,175,225,260,300,320,350);
    var x = coords[Math.floor((Math.random()*10))];
    var y = coords[Math.floor((Math.random()*10))];
    $("#elusiveText").css({"top": x + "px", "left": y + "px"})
   }
  </script>
  <style>
   span{
    background-color: #0066AA;
    color: #FFFFFF;
    font-weight: bold;
    border-color: #C0C0C0;
    border:2px solid;
    border-radius:5px;
    padding: 3px;
    position:absolute;
    top:150px;
    left:100px;
   }
  </style>
 </head>
 <body>
  <span id="elusiveText" onmouseover="moveIt()">Click Me</span>
 </body>
</html>
```

Summary

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

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

Q&A

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

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

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

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

Workshop

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

Quiz

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

Quiz Answers

1. GET
2. Client-side
3. False. You must manually enable JavaScript debugging on all browsers. Pressing F12 in most browsers will launch the Developer Tools that allow you to debug JavaScript.
4. CSS scripts are the simplest to use when defining the appearance of DOM elements.

Exercises

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

Click here to view code image

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

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

Click here to view code image

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

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

Click here to view code image

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

Lesson 2. Debugging JavaScript in Web Pages

What You'll Learn in This Lesson:

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

A major challenge when writing JavaScript applications is finding and fixing problems in your scripts. Simple syntax problems or invalid values can cause a lot of frustration and wasted time. For that reason, some excellent tools have been created to help you quickly and easily find problems in your scripts. In this lesson, you learn some of the basics of debugging JavaScript via the browser developer tools.

This lesson uses the Chrome developer tools to illustrate how to debug JavaScript in web pages. If you are already familiar with using the developer tools to debug HTML, CSS, and JavaScript, you can probably skip this lesson and move on to the next.

Although the developer tools console in Chrome, IE, and Firefox are a bit different, most of the principles are the same across browsers, and you should be able to apply the concepts equally. Also, don't be alarmed if you don't recognize the code element in the examples. They'll be covered in upcoming lessons, but you should be able to debug before you jump into coding heavily.

Viewing the Developer Tools Console

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

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

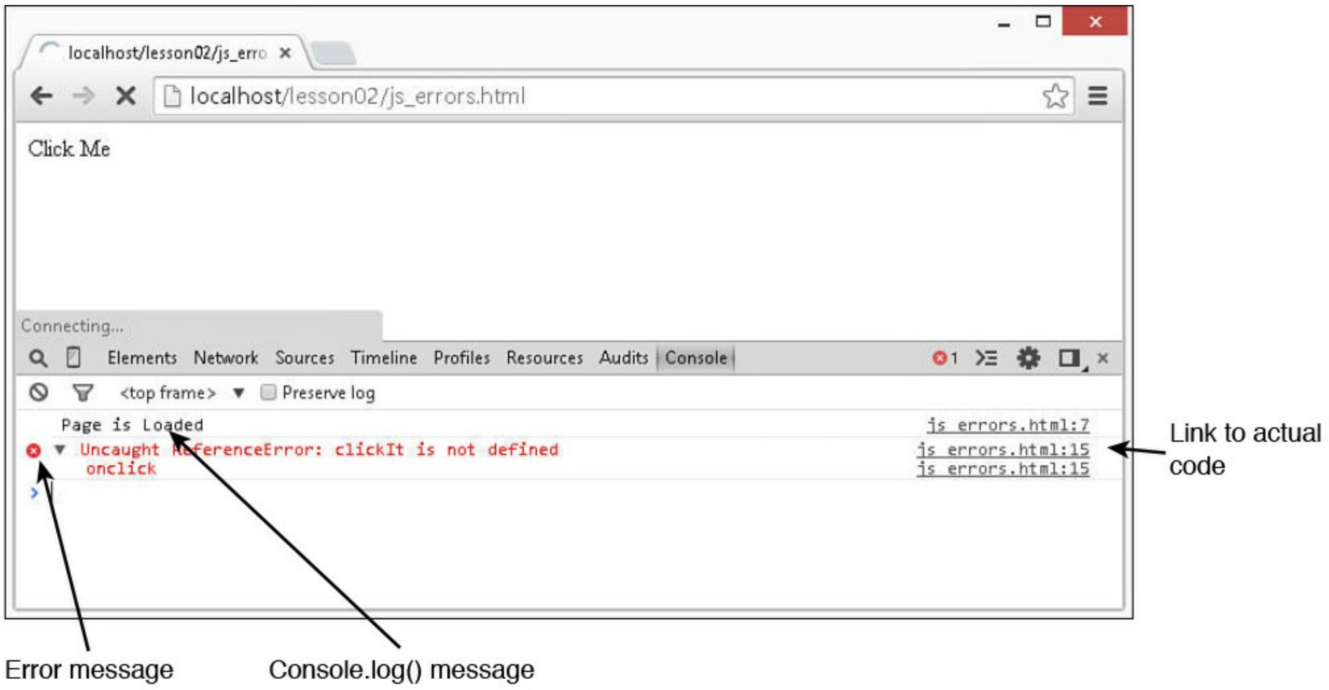
The developer tools can be accessed in Chrome using F12 or Ctrl+Shift+i on Windows or Cmd-Shift-i on Macs.

Note

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

Understanding the Browser Developer Tools Console

When you open the developer tools, you see a series of tabs. The console tab, shown in Figure 2.1, acts as the interface to your JavaScript output. `console.log()` and error messages are displayed here.


localhost/lesson02/js_erro
localhost/lesson02/js_errors.html
Click Me
Connecting...
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
Preserve log
Link to actual code
Error message
Console.log() message

```
Page is Loaded                                    js_errors.html:7
Uncaught ReferenceError: clickIt is not defined   js_errors.html:15
  onclick                                         js_errors.html:15
```

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

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

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

Try it Yourself: Using the JavaScript Console to Find Errors

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

1. Right-click the project and select New, Folder from the menu.
2. Name the folder lesson02 and click Finish.

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

LISTING 2.1 js_errors.html A Very Simple HTML Document with JavaScript Errors

Click here to view code image

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

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

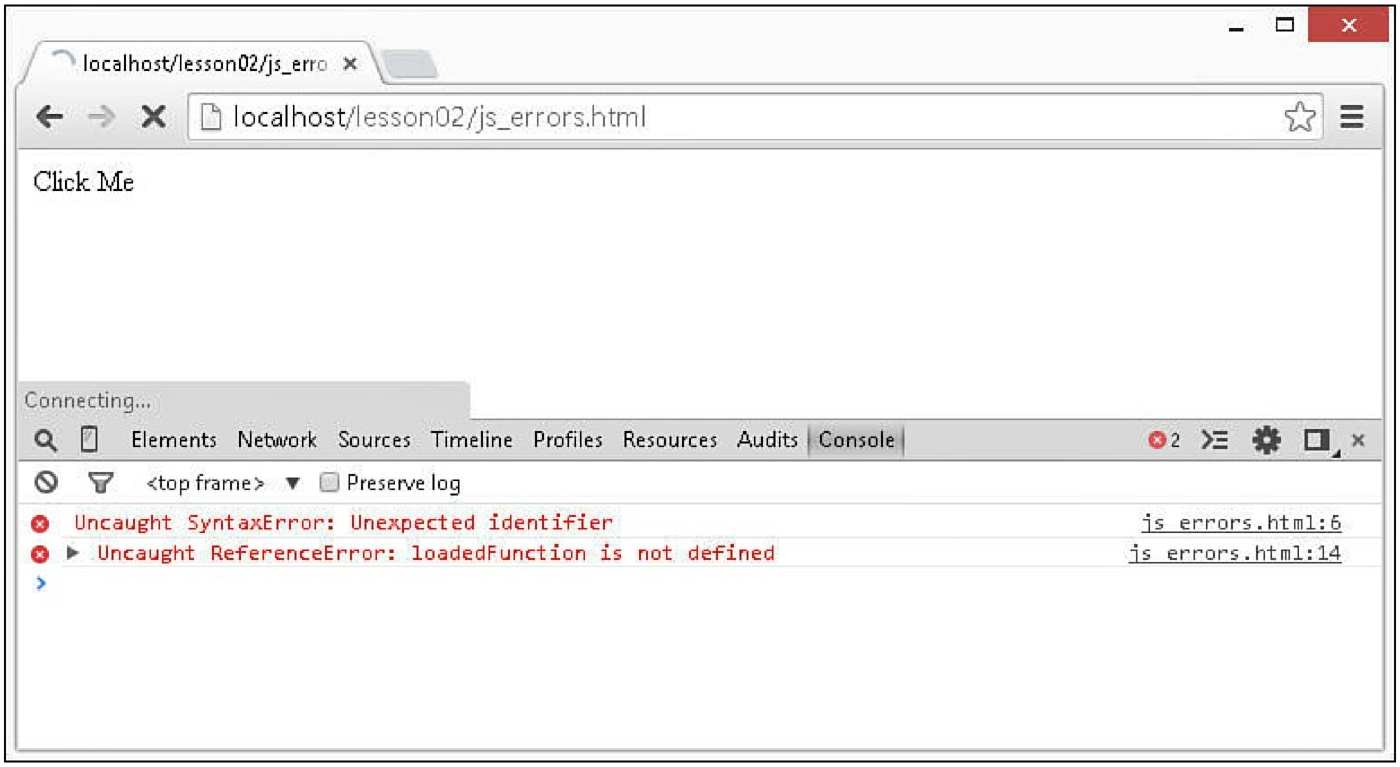
With the file now in place, use the following steps to debug the errors using the JavaScript console:

1. Open Chrome and load the developer tools.
2. Open the following URL in Chrome to load the newly created web page:

Click here to view code image

```
http://localhost/lesson02/js_errors.html
```

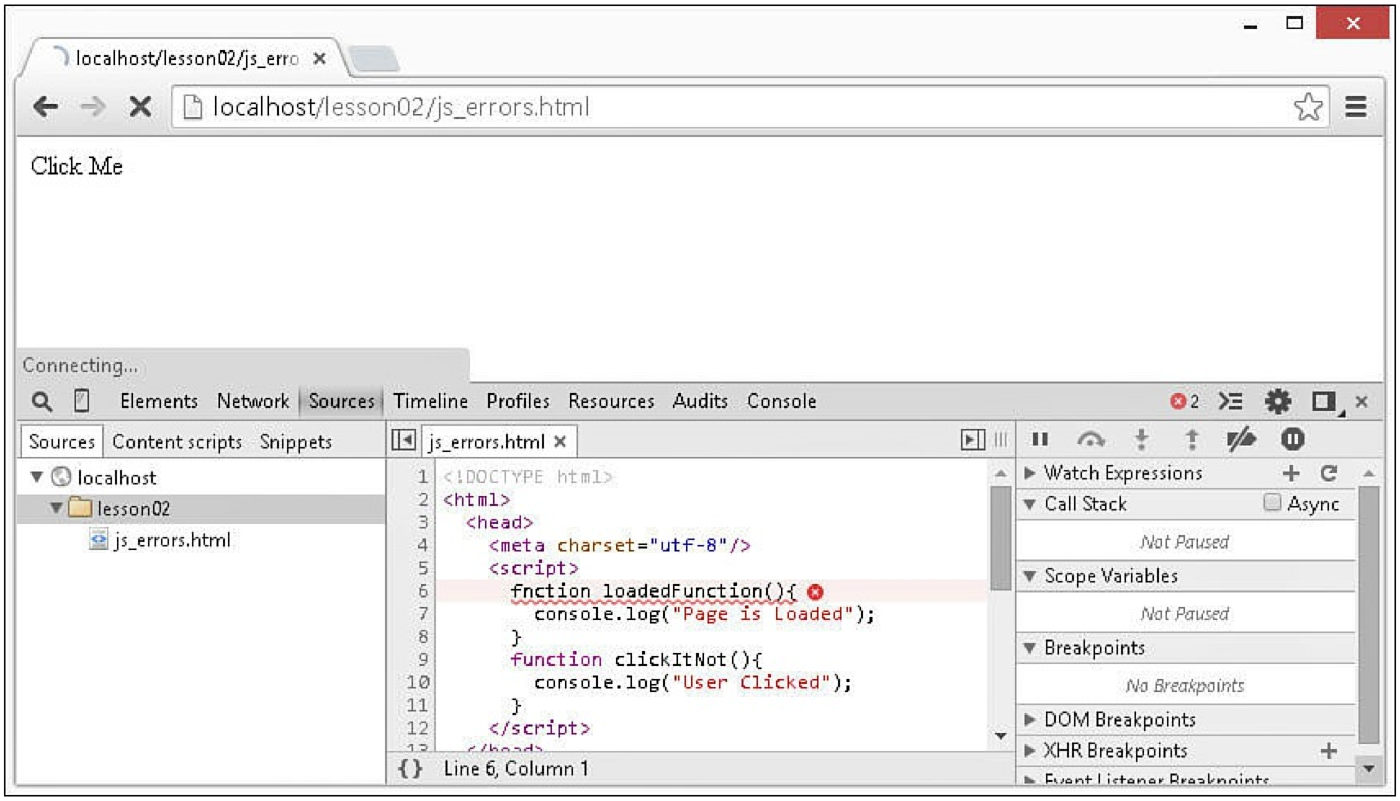
3. Click the Console tab to bring up the JavaScript console shown in Figure 2.2.


localhost/lesson02/js_erro
localhost/lesson02/js_errors.html
Click Me
Connecting...
Elements Network Sources Timeline Profiles Resources Audits Console
2
<top frame>
Preserve log

```
Uncaught SyntaxError: Unexpected identifier                          js_errors.html:6
▶ Uncaught ReferenceError: loadedFunction is not defined             js_errors.html:14
```

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

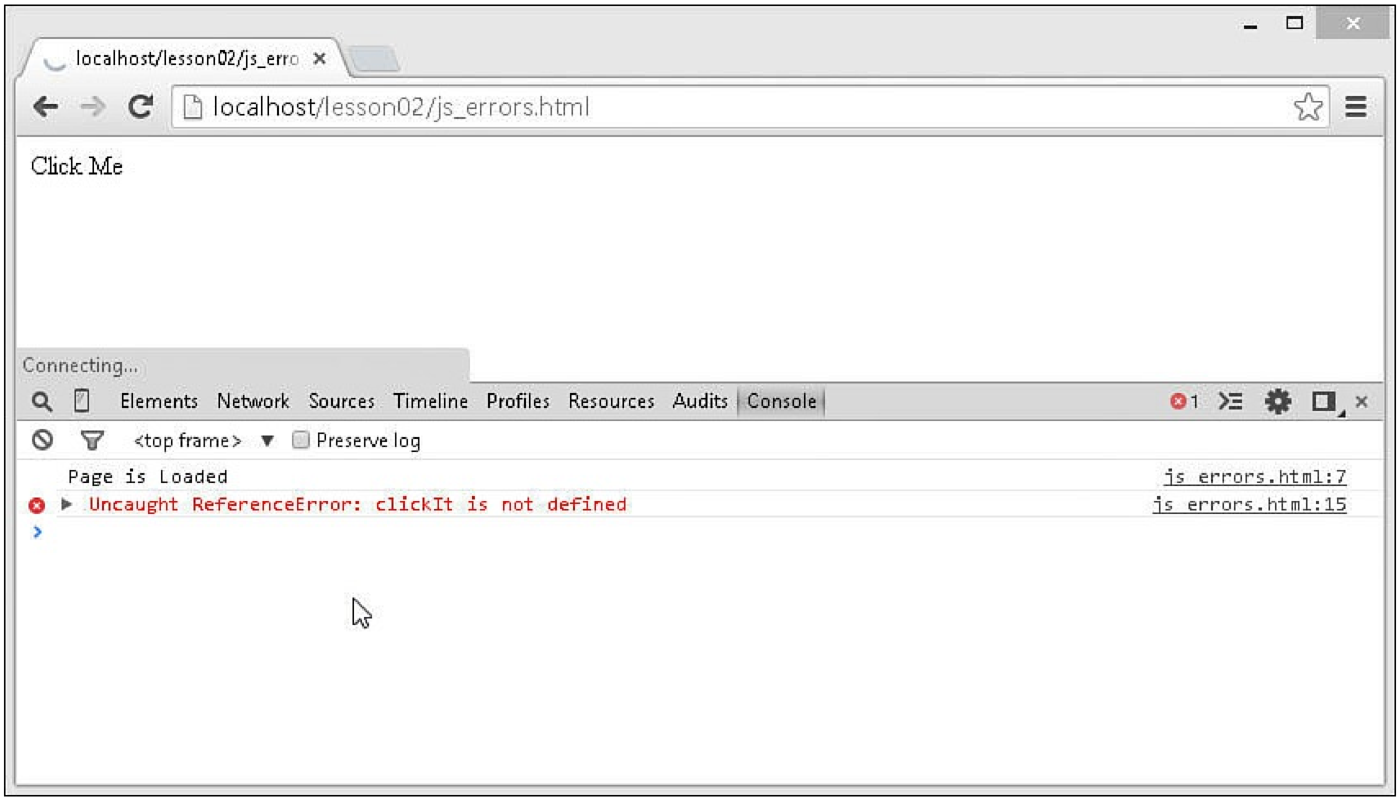
4. Notice the errors displayed in the console, as shown in Figure 2.2. The first error shows that there is an unexpected identifier on line 6 in the file. Clicking the line number brings up the Sources tab at line 6 and you can see that `function` is misspelled as `fnction`, as shown in Figure 2.3.



```
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <meta charset="utf-8"/>
5       <script>
6         fnction loadedFunction(){
7           console.log("Page is Loaded");
8         }
9         function clickItNot(){
10          console.log("User Clicked");
11        }
12      </script>
```

FIGURE 2.3 The Sources console showing a syntax error in the JavaScript code.

5. In Eclipse, change the word `fnction` in line 6 to `function`.
6. Go back to Chrome and refresh the web page. Now in the Console, you should see Page Is Loaded in the Console, the text that is logged in the `loadedFunction()` function, but no errors.
7. Click the Click Me text. An error is added to the console, as shown in Figure 2.4. The error states that `clickItNow` is not defined. When you look at the HTML file and search for `clickItw`, you can see on line 15 that an `onclick` event is linked to `clickItNot()`, but that the JavaScript function is named `clickItNow()`.


localhost/lesson02/js_erro
localhost/lesson02/js_errors.html
Click Me
Connecting...
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame> Preserve log

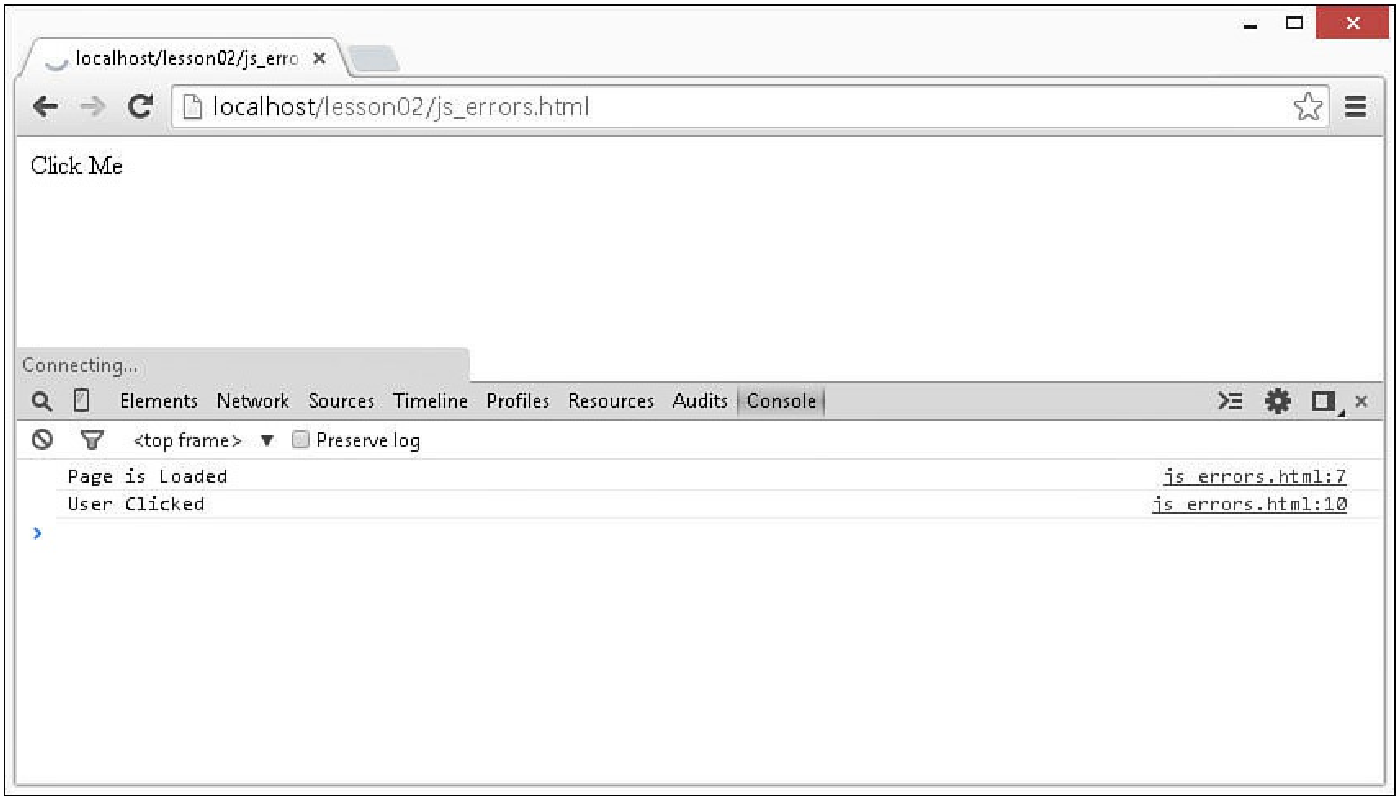
```
Page is Loaded                                        js_errors.html:7
Uncaught ReferenceError: clickIt is not defined       js_errors.html:15
```

FIGURE 2.4 The JavaScript console showing one successful log message and one error.

8. In Eclipse, change `clickItNow` in line 15 to `clickIt` and save the file.

9. Reload the web page.

10. Click the Click Me Text again. Figure 2.5 shows that both log statements are now displayed correctly and there are no errors. The page has been successfully debugged.


localhost/lesson02/js_erro
localhost/lesson02/js_errors.html
Click Me
Connecting...
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
Preserve log

```
Page is Loaded                                          js_errors.html:7
User Clicked                                            js_errors.html:10
>
```

FIGURE 2.5 The JavaScript console showing two successful log messages and no errors.

Debugging HTML Elements

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

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

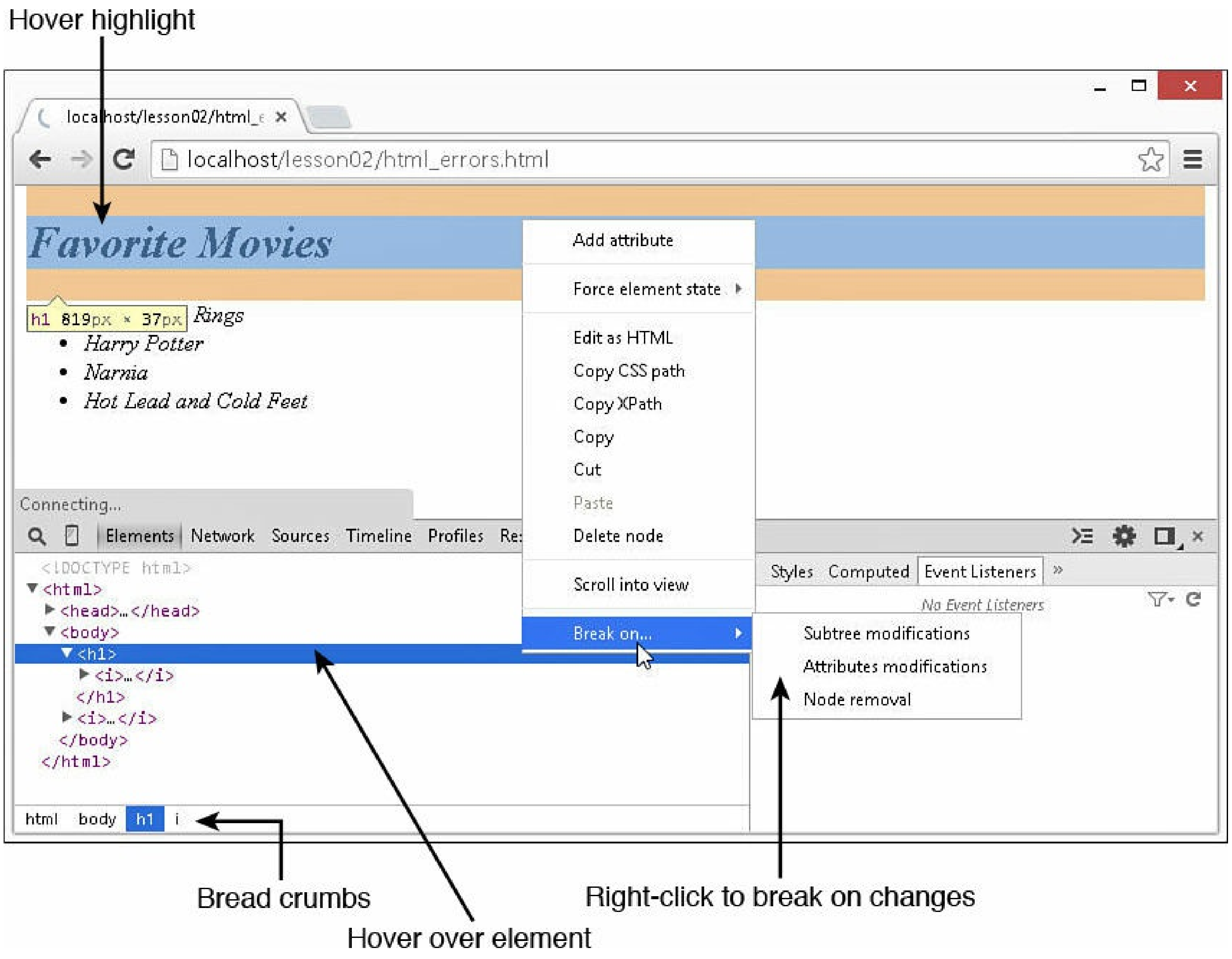
Inspecting HTML Elements

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

Figure 2.6 shows an example of the Element inspector. With the Element inspector, some very useful features are available to you, as described next:

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

- **Break on Changes**—Right-clicking an element and selecting Break On allows you to set different types of breaks for when the element is changed, removed, or sub elements are modified. The browser will break into the JavaScript debugger whenever the DOM element is changed dynamically. This helps you catch problems as they are occurring.
- **Edit as HTML**—Right-clicking an element and selecting Edit as HTML allows you to directly edit the HTML code in the browser. The browser changes what is rendered based on the changes you make here. Although this won't change the code in your project, it is much easier to use this feature to try things out until problems are fixed. Then you can copy the code from the editor and paste it into the actual file in your project.
- **Hover**—When you hover over the HTML code in the DOM tree, the element is highlighted in the browser. The hover feature of the Element inspector is one of my favorites because it gives a visual way to see the relationship between the node in the DOM tree and the rendered web page. Notice in Figure 2.6 that as the `<h1>` element hovered over, the heading is highlighted in the web page.



```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<h1>
      ▶<i>…</i>
     </h1>
    ▶<i>…</i>
   </body>
</html>
```

FIGURE 2.6 The Element inspector page in Chrome.

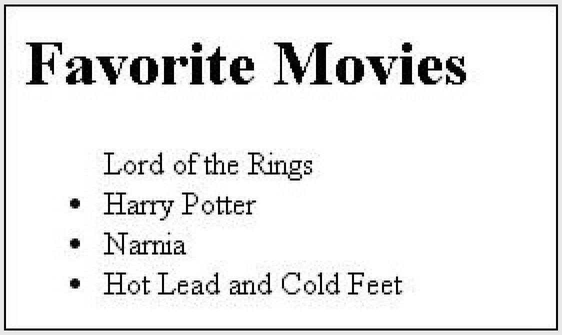
Note

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

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

Try it Yourself: Debugging HTML Using the Element Inspector

To illustrate how to use the Element inspector, consider the code in Listing 2.2. A basic HTML document with a list of movies and the word "Favorite" in the heading is supposed to be in italic. However, look at the rendered version in Figure 2.7. There are obviously some problems: Everything is in italic and there is no bullet point on the first list item. These problems are caused by just two characters in all the text.


Favorite Movies

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

FIGURE 2.7 This web page has two problems: Only the word "Favorite" should be in italic, and there is no bullet point on the first list item.

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

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
  </head>
  <body>
```

```
    <h1><i></i>Favorite Movies</h1>
    <ul>
      <ll>Lord of the Rings</li>
      <li>Harry Potter</li>
      <li>Narnia</li>
      <li>Hot Lead and Cold Feet</li>
    </ul>
  </body>
</html>
```

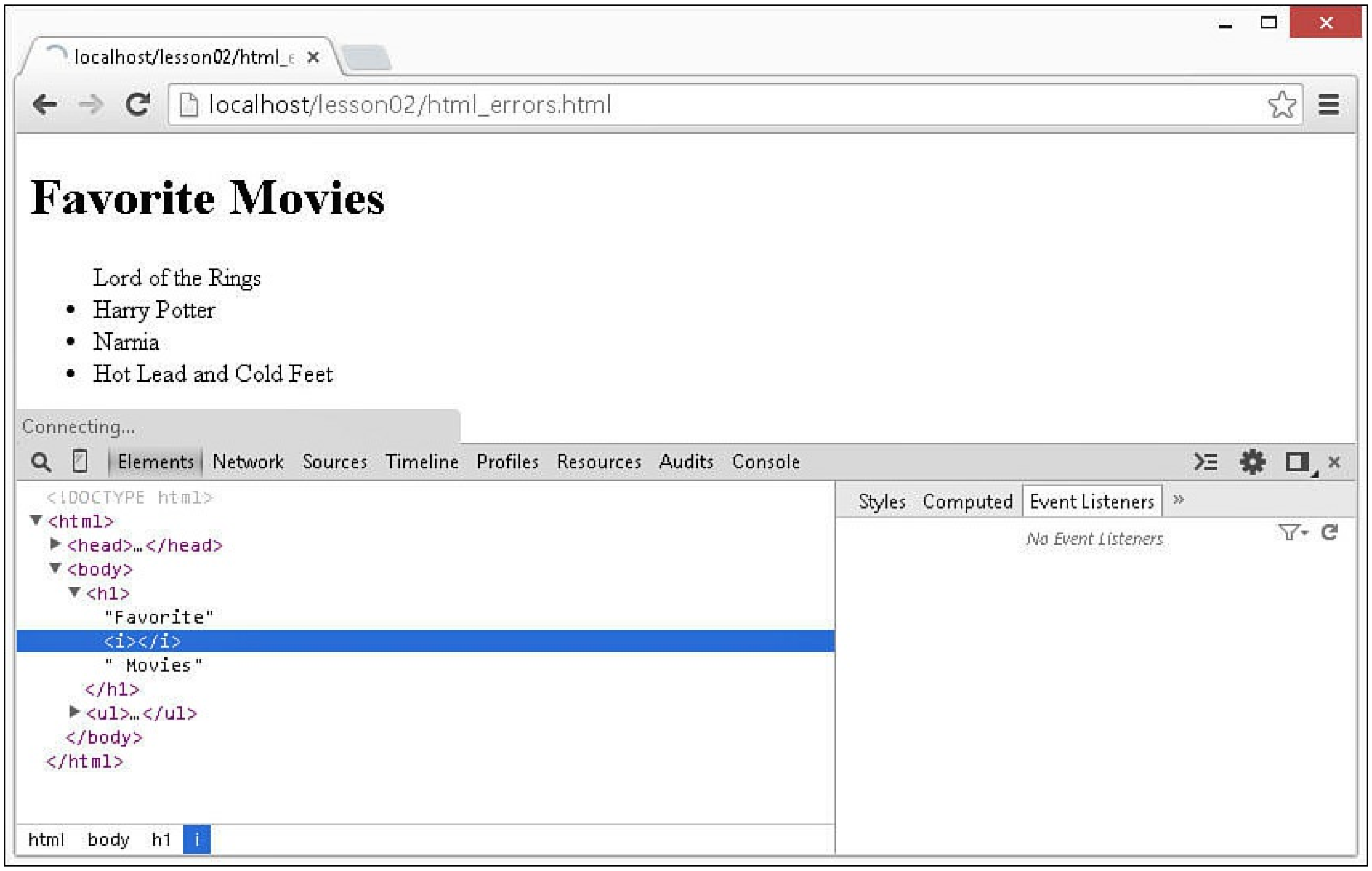
Follow along with these steps to find and fix the HTML syntax problems using the Element inspector:

1. Add the code in Listing 2.2 to a new file html_errors.html in the lesson02 folder of your project and save the document. You should be familiar with this process by now.
2. Open Chrome and load the developer tools.
3. Open the following URL; the web page should look like Figure 2.7:

Click here to view code image

```
http://localhost/lesson02/html_errors.html
```

4. Click the Elements tab in the developer tools and expand the `<html>`, then `<body>`, and then `<i>` tags, as shown in Figure 2.8. Notice that the word “Favorite” is not contained within the `<i>` tag. That isn’t right.


localhost/lesson02/html_e
localhost/lesson02/html_errors.html

Favorite Movies

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

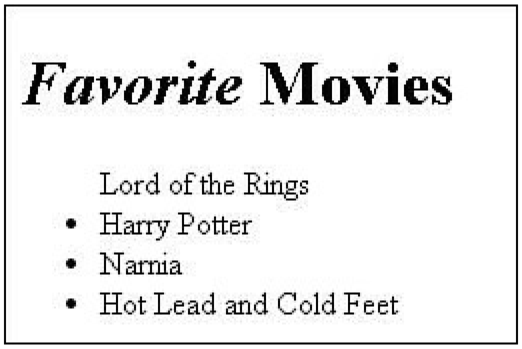
Connecting...
Elements Network Sources Timeline Profiles Resources Audits Console
Styles Computed Event Listeners
No Event Listeners

```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<h1>
       "Favorite"
       <i></i>
       " Movies"
     </h1>
    ▶<ul>…</ul>
   </body>
</html>
```

FIGURE 2.8 This Element inspector shows a second `<i>` in the DOM.

5. Go back to Eclipse and move the word “Favorite” inside the `<i>` tags on line 7 and save the document.

6. Refresh the document in the browser. Notice that the word “Favorite” is now in italic, as it should be, but the bullet point is still missing, as shown in Figure 2.9.

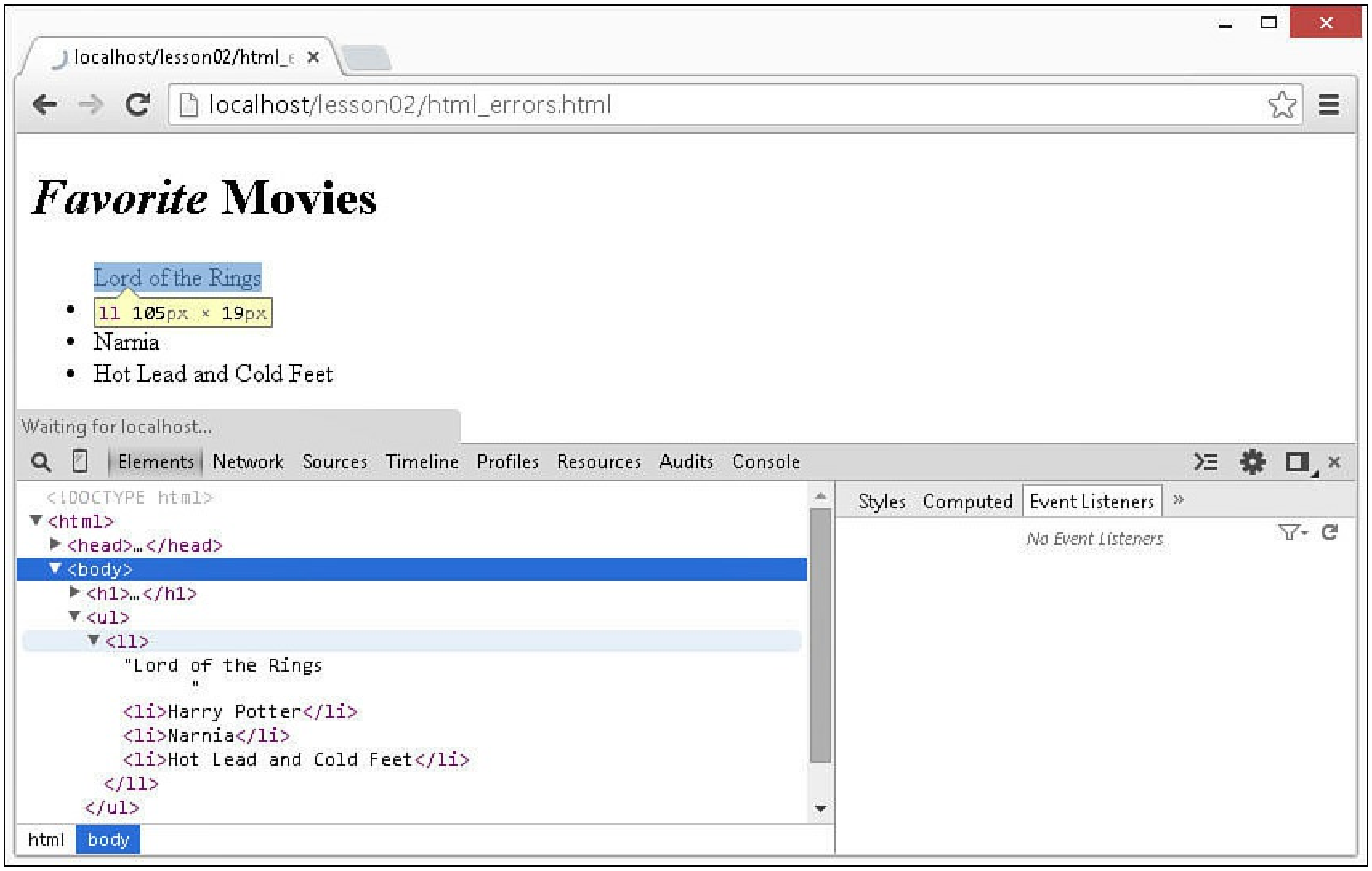


Favorite Movies

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

FIGURE 2.9 This web page now has only one problem—no bullet point on the first list item.

7. Go back to the Element inspector and expand the `<html>`, then `<body>`, then `<ul>`, then `<ll>`, as shown in Figure 2.10. Instead of a set of four `<li>` elements under the `<ul>` element, there is an `<ll>` element with the `<li>` elements underneath. We haven’t covered the HTML tags yet, but if you are familiar with HTML lists, you will recognize that `ll` is not a valid HTML tag. It should be `<li>`.



Favorite Movies

Lord of the Rings

ll 105px × 19px

- Narnia
- Hot Lead and Cold Feet

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

FIGURE 2.10 Viewing the DOM reveals that the browser sees an `<ll>` tag under the `<ul>` tag, not a set of `<li>` tags.

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

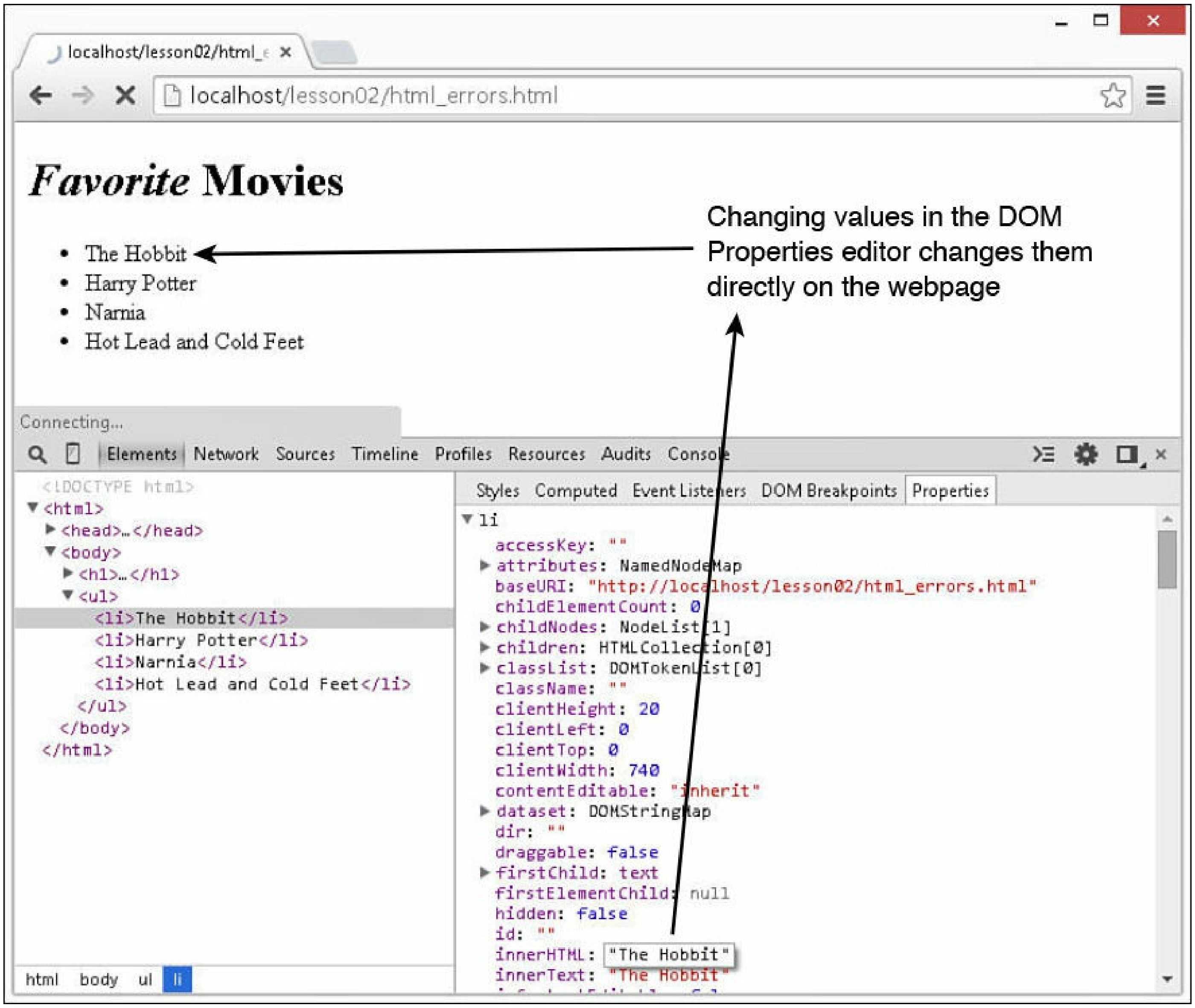
9. Reload the web page in the browser. The list is now displayed properly.

Viewing and Editing the DOM Properties of Elements

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

The DOM editor can be accessed by clicking the Elements tab in the Developer Tools console and selecting the Properties sub tab. You can launch the Elements tab from the web page by right-clicking an element and selecting Inspect Element from the pop-up menu.

Figure 2.11 shows the DOM properties editor. From the DOM properties editor, you have access to a variety of information about the browser environment. For example, in Figure 2.11, the `innerHTML` attribute of the window object is displayed.


localhost/lesson02/html_e
localhost/lesson02/html_errors.html
Changing values in the DOM Properties editor changes them directly on the webpage
Connecting...
Elements Network Sources Timeline Profiles Resources Audits Console
Styles Computed Event Listeners DOM Breakpoints Properties
html body ul li

Favorite Movies

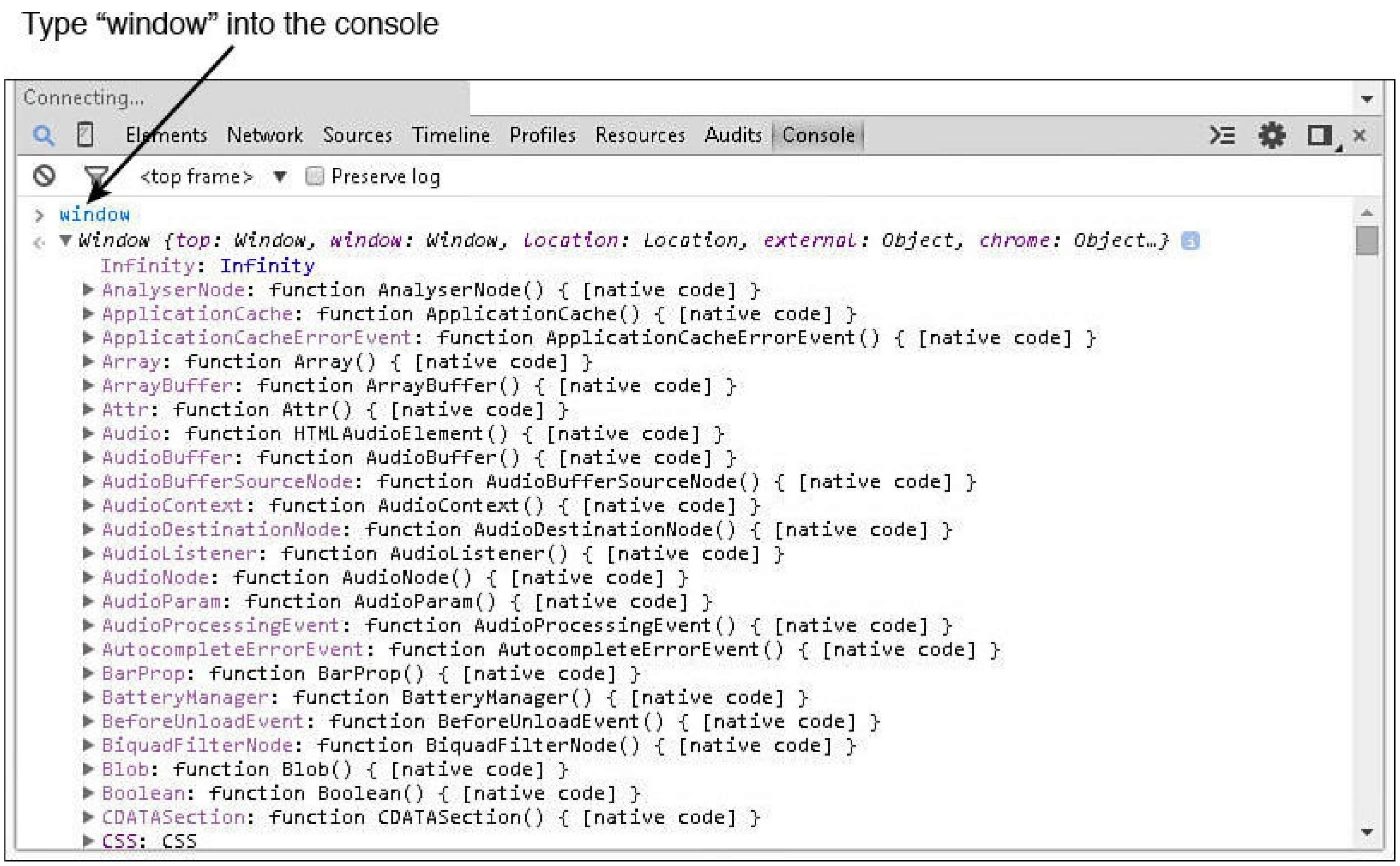
- The Hobbit
- Harry Potter
- Narnia
- Hot Lead and Cold Feet

```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▶<h1>…</h1>
    ▼<ul>
        <li>The Hobbit</li>
        <li>Harry Potter</li>
        <li>Narnia</li>
        <li>Hot Lead and Cold Feet</li>
      </ul>
    </body>
  </html>
```

```
▼ li
    accessKey: ""
  ▶ attributes: NamedNodeMap
    baseURI: "http://localhost/lesson02/html_errors.html"
    childElementCount: 0
  ▶ childNodes: NodeList[1]
  ▶ children: HTMLCollection[0]
  ▶ classList: DOMTokenList[0]
    className: ""
    clientHeight: 20
    clientLeft: 0
    clientTop: 0
    clientWidth: 740
    contentEditable: "inherit"
  ▶ dataset: DOMStringMap
    dir: ""
    draggable: false
  ▶ firstChild: text
    firstElementChild: null
    hidden: false
    id: ""
    innerHTML: "The Hobbit"
    innerText: "The Hobbit"
```

FIGURE 2.11 The main DOM Properties editor tab in Chrome.

You can also view a DOM properties editor for DOM objects in the Console tab by typing in the name of the object. Typically we've used this only to access the browser window object to get information, such as the screen dimensions and such. To do this in the JavaScript Console view in Chrome, type the word **window** directly into the editor, as shown in Figure 2.12.


Type "window" into the console
Connecting...
Elements Network Sources Timeline Profiles Resources Audits Console
<top frame>
Preserve log

```
> window
< ▼Window {top: Window, window: Window, location: Location, external: Object, chrome: Object…}
    Infinity: Infinity
  ▶ AnalyserNode: function AnalyserNode() { [native code] }
  ▶ ApplicationCache: function ApplicationCache() { [native code] }
  ▶ ApplicationCacheErrorEvent: function ApplicationCacheErrorEvent() { [native code] }
  ▶ Array: function Array() { [native code] }
  ▶ ArrayBuffer: function ArrayBuffer() { [native code] }
  ▶ Attr: function Attr() { [native code] }
  ▶ Audio: function HTMLAudioElement() { [native code] }
  ▶ AudioBuffer: function AudioBuffer() { [native code] }
  ▶ AudioBufferSourceNode: function AudioBufferSourceNode() { [native code] }
  ▶ AudioContext: function AudioContext() { [native code] }
  ▶ AudioDestinationNode: function AudioDestinationNode() { [native code] }
  ▶ AudioListener: function AudioListener() { [native code] }
  ▶ AudioNode: function AudioNode() { [native code] }
  ▶ AudioParam: function AudioParam() { [native code] }
  ▶ AudioProcessingEvent: function AudioProcessingEvent() { [native code] }
  ▶ AutocompleteErrorEvent: function AutocompleteErrorEvent() { [native code] }
  ▶ BarProp: function BarProp() { [native code] }
  ▶ BatteryManager: function BatteryManager() { [native code] }
  ▶ BeforeUnloadEvent: function BeforeUnloadEvent() { [native code] }
  ▶ BiquadFilterNode: function BiquadFilterNode() { [native code] }
  ▶ Blob: function Blob() { [native code] }
  ▶ Boolean: function Boolean() { [native code] }
  ▶ CDATASection: function CDATASection() { [native code] }
  ▶ CSS: CSS
```

FIGURE 2.12 Editing HTML elements inside the DOM editor.

Try it Yourself: Editing HTML Element Values in the DOM Editor

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

1. Open the fixed code in file html_errors.html in Chrome and open the Developer Tools console.
2. Click the Elements tab.
3. Expand the `<html>`, `<body>`, and `<ul>` nodes in the Element inspector.
4. Select the first `<li>` node.
5. Click the Properties tab to the right, as shown in Figure 2.11.
6. Scroll down and find the `innerHTML` property in the DOM properties editor.
7. Double-click the value to the right of the `innerHTML` attribute and change the text as shown in Figure 2.11. Notice that the HTML element rendered in the web page also changes. It is as easy as that to manipulate any editable attribute of your HTML nodes.

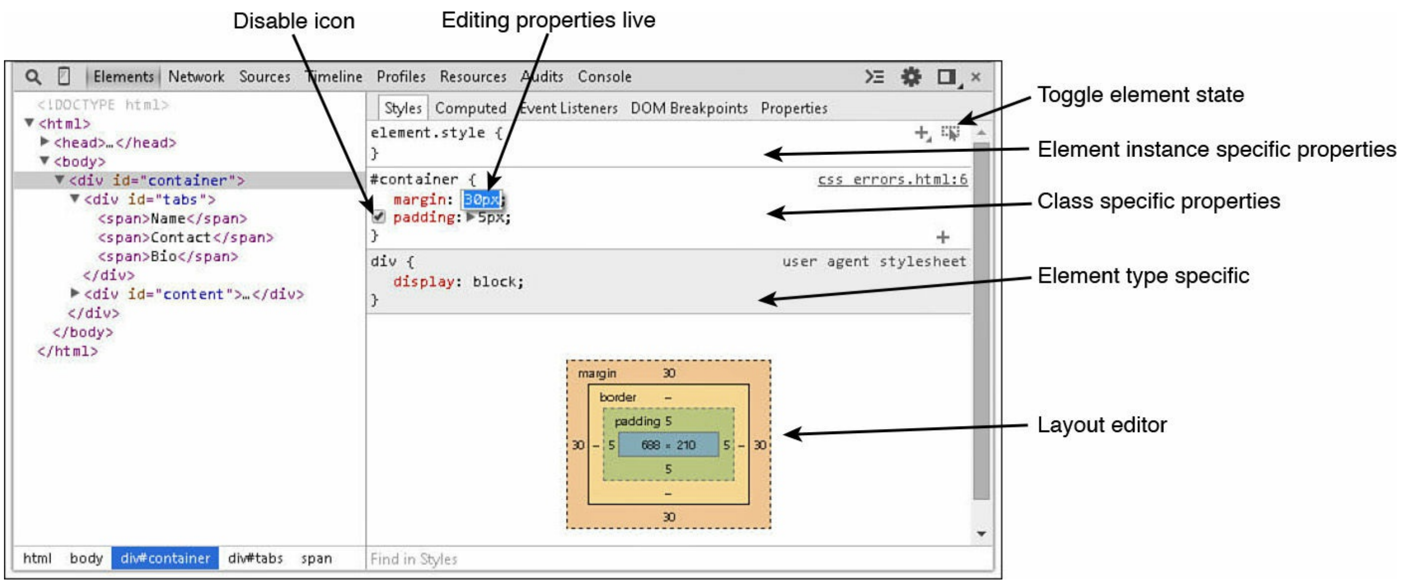
Debugging CSS

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

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

Using the CSS Style Editor

The CSS Style editor, shown in Figure 2.13, provides access to all the CSS properties of elements loaded in the web page. You can access the CSS Style editor by selecting the Elements tab in the developer tools and selecting the Styles sub tab shown in Figure 2.13.



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

```
element.style {
}
#container {                      css errors.html:6
   margin: 30px;
   padding: ▶5px;
}
div {                             user agent stylesheet
   display: block;
}
```

FIGURE 2.13 Editing CSS properties inside the CSS Style editor.

Note

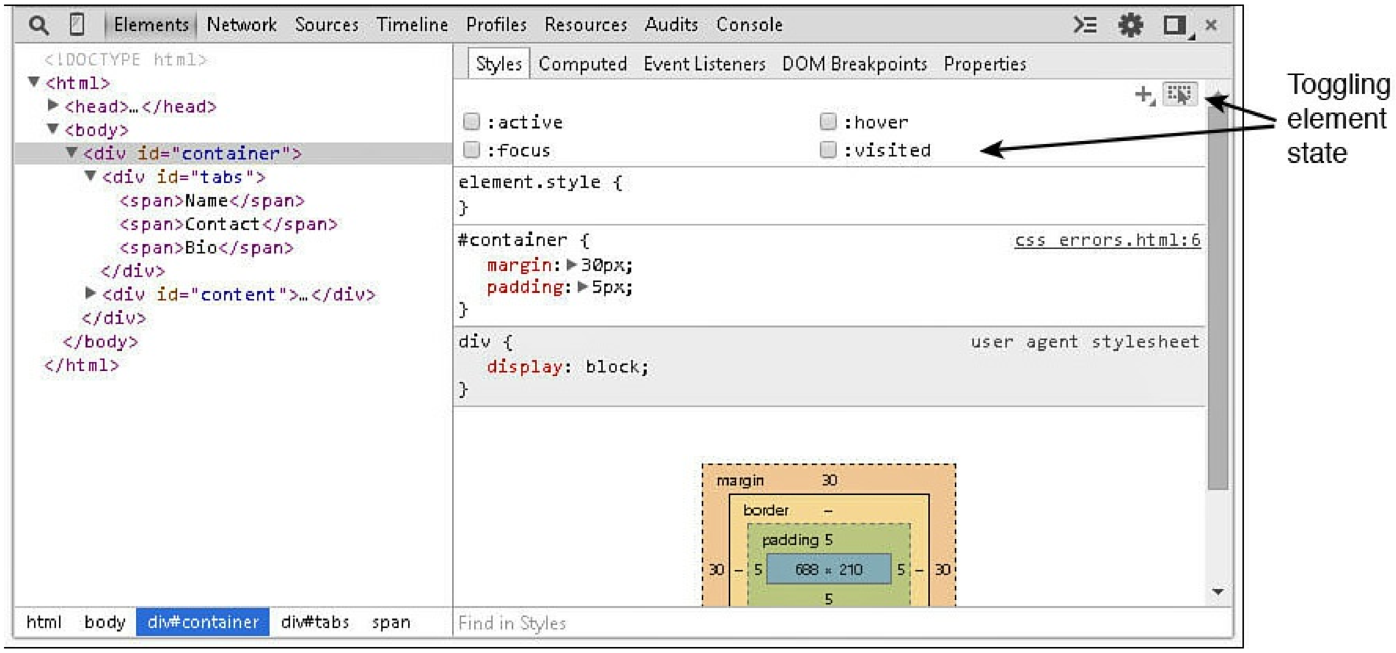
You can view the full CSS files by selecting the Sources tab in the developer tools and then selecting the HTML or CSS files you want to view.

From the CSS Style editor, you also have the capability to do the following, as shown in Figure 2.13:

- **Disable a CSS Style**—The disable icon allows you to enable/disable a specific CSS property.
- **Edit Element Specific CSS Properties**—You can view and edit the CSS

properties specific to this instance of the element.

- **Edit Class Specific CSS Properties**—You can view and edit the CSS properties specific to the element's class(es).
- **Edit Element Type Specific CSS Properties**—You can view and edit the CSS properties specific to all elements of this type.
- **Toggle Element State**—The toggle element state button loads the menu shown in Figure 2.14 that allows you to set the element state to `active`, `hover`, `focus`, or `visited` so that you can see the CSS styles that apply for each of those states.



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

```
element.style {
}
#container {                                   css_errors.html:6
   margin: ▶30px;
   padding: ▶5px;
}
div {                                       user agent stylesheet
   display: block;
}
```

FIGURE 2.14 Setting the state of the element to view CSS properties specific to active, hover, focus, or visited states.

- **View and Edit Layout**—The Layout editor, shown in Figure 2.15, allows you to view and change the margin, border padding, height, and width for the element.

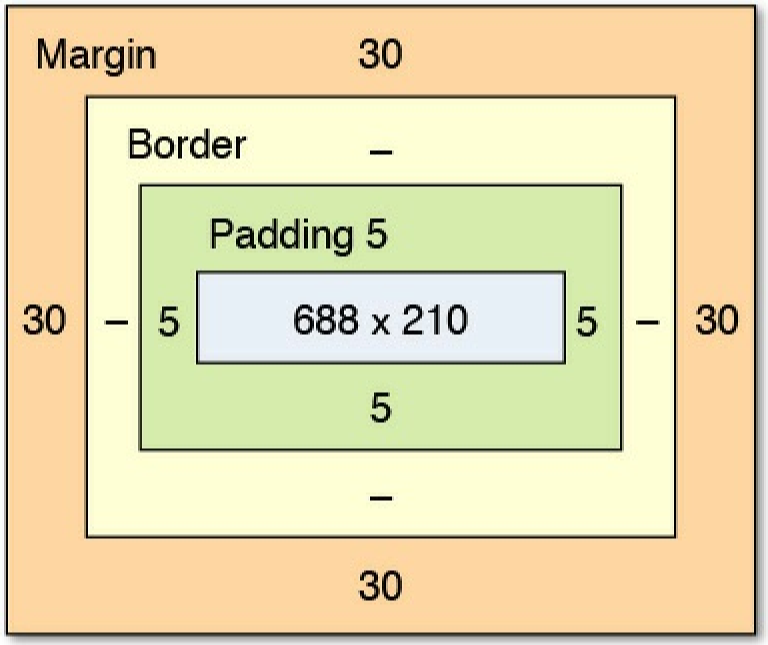


FIGURE 2.15 Viewing the CSS layout properties inside the Layout editor inside the Element inspector.

Using the Layout Editor

Another extremely powerful tool when debugging CSS is the Layout editor in the Element editor. The Layout editor, shown in Figure 2.15, provides an easy-to-use visual interface to the CSS layout of the selected HTML element. You can see the actual values for things like margin and padding. You can also double-click those values to edit the CSS properties directly.

From the Layout editor, you can view and modify the following properties:

- **Margin**—The margin is the outermost box shown in the Layout editor. There is a value on each of the four sides of the margin. You can double-click those values and change the CSS property directly in the Layout editor.
- **Border**—The border is the next box. It also has four values you can change to adjust the CSS border properties of the HTML element.
- **Padding**—The padding is the next box. It also has four values you can change to adjust the CSS padding properties of the HTML element.
- **Content**—The content is the innermost box in the Layout editor. It has two values, the length and width, that you change to set the CSS length and width CSS properties of the HTML element.

Try it Yourself: Editing the CSS Layout

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

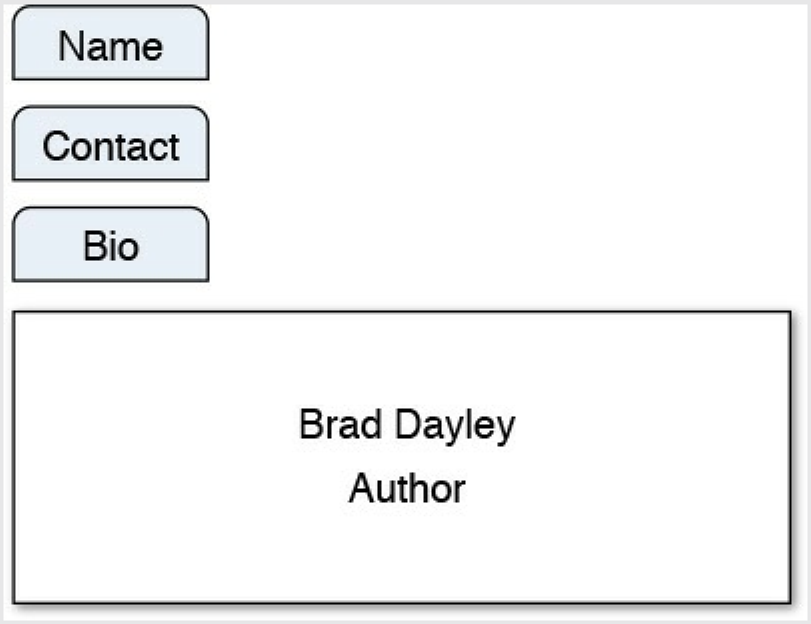


FIGURE 2.16 The poor CSS layout of the tabs makes the web page ugly.

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

Click here to view code image

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

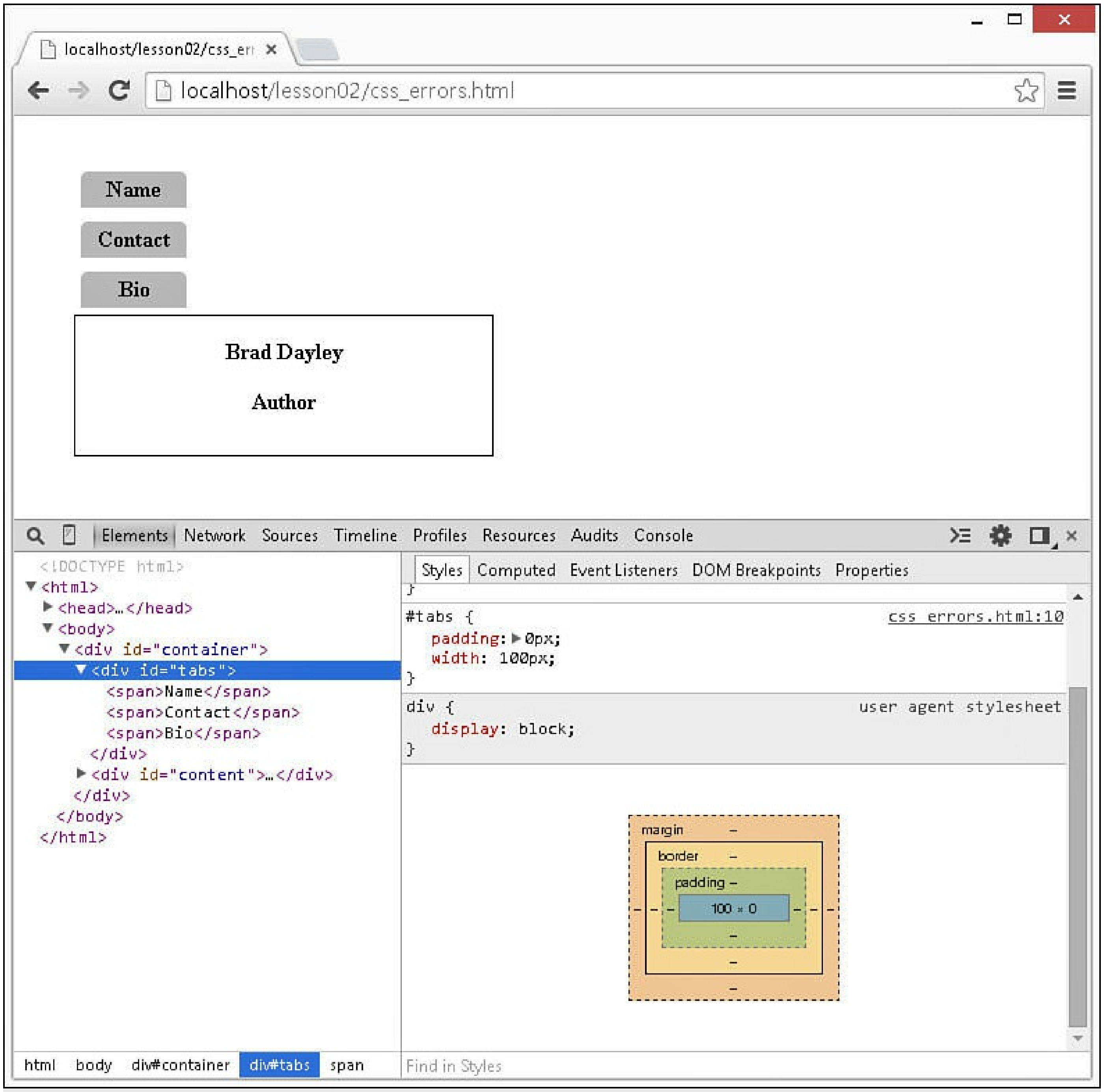
Use the following steps to correct the CSS layout:

1. Add the code in Listing 2.3 to a new file css_errors.html in the lesson02 folder of your project and save the document.
2. Open Chrome.
3. Open the following URL in Chrome and load the developer tools:

Click here to view code image

```
http://localhost/lesson02/css_errors.html
```

4. Click the Elements tab in the developer tools and expand the `<html>`, then `<body>`, `<div id="container">`, and then `<div id="tabs">` elements, as shown in Figure 2.17.


localhost/lesson02/css_err
localhost/lesson02/css_errors.html
Name
Contact
Bio
Brad Dayley
Author
Elements Network Sources Timeline Profiles Resources Audits Console
Styles Computed Event Listeners DOM Breakpoints Properties
margin –
border –
padding –
100 × 0
html body div#container div#tabs span
Find in Styles

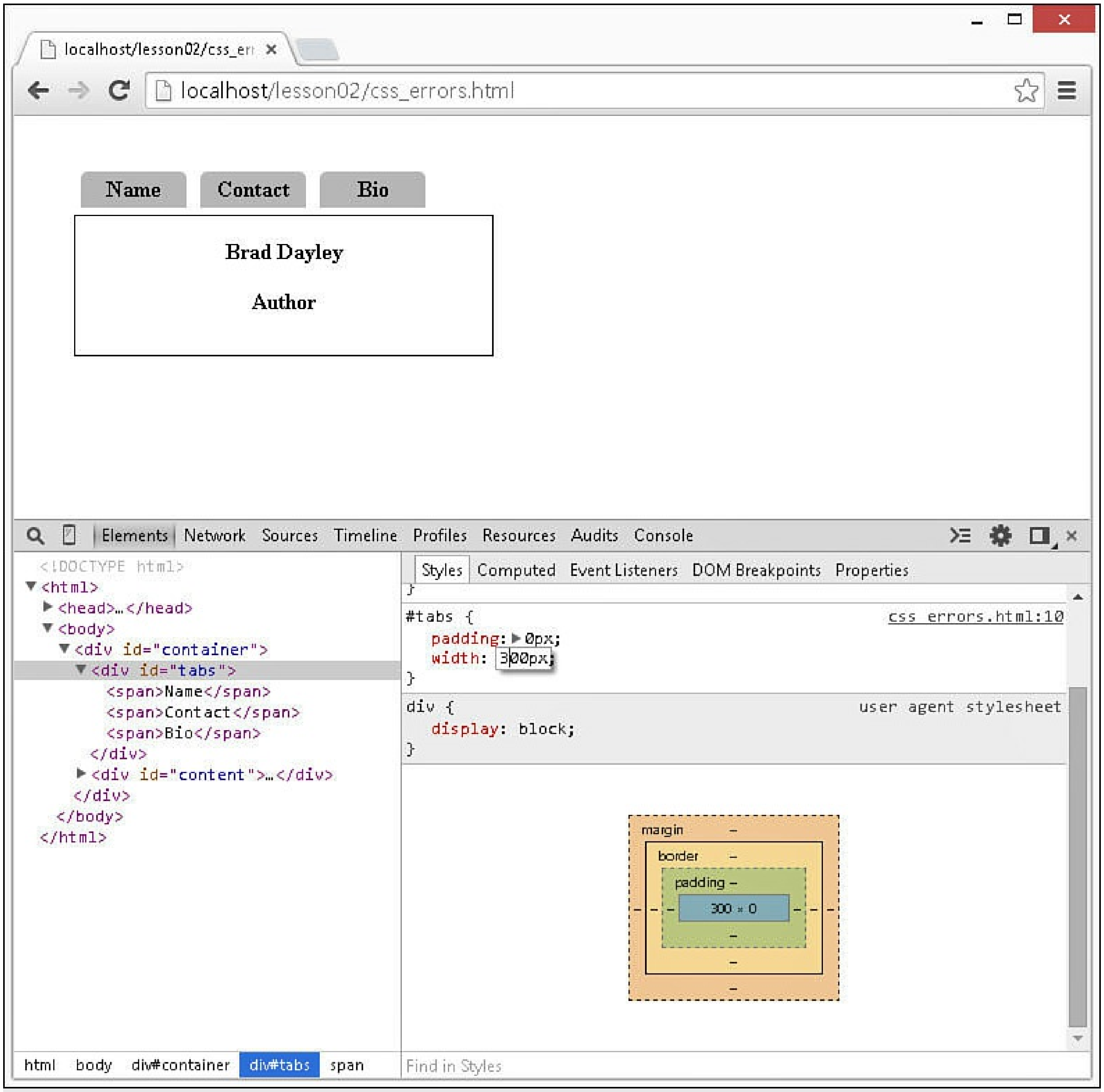
```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<div id="container">
      ▼<div id="tabs">
          <span>Name</span>
          <span>Contact</span>
          <span>Bio</span>
        </div>
      ▶<div id="content">…</div>
      </div>
    </body>
</html>
```

```
#tabs {                                  css_errors.html:10
   padding: ▶0px;
   width: 100px;
}
div {                                    user agent stylesheet
   display: block;
}
```

FIGURE 2.17 The Layout shows that the width of the tabs <div> container is only 100px, which is not enough room for three tabs side by side.

5. Look at the size of the <div id="tabs"> element and see that it is only 100px. The <span> elements are 70px wide, so the <div> element could not possibly support all three <span> elements side by side.

6. To fix this problem, click the Styles tab and change the width property to `300px`, as shown in Figure 2.18. Notice that the tabs now are all side by side, but there is still too much space between them.



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

```
#tabs {                                        css_errors.html:10
   padding: ▶ 0px;
   width: 300px;
}
div {                                          user agent stylesheet
   display: block;
}
```

FIGURE 2.18 Changing the `<div>` width allows the tabs to be side by side, but they are still too far apart.

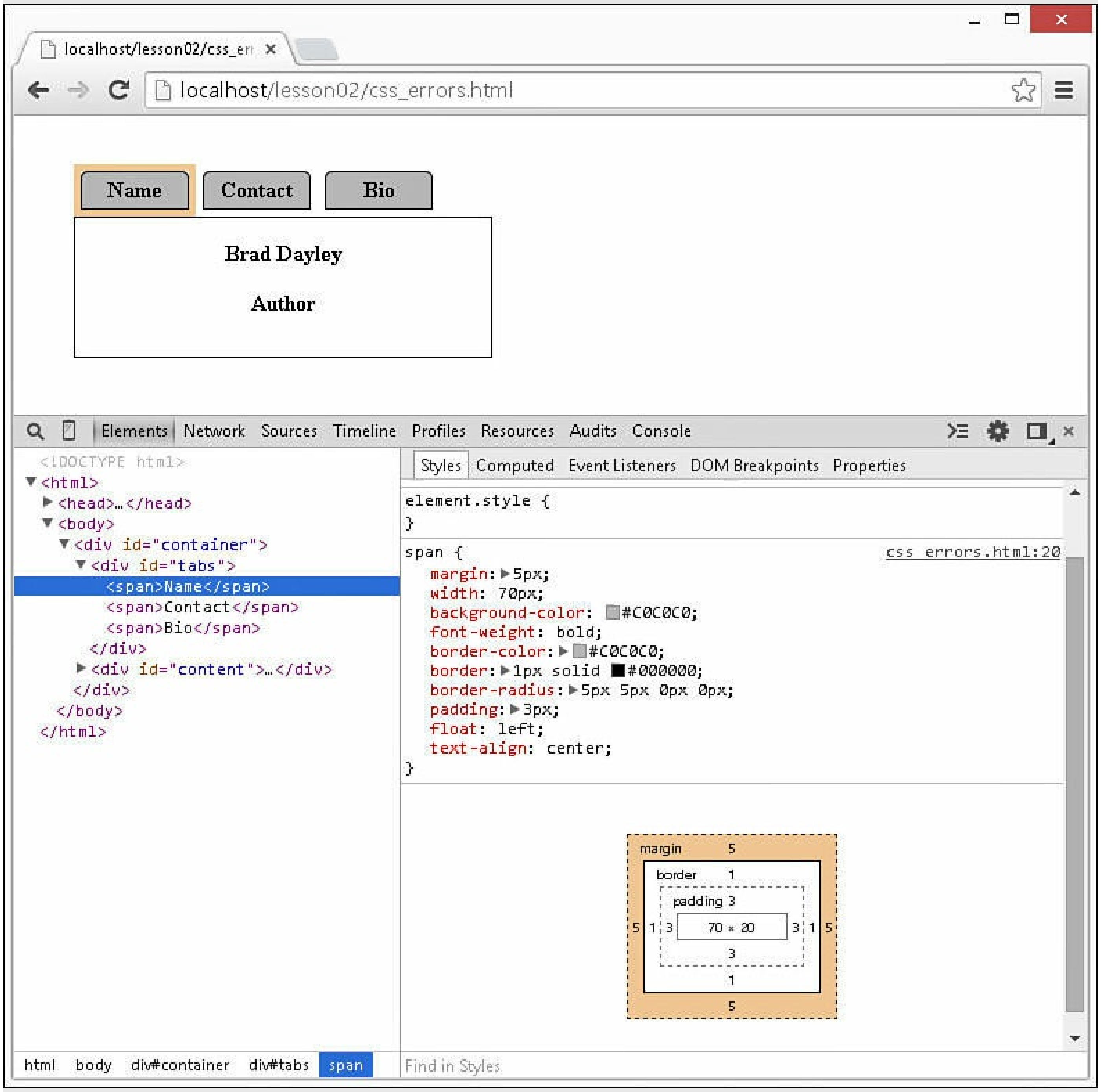
Note

You can also modify the margin, border, height, width, and padding values directly in the Layout view.

---

7. Right-click the Name tab in the web page and select Inspect Element from the list. That element is automatically selected in the Element inspector.

8. Click the Layout tab and hover over the margin box, as shown in Figure 2.19. Notice that there is a margin of 5px around the `<span>` element. That is why they are not close to each other.

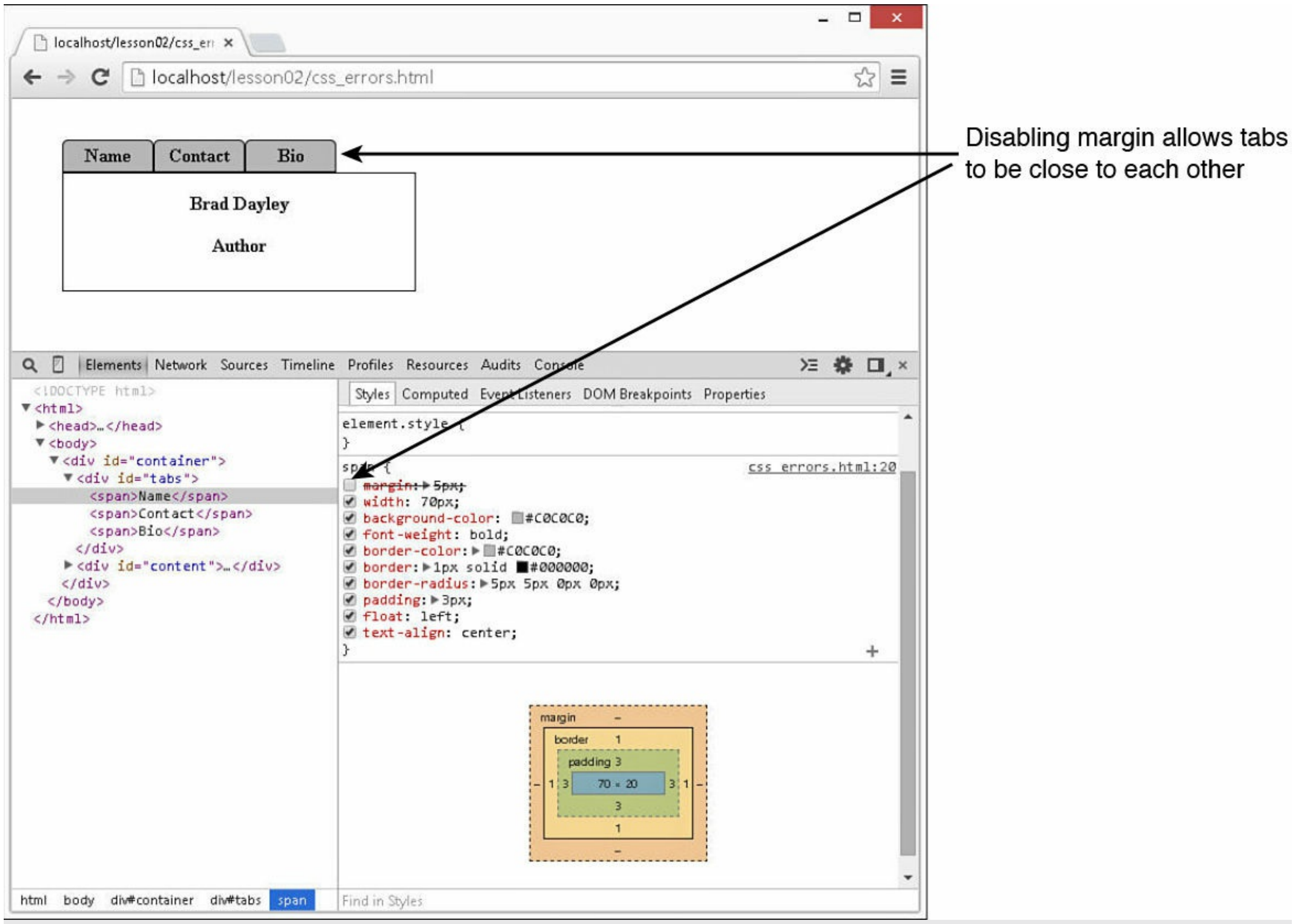


```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
    ▼<div id="container">
      ▼<div id="tabs">
          <span>Name</span>
          <span>Contact</span>
          <span>Bio</span>
        </div>
      ▶<div id="content">…</div>
      </div>
    </body>
  </html>
```

```
element.style {
}
span {                                                css_errors.html:20
   margin:▶5px;
   width: 70px;
   background-color: #C0C0C0;
   font-weight: bold;
   border-color:▶#C0C0C0;
   border:▶1px solid #000000;
   border-radius:▶5px 5px 0px 0px;
   padding:▶3px;
   float: left;
   text-align: center;
}
```

FIGURE 2.19 The Layout reveals that there is a margin around the `<span>` element keeping the tabs apart.

9. Go to the Styles tab and disable the margin property for the `<span>` element, as shown in Figure 2.20. The tabs are now right together and sitting directly on top of the display box.



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

```
element.style {
}
span {                                  css_errors.html:20
  ~~margin: ▶ 5px;~~
  width: 70px;
  background-color: #C0C0C0;
  font-weight: bold;
  border-color: ▶ #C0C0C0;
  border: ▶ 1px solid #000000;
  border-radius: ▶ 5px 5px 0px 0px;
  padding: ▶ 3px;
  float: left;
  text-align: center;
}
```

FIGURE 2.20 Disabling the margin allows the tabs to sit close to each other and the display box.

Debugging JavaScript

You already have learned to look for exception errors in JavaScript and other scripts in the JavaScript console. What if your script isn't causing any browser errors, but it just isn't working the way you want it to? All of the browsers have a very nice integrated debugger to help you out. The following sections cover using the Chrome developer tools to debug JavaScript.

Navigating the JavaScript Debugger

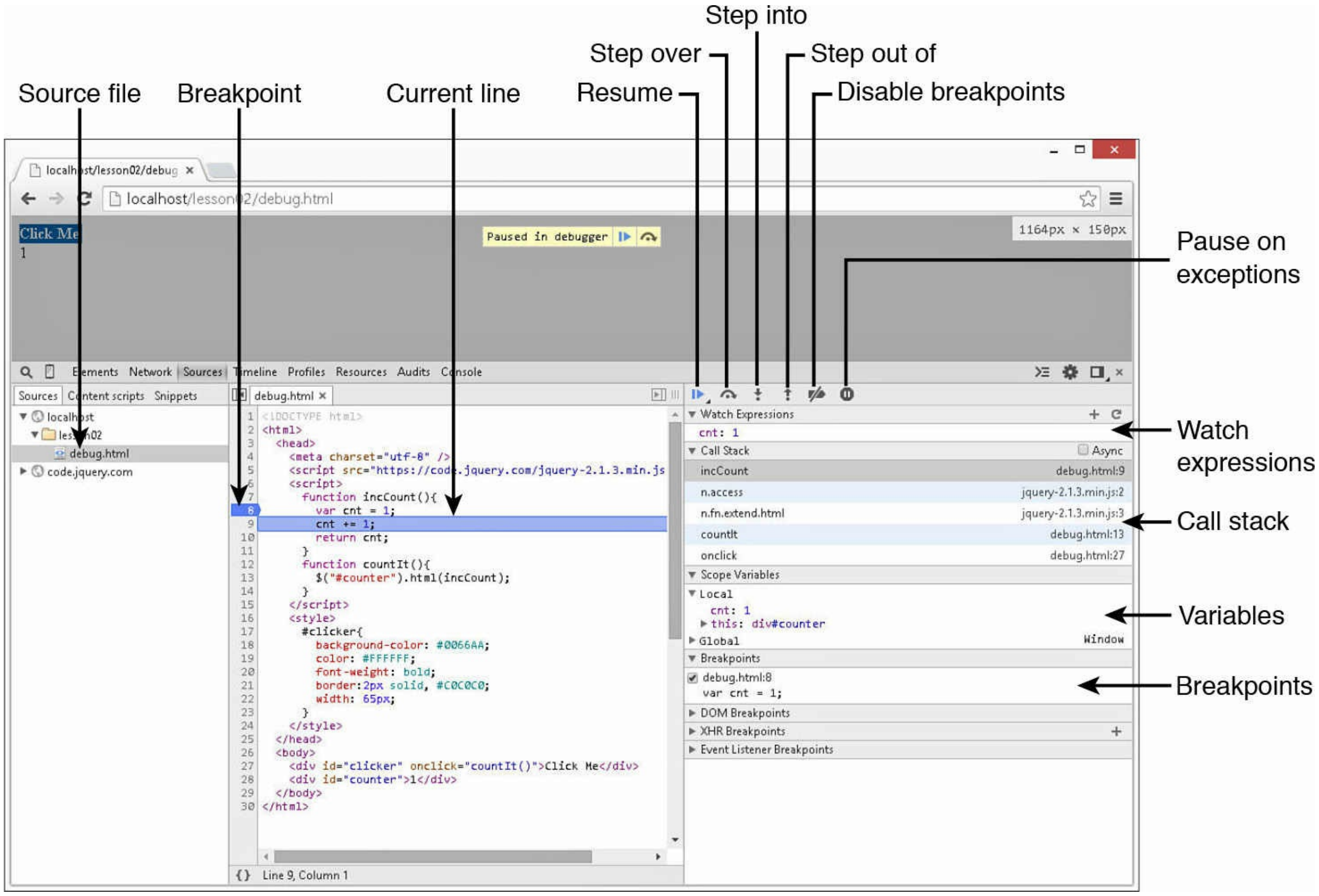
The JavaScript debugger allows you to view the JavaScript scripts that are loaded into the browser with the web page. In addition to viewing the scripts, you can set breakpoints, watch variable values, and view the call stack, just as you would with any other debugger.

Figure 2.21 shows the components of the JavaScript debugger available in the developer tools. From the JavaScript debugger, you have access to the following features:

- **JavaScript View**—This shows you the actual JavaScript code.
- **Source Selection Menu**—This menu shows a list of the source files including the JavaScript scripts loaded with the web page. You can click this menu to select which JavaScript file to load in the view.
- **Pause on Exceptions**—When this option is selected, the browser will break into the debugger and stop executing if a JavaScript exception occurs.
- **Watch**—The Watch pane shown in Figure 2.21 gives you a list of functions, variables, properties, and so on that are available at the current execution of the code. This is an extremely valuable window. From here you can see what the values of variables and objects are as the code is executing. In addition, you can add your own expressions to the Watch pane by clicking the plus icon at the top of the watch list. A great feature of the Watch pane is that you can double-click variable values and change the value that is used in execution. This is a great way to test what-if scenarios.
- **Call Stack**—The Call Stack pane provides a history of the function calls that led up to the currently executing line of code. One of the most valuable aspects of the Call Stack pane is that you can see the parameter values passed into each function by expanding the function name. You can also click the function name, and that file will be loaded in the JavaScript view and that line of code highlighted.
- **Breakpoints**—Breakpoints allow you to specify where to stop when executing JavaScript. When you set a breakpoint, the browser stops executing and breaks into the debugger before it executes that line of code. You set breakpoints by clicking to the left of the line of code in the JavaScript view. They are denoted by a red dot. To remove the breakpoint, click it. The Breakpoints pane shows you a list of breakpoints that have been set. You can disable the breakpoint by unchecking the box next to it. You can disable breakpoints by clicking on the Disable Breakpoints button.
- **Currently Executing Line**—The currently executing line of code is denoted by a yellow arrow.
- **Resume**—This allows the script to continue executing normally until hitting another breakpoint if one is encountered.
- **Step Into**—When you click this icon, code advances one line. If the line of code is executing another function, you are taken to the first line of code in that function.
- **Step Over**—When you click this icon, code advances one line. If the line of code is executing another function, that function is executed and you are taken to the next line of code in the current function. If a breakpoint is encountered when stepping

over a function, the browser will stop executing at that location in the script.

- **Step Out**—When you click this icon, the current function finishes executing and you are taken to the next line of code in the calling function.



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <script src="https://code.jquery.com/jquery-2.1.3.min.js
    <script>
      function incCount(){
        var cnt = 1;
        cnt += 1;
        return cnt;
      }
      function countIt(){
        $("#counter").html(incCount);
      }
    </script>
    <style>
      #clicker{
        background-color: #0066AA;
        color: #FFFFFF;
        font-weight: bold;
        border:2px solid, #C0C0C0;
        width: 65px;
      }
    </style>
  </head>
  <body>
    <div id="clicker" onclick="countIt()">Click Me</div>
    <div id="counter">1</div>
  </body>
</html>
```

```
▼ Watch Expressions
  cnt: 1
▼ Call Stack                                  Async
  incCount                            debug.html:9
  n.access                     jquery-2.1.3.min.js:2
  n.fn.extend.html             jquery-2.1.3.min.js:3
  countIt                            debug.html:13
  onclick                            debug.html:27
▼ Scope Variables
▼ Local
    cnt: 1
  ▶ this: div#counter
▶ Global                                    Window
▼ Breakpoints
☑ debug.html:8
  var cnt = 1;
▶ DOM Breakpoints
▶ XHR Breakpoints
▶ Event Listener Breakpoints
```

FIGURE 2.21 The developer tools JavaScript debugger provides code, watch, stack, and breakpoint views.

Try it Yourself: Using the JavaScript Debugger

The following example will help you become more familiar with the JavaScript debugger. Consider the code in Listing 2.4. This is a basic web page that contains a button and a count string. The HTML contains two `<div>` elements. The first, `<div id="clicker" onclick="countIt()">`, is used for a simple button. When you click the button, the JavaScript function `countIt()` is called. The second, `<div id="counter">`, is used to display a number.

The JavaScript is supposed to increase the number by 1 each time the button is clicked. A problem exists with the JavaScript code, though; the number will not increase past 2.

LISTING 2.4 debug.html A Very Simple HTML Document with JavaScript Errors Illustrated

Click here to view code image

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <meta charset="utf-8" />
05     <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
06     <script>
07       function incCount(){
08         var cnt = 1;
09         cnt += 1;
10         return cnt;
11       }
12       function countIt(){
13         $("#counter").html(incCount);
14       }
15     </script>
16     <style>
17       #clicker{
18         background-color: #0066AA;
19         color: #FFFFFF;
20         font-weight: bold;
21         border:2px solid, #C0C0C0;
22         width: 65px;
23       }
24     </style>
25   </head>
26   <body>
27     <div id="clicker" onclick="countIt()">Click Me</div>
28     <div id="counter">1</div>
29   </body>
30 </html>
```

Walk through the following steps to set a breakpoint in the JavaScript Debugger and debug the problem:

1. Add the code in Listing 2.4 to a new file debug.html in the lesson02 folder of your project and save the document.

2. Open Chrome.

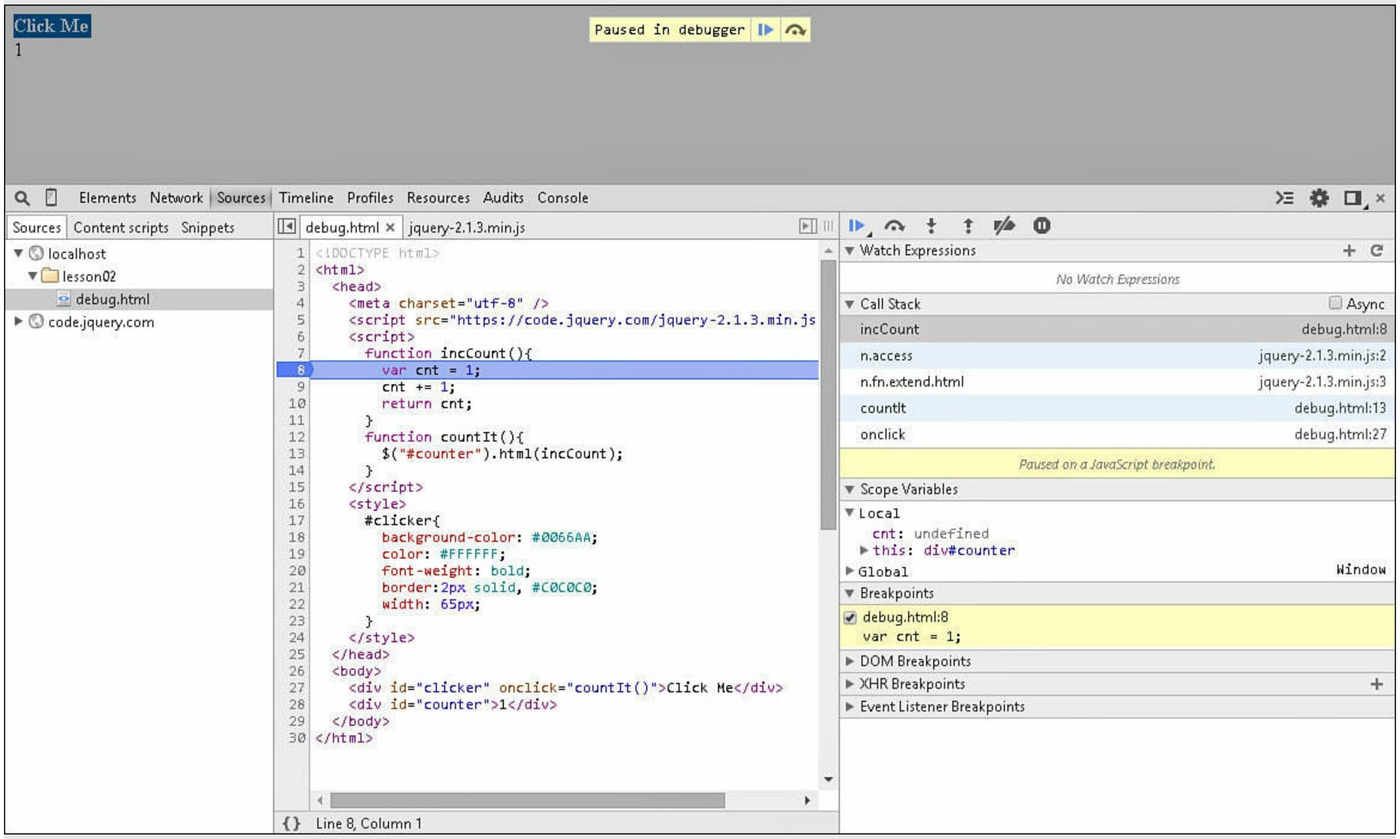
3. Open the following URL in Chrome and load the developer tools. Notice the single button and the count value of 1:

Click here to view code image

```
http://localhost/lesson02/debug.html
```

4. Click the Sources tab in the developer tools and then select debug.html from the script selection menu. You should see the code from Listing 2.4 in the Sources area of the debugger. Notice that the function that sets the value that is placed in the counter div is in lines 7–11.

5. Set a breakpoint on line 8 by clicking to the left of the line number. A blue arrow should appear, as shown in Figure 2.22.



```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <script src="https://code.jquery.com/jquery-2.1.3.min.js
6      <script>
7        function incCount(){
8          var cnt = 1;
9          cnt += 1;
10         return cnt;
11       }
12       function countIt(){
13         $("#counter").html(incCount);
14       }
15     </script>
16     <style>
17       #clicker{
18         background-color: #0066AA;
19         color: #FFFFFF;
20         font-weight: bold;
21         border:2px solid, #C0C0C0;
22         width: 65px;
23       }
24     </style>
25   </head>
26   <body>
27     <div id="clicker" onclick="countIt()">Click Me</div>
28     <div id="counter">1</div>
29   </body>
30 </html>
```

Watch Expressions	
No Watch Expressions	

Call Stack	Async
incCount	debug.html:8
n.access	jquery-2.1.3.min.js:2
n.fn.extend.html	jquery-2.1.3.min.js:3
countIt	debug.html:13
onclick	debug.html:27

Paused on a JavaScript breakpoint.

```
Scope Variables
▼ Local
    cnt: undefined
  ▶ this: div#counter
▶ Global                                   Window
Breakpoints
☑ debug.html:8
  var cnt = 1;
DOM Breakpoints
XHR Breakpoints
Event Listener Breakpoints
```

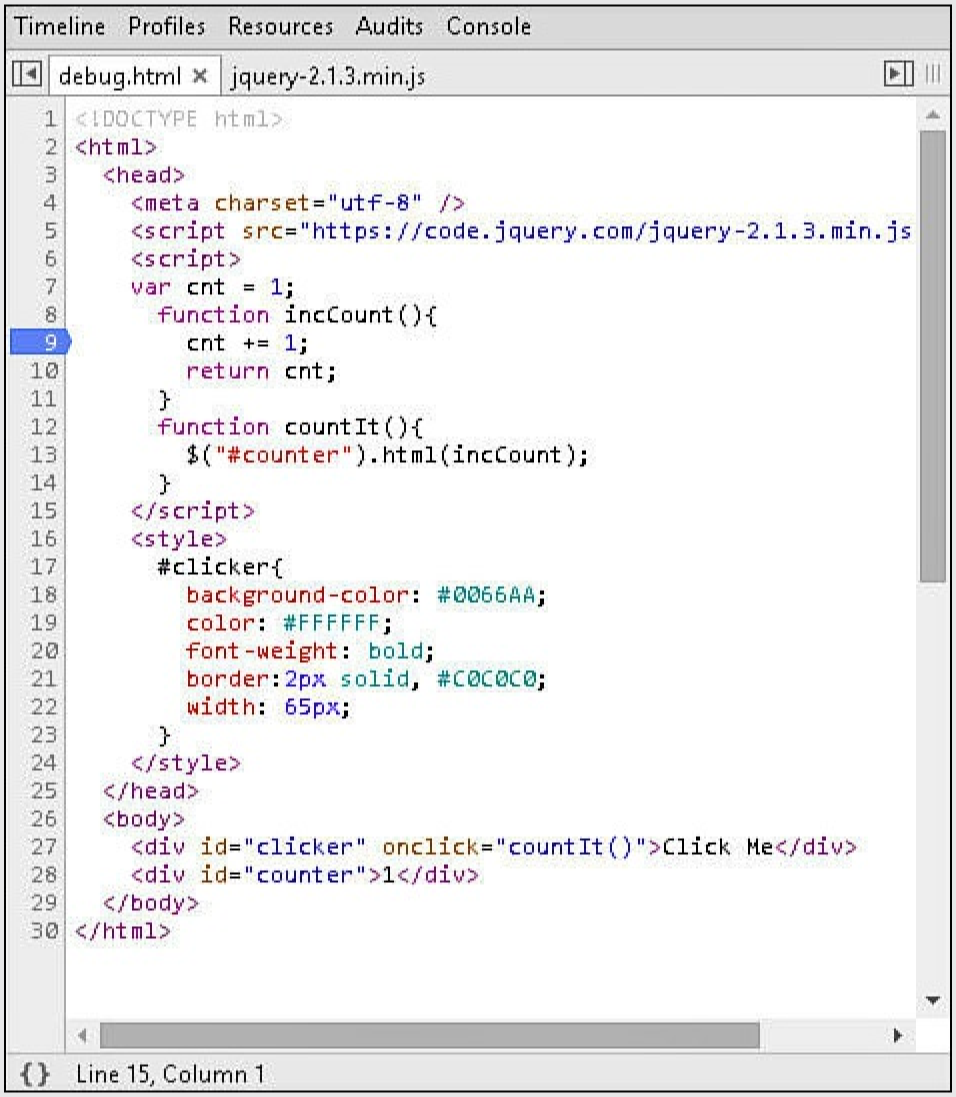
FIGURE 2.22 The JavaScript debugger in the developer tools is stopped on line 8 because of a breakpoint. The Scope Variables shows the value of the `cnt` variable as undefined.

6. Now click the button on the web page. You should see line 8 in the debugger highlighted. The script has stopped executing on that line. This function will determine what value will be placed in the counter. Notice that the value of the `cnt` variable in the Scope Variables tab is undefined.
7. Click the Step Over icon. You should see the value of `cnt` go to 1.
8. Click the Step Over icon again. Now the value of `cnt` is 2, as expected, changed by the `cnt += 1;` line.
9. Click the Resume button to allow the script to finish execution.
10. Notice that the value on the web page has gone to 2. So far, so good.
11. Click the button again in the web page. The debugger should activate again and be stopped in the same location as step 6. Notice that the value of `cnt` is undefined again.
12. Click the Step Over icon; `cnt` changes to 1. Click Step Over again and `cnt` changes to 2. As the button is clicked, `cnt` is reset to undefined, set to 1, and then incremented to 2.
13. To fix the problem, switch lines 7 and 8 in the original file in Eclipse so that

the definition of `cnt` happens before the definition of `incCount()`. This defines the variable `cnt` and sets the value only once when the script is loaded before the function is defined. Save the file.

14. Clear the breakpoint on line 8 by clicking on it in Chrome, and reload the web page.

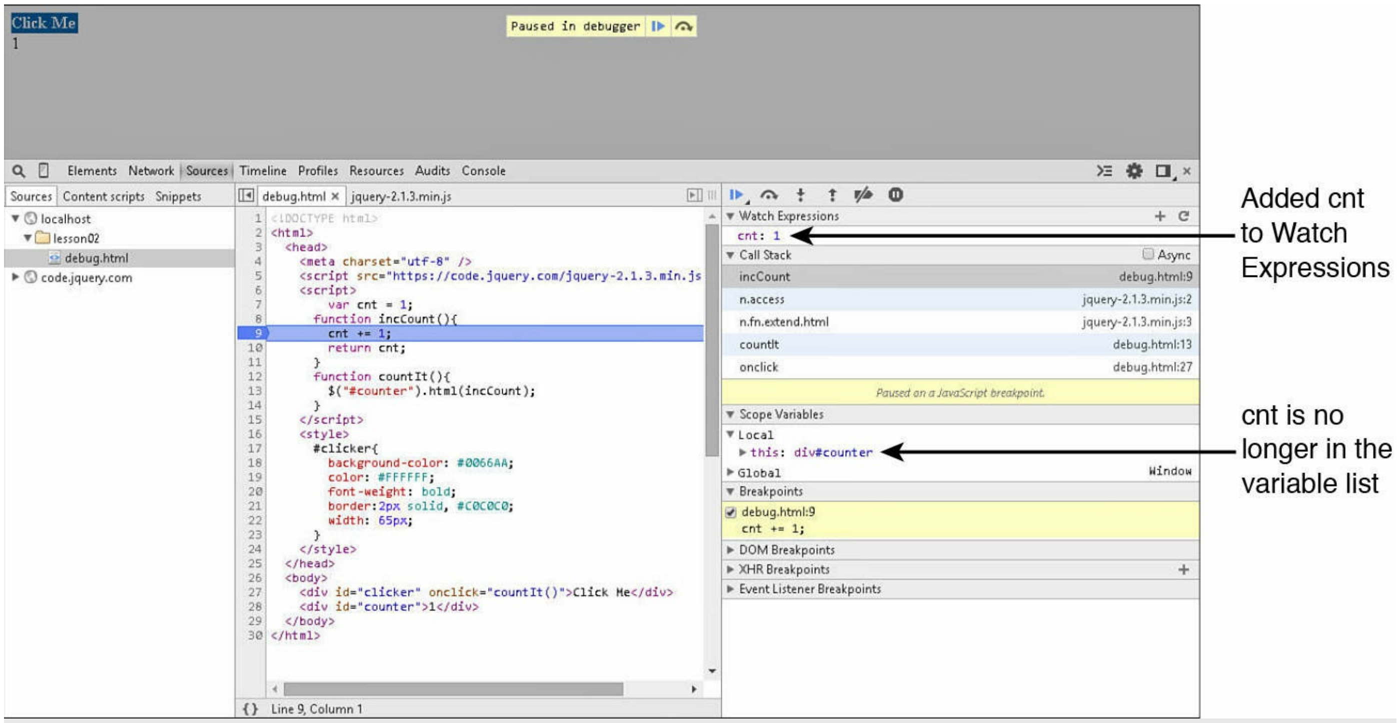
15. Add a breakpoint to line 9, as shown in Figure 2.23.



```
 1 <!DOCTYPE html>
 2 <html>
 3   <head>
 4     <meta charset="utf-8" />
 5     <script src="https://code.jquery.com/jquery-2.1.3.min.js
 6     <script>
 7     var cnt = 1;
 8       function incCount(){
 9         cnt += 1;
10         return cnt;
11       }
12       function countIt(){
13         $("#counter").html(incCount);
14       }
15     </script>
16     <style>
17       #clicker{
18         background-color: #0066AA;
19         color: #FFFFFF;
20         font-weight: bold;
21         border:2px solid, #C0C0C0;
22         width: 65px;
23       }
24     </style>
25   </head>
26   <body>
27     <div id="clicker" onclick="countIt()">Click Me</div>
28     <div id="counter">1</div>
29   </body>
30 </html>
```

FIGURE 2.23 Changing the breakpoint from the function definition to the first line in the function.

16. Click the button in the web page, and the JavaScript should break again—this time on line 9. This time you should not see the `cnt` variable in the Scope Variables pane, as shown in Figure 2.24.


Click Me
1
Paused in debugger
Elements Network Sources Timeline Profiles Resources Audits Console
Sources Content scripts Snippets
localhost
lesson02
debug.html
code.jquery.com
debug.html ×
jquery-2.1.3.min.js
Line 9, Column 1
Added cnt to Watch Expressions
cnt is no longer in the variable list

```
 1 <!DOCTYPE html>
 2 <html>
 3   <head>
 4     <meta charset="utf-8" />
 5     <script src="https://code.jquery.com/jquery-2.1.3.min.js
 6     <script>
 7         var cnt = 1;
 8       function incCount(){
 9         cnt += 1;
10         return cnt;
11       }
12       function countIt(){
13         $("#counter").html(incCount);
14       }
15     </script>
16     <style>
17       #clicker{
18         background-color: #0066AA;
19         color: #FFFFFF;
20         font-weight: bold;
21         border:2px solid, #C0C0C0;
22         width: 65px;
23       }
24     </style>
25   </head>
26   <body>
27     <div id="clicker" onclick="countIt()">Click Me</div>
28     <div id="counter">1</div>
29   </body>
30 </html>
```

```
▼ Watch Expressions
  cnt: 1
▼ Call Stack                                   Async
  incCount                             debug.html:9
  n.access                     jquery-2.1.3.min.js:2
  n.fn.extend.html             jquery-2.1.3.min.js:3
  countIt                             debug.html:13
  onclick                             debug.html:27
            Paused on a JavaScript breakpoint.
▼ Scope Variables
▼ Local
  ▶ this: div#counter
▶ Global                                     Window
▼ Breakpoints
  ☑ debug.html:9
    cnt += 1;
▶ DOM Breakpoints
▶ XHR Breakpoints
▶ Event Listener Breakpoints
```

FIGURE 2.24 Adding a new Watch expression for `cnt` so you don't have to expand the Window element each time.

17. Rather than having to expand the Window element each time you want to debug, click New Watch Expression at the top of the Watch list shown in Figure 2.24, type in `cnt`, and press Enter. This adds a new watch expression right at the top for the `cnt` variable, as shown in Figure 2.24.

18. Click the Step Over icon and `cnt` will go to 2. Then click the Continue button to resume execution.

19. Click the button in the web page again, and this time the `cnt` variable will show as 2. When you click the Step Over button, the `cnt` variable will go to 3.

20. The program appears to be working, so click the breakpoint on line 9 to remove the breakpoint, and then click the Continue button to resume execution.

21. Now every time you click the button, the number is incremented. You've just debugged the JavaScript.

This was a very basic example, but it was made simple so that it would be easy to follow the steps and get used to how the debugger works. You will likely come back to the debugger several times when doing exercises in the book. Keep in mind the basic steps. Set a breakpoint and watch the variables as you step through the code.

So How Do You Debug jQuery or AngularJS?

A question that comes up frequently, even with people who are experienced with debugging JavaScript, is how to debug jQuery or AngularJS. The answer is simple. AngularJS, jQuery, and the numerous JavaScript plug-ins and versions are just additional JavaScripts. To debug AngularJS or jQuery, use a nonminified version of the library in your project. You learn how to do that later in this book.

The reason you download a nonminified version is that the minified is unreadable. Everything is crunched together in one line and doesn't show up well in the debugger. The nonminified version is formatted in a readable form.

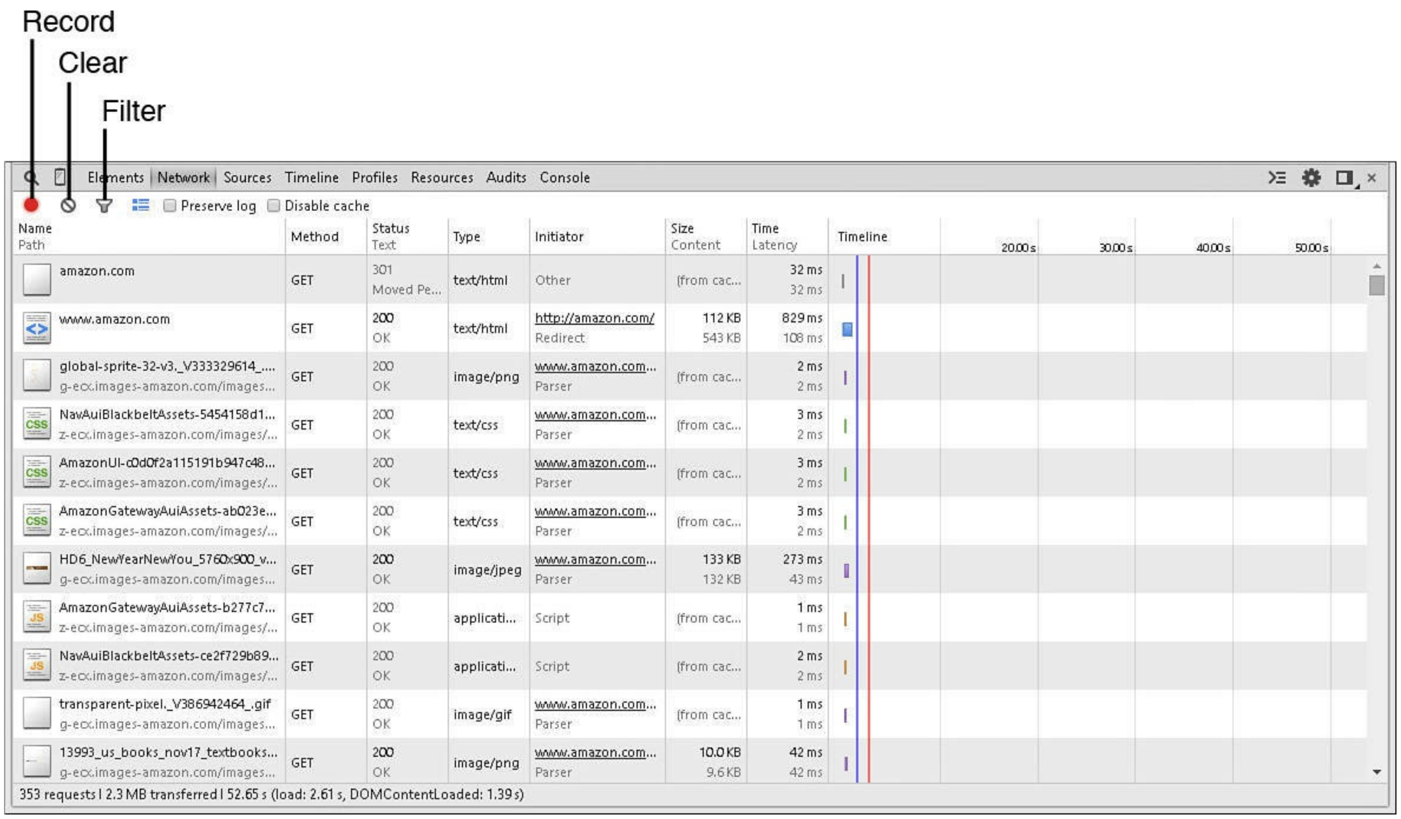
Note

Even if you cannot get a nonminified version of a JavaScript file, you can always open the file in Eclipse and select File, Format from the main menu. Eclipse will automatically format the file to a readable form. Most IDEs will have that type of feature.

With the AngularJS, jQuery, or any other JavaScript library formatted, you can debug it like any other JavaScript file.

Analyzing the Network Traffic

A very valuable tool available in the developer tools that is often used in debugging JavaScript is the network traffic analyzer. The network traffic analyzer, shown in Figure 2.25, is available by clicking the Net tab in the developer tools. The traffic analyzer displays information about each request from the browser to the web server. This allows you to get a better understanding about what data is being transferred and whether requests are happening at all and in the right order.



Name / Path	Method	Status / Text	Type	Initiator	Size / Content	Time / Latency	Timeline
amazon.com	GET	301 Moved Pe...	text/html	Other	(from cac...	32 ms / 32 ms	
www.amazon.com	GET	200 OK	text/html	http://amazon.com/ Redirect	112 KB / 543 KB	829 ms / 108 ms	
global-sprite-32-v3._V333329614_.... g-ecx.images-amazon.com/images...	GET	200 OK	image/png	www.amazon.com... Parser	(from cac...	2 ms / 2 ms	
NavAuiBlackbeltAssets-5454158d1... z-ecx.images-amazon.com/images/...	GET	200 OK	text/css	www.amazon.com... Parser	(from cac...	3 ms / 2 ms	
AmazonUI-c0d0f2a115191b947c48... z-ecx.images-amazon.com/images/...	GET	200 OK	text/css	www.amazon.com... Parser	(from cac...	3 ms / 2 ms	
AmazonGatewayAuiAssets-ab023e... z-ecx.images-amazon.com/images/...	GET	200 OK	text/css	www.amazon.com... Parser	(from cac...	3 ms / 2 ms	
HD6_NewYearNewYou_5760x900_v... g-ecx.images-amazon.com/images...	GET	200 OK	image/jpeg	www.amazon.com... Parser	133 KB / 132 KB	273 ms / 43 ms	
AmazonGatewayAuiAssets-b277c7... z-ecx.images-amazon.com/images/...	GET	200 OK	applicati...	Script	(from cac...	1 ms / 1 ms	
NavAuiBlackbeltAssets-ce2f729b89... z-ecx.images-amazon.com/images/...	GET	200 OK	applicati...	Script	(from cac...	2 ms / 2 ms	
transparent-pixel._V386942464_.gif g-ecx.images-amazon.com/images...	GET	200 OK	image/gif	www.amazon.com... Parser	(from cac...	1 ms / 1 ms	
13993_us_books_nov17_textbooks... g-ecx.images-amazon.com/images...	GET	200 OK	image/png	www.amazon.com... Parser	10.0 KB / 9.6 KB	42 ms / 42 ms	

353 requests | 2.3 MB transferred | 52.65 s (load: 2.61 s, DOMContentLoaded: 1.39 s)

FIGURE 2.25 Network traffic required to load amazon.com.

Figure 2.25 shows the traffic involved in loading the amazon.com web page. There are numerous requests, each one represented by a single line in the traffic list. For each request, the following is shown in the traffic:

- **URL Path**—The URL of the request can be very useful. You can right-click the URL and copy it, or even open it in another tab or window. This allows you to debug a single request and not the full web page load.
- **Method**—The type of HTTP method that was used in the request.
- **Status**—You can use the status to determine whether the request was successful and whether it is still running. For example, the web page may not look right because an image request failed to load, which is very easy to diagnose from the Net tab in the developer tools.
- **Type**—Type of data that was retrieved by the request.
- **Initiator**—Where the request was initiated from.
- **Size**—The size may also be useful in that it allows you to quickly find requests that require a lot of disk space and network bandwidth.
- **Latency**—Shows the latency times for the request in milliseconds.
- **Timeline**—Shows the time in milliseconds the request took. This is very useful in diagnosing slow-responding web pages and other problems related to speed.

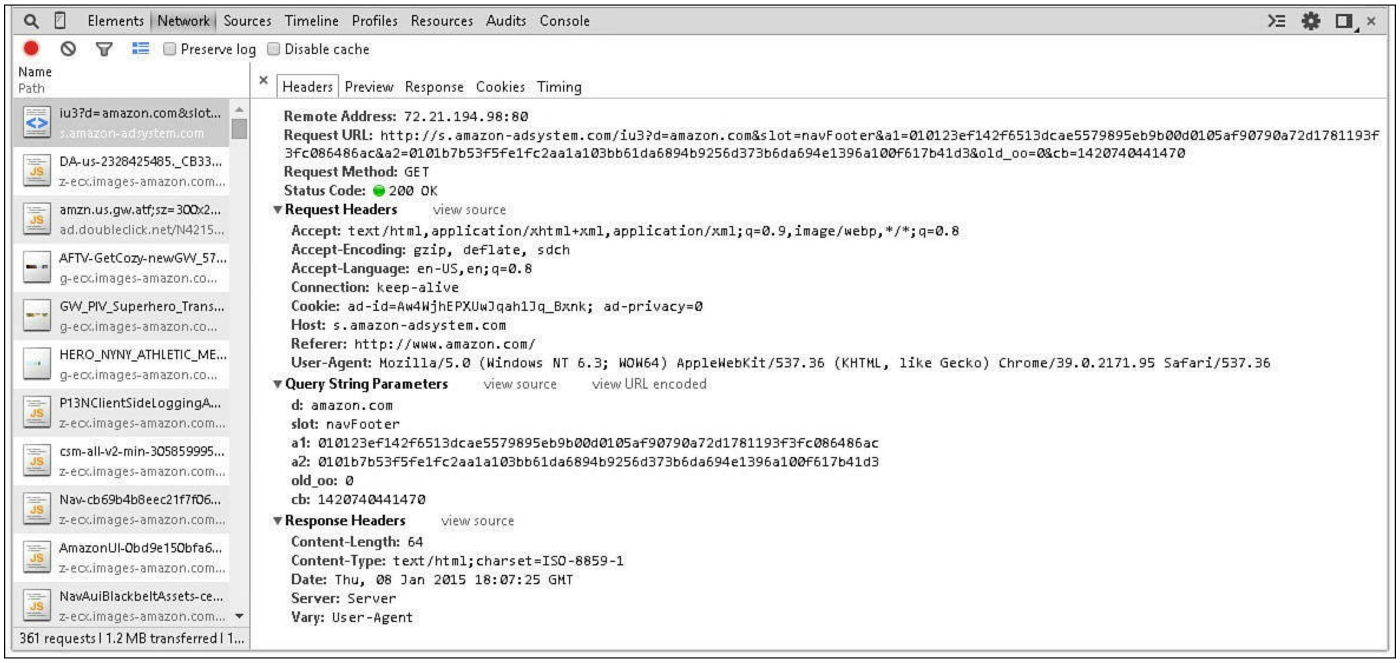
Note

An option at the top of the Network tab allows you to disable the browser cache. This option can be very useful when you are updating files on the web server to debug and fix issues. When this option is checked, the browser will always retrieve the latest from the web server.

With some complex web pages, you may have too much traffic to try to debug all the requests. The filter options in the Net tab allow you to view only certain types of requests, such as HTML, CSS, or JS. The XHR filter stands for `XMLHttpRequest`, which is the communication used in AJAX. Selecting the XHR filter will show only AJAX communication.

When you expand a request, as shown in Figure 2.26, you get a lot of additional information about the request. What tabs are available in the expanded request depend on the request type and the response type, but here are some of the most useful items:

- **Headers**—Displays the HTML request and response headers that were sent. This is very useful if you are accessing a service via AJAX that requires specific headers to be sent.
- **Preview**—This shows a preview of the request data. If this is a POST request, it shows you the values of the parameters sent in the POST request to the server.
- **Response**—This will vary, depending on what the response is. For example, if you are downloading a JavaScript file from the web server, this displays the raw JavaScript; for HTML files, it shows the HTML.
- **Cookies**—Displays the cookies and values involved in the request.
- **Timing**—Displays timing information on the request, including send time, wait time, and download time.



```
iu3?d=amazon.com&slot...
s.amazon-adsystem.com
DA-us-2328425485._CB33...
z-ecx.images-amazon.com...
amzn.us.gw.atf;sz=300x2...
ad.doubleclick.net/N4215...
AFTV-GetCozy-newGW_57...
g-ecx.images-amazon.co...
GW_PIV_Superhero_Trans...
g-ecx.images-amazon.co...
HERO_NYNY_ATHLETIC_ME...
g-ecx.images-amazon.co...
P13NClientSideLoggingA...
z-ecx.images-amazon.com...
csm-all-v2-min-305859995...
z-ecx.images-amazon.com...
Nav-cb69b4b8eec21f7f06...
z-ecx.images-amazon.com...
AmazonUI-0bd9e150bfa6...
z-ecx.images-amazon.com...
NavAuiBlackbeltAssets-ce...
z-ecx.images-amazon.com...
361 requests | 1.2 MB transferred | 1...
```

```
Remote Address: 72.21.194.98:80
Request URL: http://s.amazon-adsystem.com/iu3?d=amazon.com&slot=navFooter&a1=010123ef142f6513dcae5579895eb9b00d0105af90790a72d1781193f
3fc086486ac&a2=0101b7b53f5fe1fc2aa1a103bb61da6894b9256d373b6da694e1396a100f617b41d3&old_oo=0&cb=1420740441470
Request Method: GET
Status Code: ● 200 OK
▼Request Headers     view source
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
  Accept-Encoding: gzip, deflate, sdch
  Accept-Language: en-US,en;q=0.8
  Connection: keep-alive
  Cookie: ad-id=Aw4WjhEPXUwJqah1Jq_Bxnk; ad-privacy=0
  Host: s.amazon-adsystem.com
  Referer: http://www.amazon.com/
  User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36
▼Query String Parameters     view source     view URL encoded
  d: amazon.com
  slot: navFooter
  a1: 010123ef142f6513dcae5579895eb9b00d0105af90790a72d1781193f3fc086486ac
  a2: 0101b7b53f5fe1fc2aa1a103bb61da6894b9256d373b6da694e1396a100f617b41d3
  old_oo: 0
  cb: 1420740441470
▼Response Headers     view source
  Content-Length: 64
  Content-Type: text/html;charset=ISO-8859-1
  Date: Thu, 08 Jan 2015 18:07:25 GMT
  Server: Server
  Vary: User-Agent
```

FIGURE 2.26 Expanding the request provides additional tabs with more information about the request and server response.

Summary

In this lesson, you learned a myriad of ways to debug problems in your dynamic web pages. You learned how to output messages from your scripts to the JavaScript console. You learned how to use the Element inspector to see the HTML elements that the browser has built while loading the web page.

You also followed several example of debugging problems in HTML, CSS, and JavaScript. The methods you learned throughout this lesson will be very helpful to you as you finish this book and in future projects because they will save a lot of time and frustration with simple syntax problems that always seem to creep up.

Q&A

Q. Is there a way to debug server-side scripts?

A. Yes, there is. It is really beyond the scope of this book; however, most good languages have a method of remotely debugging problems. If you are trying to debug PHP server-side scripts, look into the capabilities of ZEND at www.zend.com/en/community/pdt. If you are working with Python server-side scripts, check into using PyDev at pydev.org.

Q. Is there a way to debug cookies?

A. As far as debugging cookies, all you need to know is whether cookies are enabled, which cookies are set in the browser, what the cookie values are, and when they expire. All that information can be found in the Cookies tab in the developer tools. There are similar features with both Chrome and Internet Explorer in their developer consoles.

Workshop

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

Quiz

1. Where do you go in the developer tools to find what the background-color CSS property value is for a specific <div> tag?
2. Where would you go in the developer tools to see the available size of the browser window?
3. How do you get your JavaScript to stop executing on a specific line of code?
4. When JavaScript execution is halted, how do you find the values of a variable?

Quiz Answers

1. The Styles tab of the Element inspector with that `<div>` tag selected.
2. Type `window` in the Console view.
3. Set a breakpoint in the Sources debugger tab.
4. Look in the Watch tab of the Sources debugger.

Exercises

1. Modify the debug.html code to output the value of `cnt` to the JavaScript console by adding the following code at line 10:

```
console.log("cnt=%d",cnt);
```

2. Use the Net traffic as you browse the traffic from some different pages. Expand some of the requests and look at the data represented in some of the tabs. This can help you understand the ebb and flow of browser to web server traffic a bit better.

Lesson 3. Understanding Dynamic Web Page Anatomy

What You'll Learn in This Lesson:

- How to build a basic dynamic web page
- Where to add CSS and JavaScript in web pages
- What the difference is between block and inline elements
- How to add images to web pages
- How to build web forms
- How to add links to specific spots in a web page
- How to build tables into your web pages
- What HTML5 SVG graphics can do
- Ways to use HTML5 to prep for dynamic audio and video

Throughout the rest of this book, you will use a lot of HTML, CSS, jQuery, and JavaScript. For that purpose, this lesson is designed to accomplish two tasks. The first is to familiarize you, in case you are not already familiar, with some important basics of HTML so that you will be able to easily understand the examples.

The second is to help you understand how to design your HTML to make it easier to add dynamics to your web pages later using jQuery and JavaScript. Understanding how to design your HTML elements will make it easier later to add some cool effects and dynamically update data stored in lists or tables.

The following sections discuss the basics of HTML and how they relate to jQuery and JavaScript.

Using HTML/HTML5 Elements to Build a Dynamic Web Page

You have already seen some examples of HTML code in Lessons 1 and 2. Now it's time to delve a bit deeper in understanding the syntax and which HTML elements and attributes are important to dynamic web pages.

To properly build dynamic web pages, you need to understand the HTML syntax, some fundamental HTML elements, and how to organize and structure those elements. The following sections cover those topics.

Understanding HTML Structure

HTML documents are composed of three main parts: the `<!DOCTYPE>` tag, the head, and the body. Each of these parts plays a specific role in helping the browser to render the HTML document into a web page.

The `<!DOCTYPE>` tag must be the first statement in the HTML file; it tells the browser how to read the rest of the file. Although this tag is not strictly required, it is a good idea to include it in your HTML documents. There are several forms of the `<!DOCTYPE>` element; a few of them are listed next:

- **HTML5**—Version used for HTML5 documents:

```
<!DOCTYPE html>
```

- **HTML 4.01 Strict**—Enforces strict compliance with the HTML 4.01 standard and will remove deprecated elements such as `<font>`:

Click here to view code image

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

- **HTML 4.01 Transitional**—Relaxed compliance with the HTML 4.01 standard and will allow deprecated elements such as `<font>`:

Click here to view code image

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.
w3.org/TR/html4/loose.dtd">
```

The `head` and `body` components are contained in the HTML tag. The purpose of the `head` element is to contain elements that are used in parsing the HTML document but are not rendered inside the browser window, such as scripts and metadata. The purpose of the `body` tag is to contain elements that will be rendered to the browser window and viewed by the user.

The following code shows an example of the basic HTML document structure with one head element and one body element:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
  </head>
  <body>
    <h1>New Page</h1>
  </body>
</html>
```

Syntax is everything when you are working with HTML elements. If the browser cannot parse the document correctly, the page will not be rendered correctly to the user.

Tip

Most web development IDEs and editors have built-in error checking as well as code completing for HTML syntax. These are shown in a variety of ways, from highlighted or underlined code, to warning icons inline with the code, to a list of potential problems in a different pane. Pay attention to what the IDE is trying to tell you about possible problems.

An HTML element is composed of the following three main components:

- **Tag**—The tag is enclosed in <> characters and tells the browser what type of HTML element to parse and render. For example, the tag for a paragraph element is `<p>`, the tag for an unordered list is `<ul>`, and the tag for a list item is `<li>`.
- **Content**—The content portion of the HTML element can be another HTML element, simple text, or nothing. To define what is contained inside an HTML element, a closing tag is added at the end of the content. The following example illustrates that perfectly. Notice that there are several `<li>` elements—each with an opening tag `<li>`, some content, and then a closing tag `</li>`. The content of the `<li>` tags is the text in between. There is also an opening and closing `<ul>` and `</ul>` tag. The content of the `<ul>` element is all the `<li>` elements in between:

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

- **Attribute**—Attributes provide a way of attaching additional information about the element that can be used by the browser. This information can be used to define how the element is rendered by the browser or provide a way to identify or classify the element. The following shows an example of adding an `align` attribute to a paragraph element to tell the browser to center the text when rendering it. Notice that the attribute value is assigned using an equal sign and that the value is enclosed in double quotes:

Click here to view code image

```
<p align="center">This is some centered text.</p>
```

Note

If the value of an attribute requires a double quote, the normal assignment of attribute="value" will not work. In these instances, you can use single quotes in the assignment. For example, attribute='some "quoted" value'.

If an element does not have an end tag, you can include the / at the end of the first tag and completely leave out the end tag. For example, both of the following are acceptable to the browser:

```
<p align="center"></p>
<p align="center" />
```

Implementing HTML Head Elements

The HTML `<head>` element is designed as a container for nonvisual elements of the web page. The tags in the <head> element are parsed by the browsers into the DOM, but they are not rendered to the browser window.

The following sections describe some of the more important `<head>` elements and how they relate to using jQuery and JavaScript.

<title>

The `<title>` element is supposed to be required in all HTML documents. The browser will still render the page without it, but there are many reasons to include the title element in your web pages, including that the `<title>` element does the following:

- Defines the title that is displayed in the browser toolbar/tab.
- Provides the title that the web page is listed as when it is added to favorites.
- Determines what is displayed as the title when the web page is displayed by search engines.

The following code shows an example of adding a title to the web page:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page</title>
```

The value of the `<title>` element can easily be changed using jQuery and JavaScript. For example, the code in Listing 3.1 uses JavaScript to change the title of the web page after it is loaded.

LISTING 3.1 page_title.html JavaScript Code Changing the <title> Value After the Page Has Loaded

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
  <title>Page</title>
  <meta charset="UTF-8" />
  <script>
   function appendTitle(newTitle){
    document.title += newTitle;
   }
  </script>
 </head>
 <body onload="appendTitle('... Loaded')">
 </body>
</html>
```

Notice in Figure 3.1 that the Title in the tab has changed to read Page...Loaded.

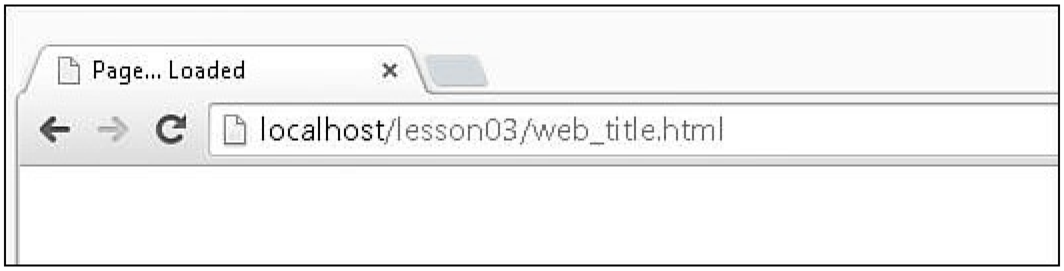


FIGURE 3.1 The JavaScript function has changed the title of the web page that is displayed in the browser tab.

<meta>

The `<meta>` tag has many uses. Browsers use the `<meta>` tags to determine such things as what character sets to use when rendering the web page. Search engine crawlers use the `<meta>` tags to determine the purpose of the content of the web page for better optimized searches. The best way to introduce it is to show you some examples of syntax.

The meta tag required to set the character set in HTML5 web pages is as follows:

```
<meta charset="UTF-8">
```

The `<meta>` tag required to set the character set in HTML 4 web pages is as follows:

Click here to view code image

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
```

The `<meta>` tag to define keywords for search engines is as follows, where the value of content is the keywords you want the search engine to use when finding the web page:

Click here to view code image

```
<meta name="keywords" content="HTML, CSS, jQuery, JavaScript">
```

The `<meta>` tag to tell the browser to refresh the web page every 60 seconds is as follows:

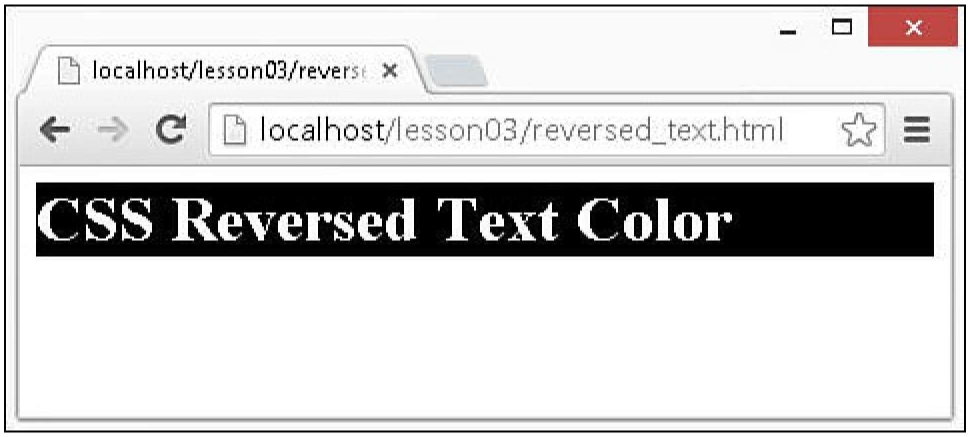
Click here to view code image

```
<meta http-equiv="refresh" content="300">
```

<style>

The `<style>` tag allows you to add CSS code directly inside the HTML document. Everything included inside the `<style>` tag is treated like a CSS document and is used by the browser to render the web page.

As an example of the syntax, the Listing 3.2 HTML file includes CSS to turn the background of `<h1>` elements black and the foreground white, as shown in Figure 3.2.


localhost/lesson03/reverse
localhost/lesson03/reversed_text.html

CSS Reversed Text Color

FIGURE 3.2 This `<h1>` element has had its style reversed by a CSS style script in the HTML document.

LISTING 3.2 reversed_text.html Adding CSS to an HTML Document Using the <style> Tag

Click here to view code image

```
<html>
  <head>
    <meta charset="UTF-8">
    <style type="text/css">
      h1 {
        background-color:black;
        color:white;
      }
    </style>
  </head>
  <body>
    <h1>CSS Reversed Text</h1>
  </body>
</html>
```

<script>

The <script> tag enables you to add JavaScript code directly inside the HTML document or link to a separate external JavaScript file. If you include inline JavaScript code, everything included inside the <script> tag is treated like a JavaScript document and loaded into the browser when the HTML document is parsed. If you include a link to an external file, that file is downloaded from the web server and loaded by the browser.

Caution

When using the <script> tag, scripts are loaded in the order in which they are parsed. That means that any subsequent scripts that have the same global variable or function names will overwrite the ones already loaded. When adding multiple scripts to a web page, you should be careful to not override global variable and function names that you do not intend to.

For example, the HTML document in Listing 3.3 includes two <script> elements: one to load the external jQuery library and the second with a simple JavaScript and jQuery function to turn the background of <h1> elements black and the foreground white, as shown in Figure 3.3.

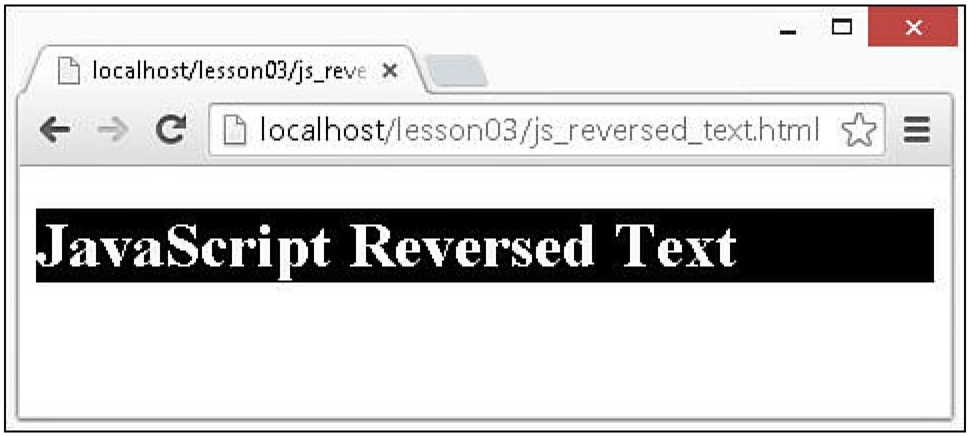

localhost/lesson03/js_reve
localhost/lesson03/js_reversed_text.html
JavaScript Reversed Text

FIGURE 3.3 This <h1> element has had its style reversed by a JavaScript function using jQuery directly in the HTML document.

LISTING 3.3 js_reversed_text.html Adding JavaScript and jQuery to an HTML Document Using the <script> Tag

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script>
      function reverseText(){
        $("h1").css("background-color", "black");
        $("h1").css("color", "white");
      }
    </script>
  </head>
  <body onload="reverseText()">
    <h1>JavaScript Reversed Text</h1>
  </body>
</html>
```

<noscript>

Although it doesn’t happen very often anymore, occasionally people disable JavaScript on their web browsers. If JavaScript is disabled, any JavaScript, and consequently jQuery scripts, will not be executed. This can provide a very bad experience for users.

The `<noscript>` tag allows you to provide elements that will be rendered before items in the `<body>` element. This allows you to display warning messages, or alternative forms of the page in the event that JavaScript is disabled.

The `<noscript>` tag supports the same child element types that the `<body>` tag supports. For example, the following code loads the JavaScript file if JavaScript is enabled, but if not, it adds a `<h3>` heading to the top of the web page that warns users that the web page will not function properly:

Click here to view code image

```
<head>
  <script src="DynamicPage.js"></script>
  <noscript>
    <h3>This web page uses JavaScript but it is disabled in your browser.
        The web page may not function properly.</h3>
  </noscript>
</head>
```

<link>

The `<link>` tag allows you to link separate resources to the current document. For the most part, it is used for CSS files. That means that when the browser loads the HTML document, linked CSS files are downloaded from the web server and loaded as well. For example, the following head block will load two different CSS documents:

Click here to view code image

```
<head>
  <link rel="stylesheet" type="text/css" href="styleA.css">
  <link rel="stylesheet" type="text/css" href="styleB.css">
</head>
```

In addition to CSS files, you can link things like icons, help documents, licenses, and searches to the web page. For example, the following code links an icon that is displayed in the browser tab when the web page is loaded:

Click here to view code image

```
<link rel="icon" type="image/png" href="http://example.com/myicon.png">
```

Adding HTML Body Elements

Most of the HTML elements that are added to the body block are rendered by the browser and displayed on the screen to the user. Each of the HTML body element tags provides different options that will help you show information on the web page in a meaningful and elegant way.

In the following sections, you'll see some specific attributes that are useful to apply to HTML elements when programming in jQuery and JavaScript. You also learn the syntax of some of the more common elements that you will see throughout the rest of the book so that the examples will be easier to follow.

Using Important Body Element Attributes

Each of the different body elements will have some attributes that are specific to the element only. However, some attributes are important and common for all elements. Those tags are the following:

- **id**—This is a string identifier that must be unique among all elements on the web page. This is used by CSS, JavaScript, and jQuery to identify, access, and modify a specific element on the web page.
- **class**—The `class` attribute allows you to specify a grouping of multiple elements. This option is used heavily with CSS stylizing to set styles such as background-color for several HTML elements at once. It is also useful in jQuery to select multiple elements based on their class value.
- **style**—The `style` attribute allows you to place CSS definitions directly in the HTML code inside of the HTML object. The values in the `style` override all other CSS directives.

The following code illustrates the syntax to add id, class, and style attributes to HTML elements. Notice that the classes "`heading`" and "`content`" are used multiple times, but the `id` attribute is different for every element. Also notice that for the first heading, an additional style attribute was added to force the first heading to always be bold:

Click here to view code image

```
<div id="Div1">
  <p id="Div1Heading" class="heading" style="font-weight:bold">Heading
Text</p>
```

```
  <p id ="Div1Content" class ="content">Some Content</p>
</div>
<div id="Div2">
  <p id="Div2Heading" class="heading">Another Heading Text</p>
  <p id ="Div2Content" class ="content">Some More Content</p>
</div>
```

Understanding Block Versus Inline Elements

Remember that the web page is rendered to the display screen by the browser parsing each element in the HTML document and rendering it along with any children. The difference between block and inline elements is how much room the browser gives the element.

Block elements are given the full width available to render the item in the browser. That means that any elements before will be displayed above, and elements that come after will be displayed below. `<div>` and `<p>` are examples of block elements.

Inline elements are given only enough width to display the element on the page. That means that multiple inline elements are displayed side by side in the browser as long as there is enough room. `<span>`, `<a>`, and `<img>` are some examples of inline elements.

The following code and the rendered version shown in Figure 3.4 illustrate the difference between block and inline tags. Notice that the `<p>` tags take up the full width of the browser screen, and the `<span>` tags take only enough space to display the text and margin:

```
<p>Paragraph A</p>
<p>Paragraph B</p>
<p>Paragraph C</p>
<span>Span A</span>
<span>Span B</span>
<span>Span CA</span>
```

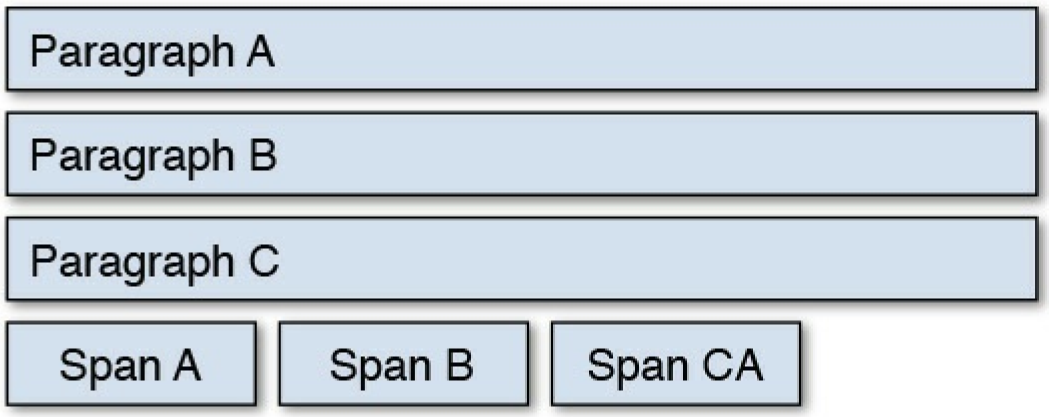


FIGURE 3.4 The block element `<p>` takes up the entire width, but the inline element `<span>` uses only the width that it needs.

Although elements are block or inline by default, you can always change them by setting the `display` CSS style. For example, setting a `<span>` display to block as follows will make it behave like a `<div>` element:

Click here to view code image

```
<span style="{display:block;}">Span Text</span>
```

The following is a list of values that you can set the display style to:

- **block**—Renders the element as a block element.
- **inline**—Renders the element as an inline element.
- **none**—Will not render the element to the screen even though it is parsed and exists in the DOM.

You can also force items to be rendered below other items by separating them by using one of the following line break tags:

- **<br>**—Adds a line break that causes the content after to be rendered below the current content, even if they are inline.
- **<hr>**—Similar to `<br>` but also causes the browser to render a horizontal line under the current content.

These tags do not require a closing tag. For example, the following code creates a set of three `<span>` elements that are separated by first the `<br>` tag and then the `<hr>` tag. The rendered results are shown in Figure 3.5. Notice that `<br>` inserted a line break and `<hr>` inserted a line break and rendered a line between the content:

Click here to view code image

```
<span>Text on</span>
<span> Line 1</span><br>
<span>Text on Line 2</span><hr>
<span>New section on Line 3</span>
```

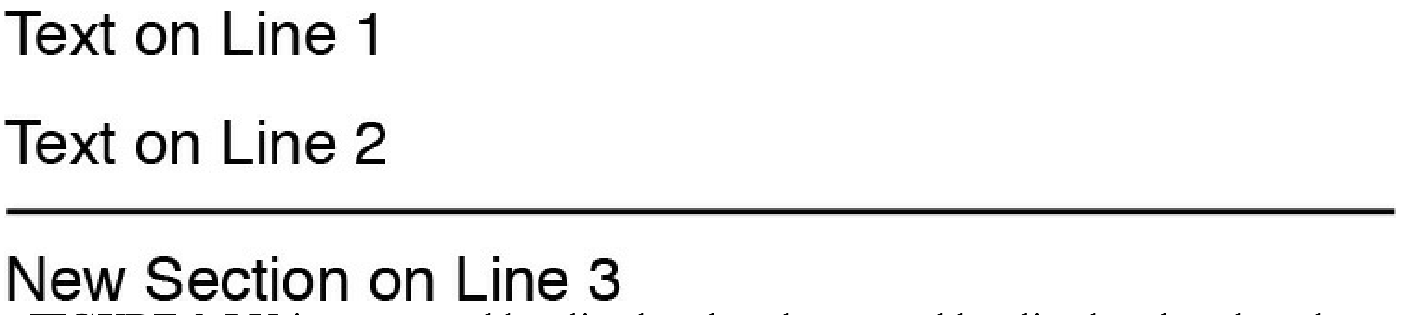


FIGURE 3.5 Using `<br>` adds a line break and `<hr>` adds a line break and renders a line between the content.

Creating Container Elements

Container elements are used for a variety of purposes, such as grouping and formatting sets of elements or text. This allows you to apply formatting and dimensions to the container and have it affect all the items within. Although just about any HTML element could be used as a container, three main ones are `<p>`, `<div>`, and `<span>`.

Container elements by themselves do not alter the appearance of the text or data within them. For example, the text inside all the following will render the same in the browser:

```
Some Text
<p>Some Text <p>
<div>Some Text </div>
<span>Some Text</span>
```

Try it Yourself: Using Container Elements to Group and Style

Container elements are typically used to group elements for layout and style settings. The following exercise creates the code in Listing 3.4. The code is designed to introduce you to grouping and styling elements:

1. Start by creating a folder in your Eclipse project called lesson03.

2. Create a new HTML document in the lesson03 folder called css_styling.html (to match the filename on the book's website).

3. Add the appropriate `<html>`, `<head>`, and `<body>` tags.

4. Add the following `<div>` element with the `class` attribute set to "`heading`".

This will be the container for your heading items:

Click here to view code image

```
<div class="heading">
  <p>Heading A</p>
  <p>Heading B</p>
  <p>Heading C</p>
</div>
```

5. Add the following second `<div>` element with the `class` attribute set to "`content`". This is the container for your content items:

Click here to view code image

```
<div class="content">
  <p>Paragraph A</p>
  <p>Paragraph B</p>
  <p>Paragraph C</p>
</div>
```

6. Save the file and load the following URL in the web browser to view the unformatted code, as shown in Figure 3.6:

Click here to view code image

http://localhost/lesson03/css_styling.html

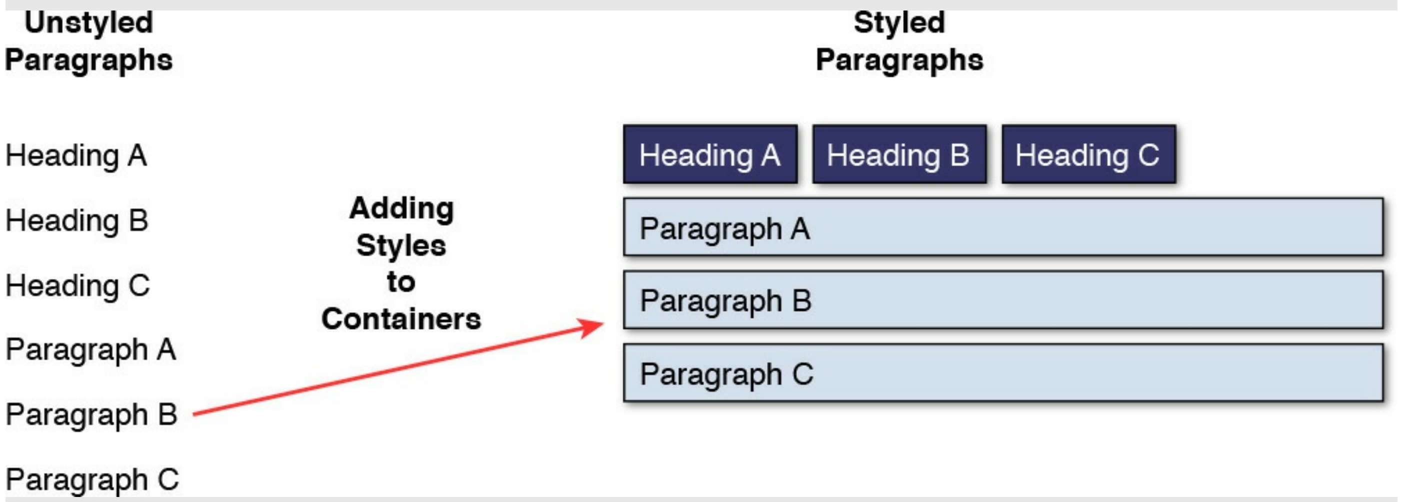


FIGURE 3.6 Using `<div>` container elements makes it easy to change the look and behavior of `<p>` elements in different areas of the web page.

7. Go back to Eclipse and add a `<style>` tag to the header of the file.

8. Inside the `<style>` tag, add the following code that formats the content elements with a gray background and a bit of padding around the edges:

Click here to view code image

```
.content p{
  background-color:#C0C0C0;
  padding: 3px;
}
```

9. Add the following additional rule to style the header elements. This rule changes the `<p>` elements in the heading container to be inline, with bold white text and a black background:

Click here to view code image

```
.heading p{
  display: inline;
  background-color:black;
  color: white;
  font-weight:bold;
  padding:3px;
}
```

10. Save the document and then reload it in the web browser. Notice in Figure 3.6 that the elements in the "`heading`" class `<div>` element are rendered as inline with a black background and white text, and the elements in the "content" class `<div>` element are rendered with a gray background and remain block style.

LISTING 3.4 css_styling.html Adding CSS to an HTML Document Using the

<style> Tag

Click here to view code image

```
<html>
  <head>
    <title>CSS Styling</title>
    <meta charset="UTF-8">
    <style type="text/css">
      .content p{
        background-color:#C0C0C0;
        padding: 3px;
      }
      .heading p{
        display: inline;
        background-color:black;
        color: white;
        font-weight:bold;
        padding:3px;
      }
    </style>
  </head>
  <body>
    <div class="heading">
      <p>Heading A</p>
      <p>Heading B</p>
      <p>Heading C</p>
    </div>
    <div class="content">
      <p>Paragraph A</p>
      <p>Paragraph B</p>
      <p>Paragraph C</p>
    </div>
  </body>
</html>
```

You will be using the container elements heavily in JavaScript and jQuery development to group, format, and control various aspects of the dynamic web page.

Adding Link Elements

Web pages often contain a series of links to additional pages. Links are easy to add to a web page using the anchor tag `<a>`. The `<a>` tag has three purposes:

- **Link to External Web Page**—Displays a link on the web page. When the user clicks the link, the browser requests the web page in the link from the server. The following is an example of adding a link to an external web page:

Click here to view code image

```
<a href="http://www.dayleycreations.com/">Dayley Creations</a>
```

- **Link to Location on Current Web Page**—Displays a link on the web page. When the user clicks the link, the browser changes the scroll down to the location so that the anchor linked to is visible:

Click here to view code image

```
<a href="#local link">Link to Some Text Below</a>
```

- **Web Page Anchor**—Adds a link inside the web page that can be directly linked to either by the current page or by another web page. To define an anchor, you need to use the `id` attribute. For example, to add the anchor for the preceding link, use the following:

Click here to view code image

```
<a id="local_link">Page Anchor</a>
```

Web page anchors can be useful when using JavaScript to build the link elements. For example, you can dynamically change the `url` attribute of the link to control where the link takes the user.

Using Image Elements

One of the greatest aspects of web pages is the capability to display images along with other graphics and text. Images are displayed using the `<img>` tag.

The actual image file on the web server to display in the `<img>` tag is determined by the `src` attribute. The size of the image on the screen is determined by the `height` and `width` attributes.

The following code shows some examples of displaying an image with different sizes and whitespace, as shown in Figure 3.7. Notice that when only the `height` or `width` is specified, the image is scaled to keep the aspect ratio, but when both are specified, the image is stretched to fit the specific height and width settings:

Click here to view code image

```
<img src="/images/peak.jpg" height="200px"/>
<img src="/images/peak.jpg" width="200px"/>
<img src="/images/peak.jpg" height="200px" width="200px"/>
```

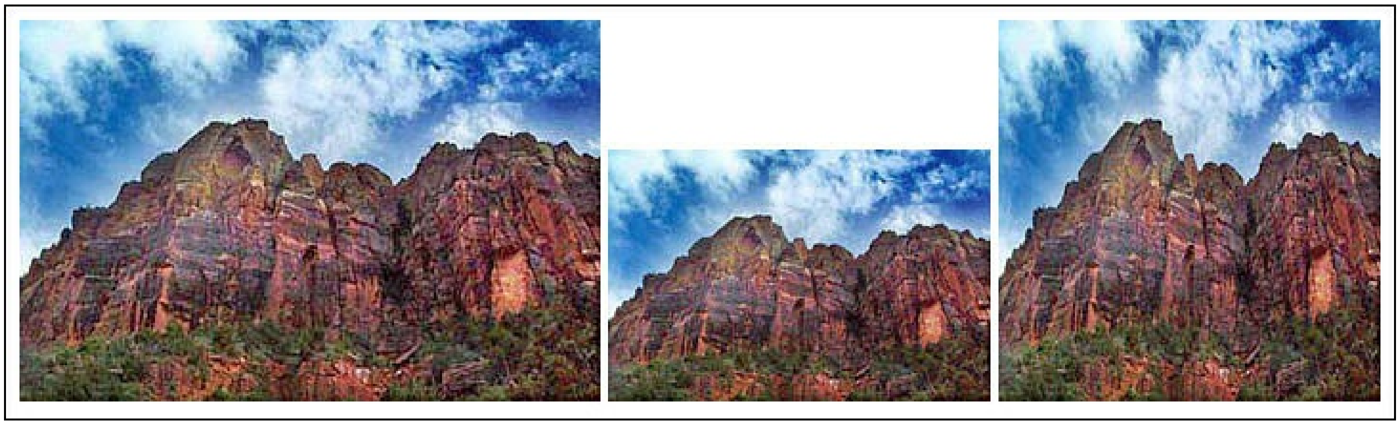


FIGURE 3.7 Using `<img>` to add images to a web page and specify height and width.

You will also be dealing with images frequently when creating dynamic web pages using jQuery and JavaScript because they allow you to quickly resize, move, hide, and add them to the web page. This allows your images to interact with mouse actions and other input from users.

Applying List Elements

List elements allow you to group a set of items together and have the browser automatically format them with bullet or numbers. Ordered lists are created using the `<ol>` tag, and bulleted lists are created using the unordered list tag `<ul>`. Items within the list are contained in the `<li>` tag. For example, the following code renders to Figure 3.8:

```
<ul>
  <li>Yellowstone</li>
  <li>Yosemite</li>
  <li>Glacier</li>
  <li>Arches</li>
  <li>Zion</li>
</ul>
```

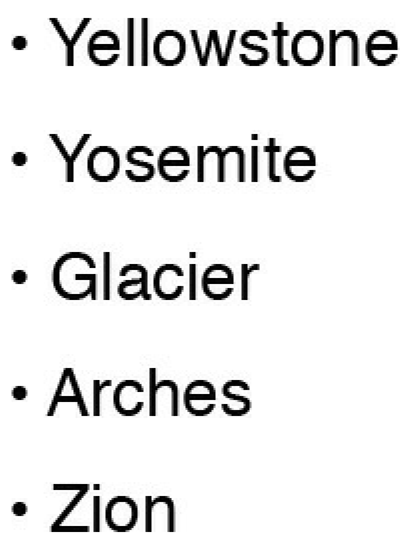

- Yellowstone
- Yosemite
- Glacier
- Arches
- Zion

FIGURE 3.8 Adding a list to a web page using the `<ul>` and `<li>` tags.

Creating Table Elements

Table elements are some of the more complex HTML elements. They are used to organize other elements on the screen in a series of rows and columns. They can be used for a variety of purposes, from laying out entire web pages to a simple table of data. This section covers the basics that you need to know for this book.

Tables are constructed using a series of tags that define the table, headers, body, and cells. The following lists the various tags that can be used when constructing a table:

- **<table>**—Acts as the container element for all other table elements and defines the overall table.
- **<thead>**—This element is not required, but it can be useful in that it allows you to group and define the header elements in a table, or refer to the parent of header elements directly from jQuery or JavaScript via the `id` attribute.
- **<tbody>**—This is also not required, but it is useful to define the overall body of the table or refer to the parent of body elements directly from jQuery or JavaScript via the `id` attribute.
- **<th>**—Defines a single header cell in a table.
- **<td>**—Defines a cell in a table.
- **<tr>**—Defines a single row in a table. This acts as a container for either `<th>` and/or `<td>` elements or cell elements.
- **<caption>**—Provides a container for caption content. The caption content can be either text or other HTML elements.
- **<colgroup>**—Enables you to place one or more columns in a table into a group that can be formatted together. This is also useful if you want to access a specific set of columns from jQuery or JavaScript via the `id` attribute.
- **<col>**—Specifies the column properties for a single column within a `<colgroup>` element.
- **<tfoot>**—This element is not required, but it can be useful in that it allows you to group and define the footer elements in a table or refer to the parent of footer elements directly from jQuery or JavaScript via the `id` attribute.

In HTML 4, several attributes can be set on the table elements. Many of those have been deprecated in HTML5 because they were used for formatting size, alignment, and color, which should be done with CSS styles. The following is a list of the more important attributes that you will still need to use on table elements:

- **border**—Specifies whether the table cells should have borders.
- **colspan**—Specifies the number of columns a cell should span. This allows an individual `<td>` or `<th>` element to occupy more than one column in the table.
- **rowspan**—Specifies the number of rows a cell should span. This allows an individual `<td>` or `<th>` element to occupy more than one row in the table.
- **headers**—Added to `<td>` or `<th>` elements. Allows you to specify the `id` value of one or more `<th>` elements. This creates an association to the header element that can be accessed from jQuery and JavaScript.

Try it Yourself: Creating HTML Tables

The easiest way to explain tables is to show you an example of creating a basic table. Use the following steps to create the table code in Listing 3.5:

1. Create a new HTML document in the lesson03 folder called tables.html (to match the filename on the book's website).
2. Add the appropriate `<html>`, `<head>`, and `<body>` tags.
3. Add the following `<table>` element with the `border` attribute set to 1:

Click here to view code image

```
<table border=1></table>
```

4. Add the following `<caption>` element inside the `<table>` element to give the table a caption:

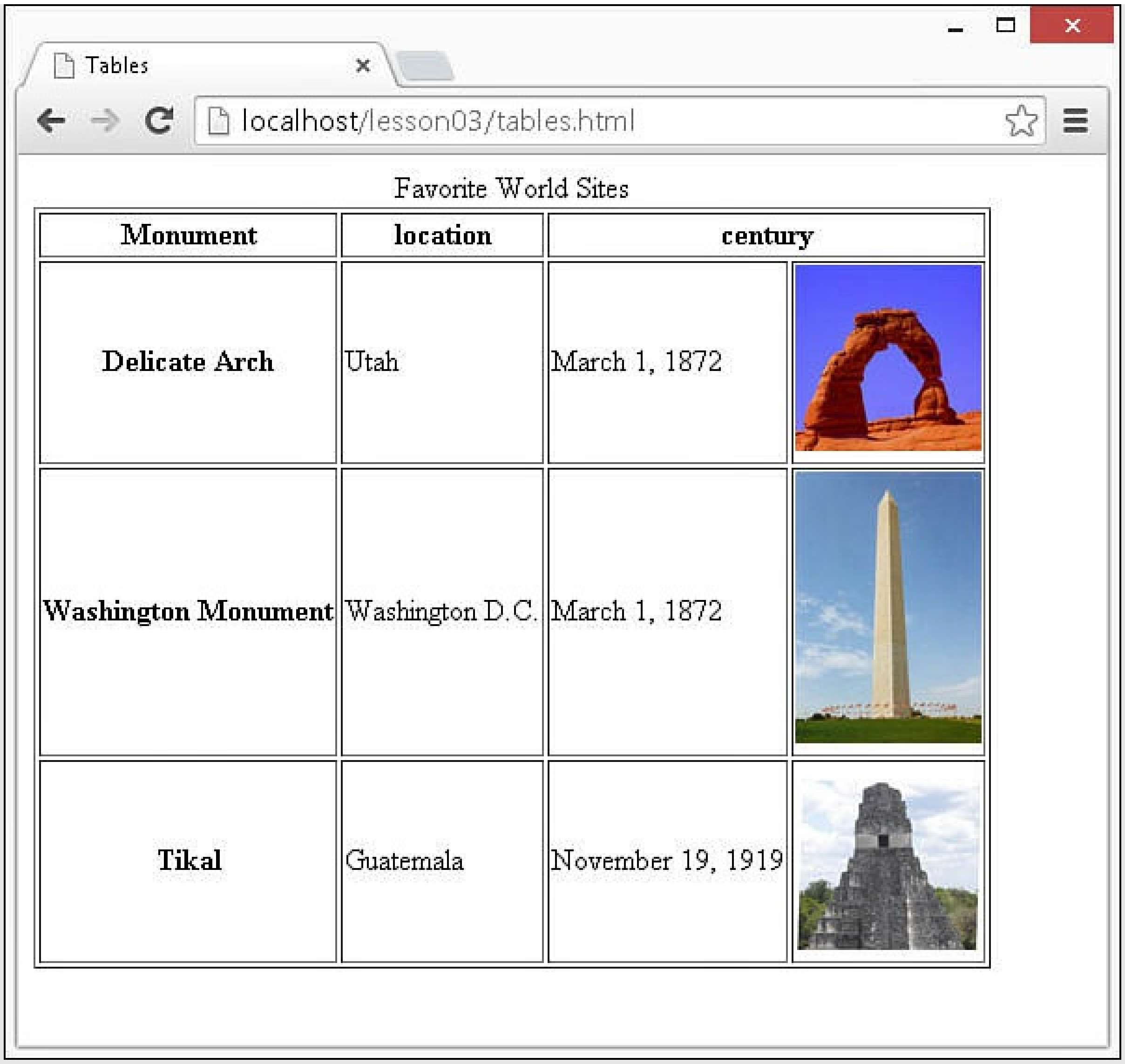
Click here to view code image

```
<caption>Favorite World Sites</caption>
```

5. Create the table headers by adding the following `<thead>` element to the `<table>` element. Notice that the third `<th>` sets the `colspan` to 2 so that it will cover the last two columns, as shown in Figure 3.9:

Click here to view code image

```
<thead>
  <th>Monument</th>
  <th>location</th>
  <th colspan=2>century</th>
</thead>
```



Favorite World Sites

Monument	location	century	
Delicate Arch	Utah	March 1, 1872	
Washington Monument	Washington D.C.	March 1, 1872	
Tikal	Guatemala	November 19, 1919	

FIGURE 3.9 An HTML table example showing headers in rows and columns and a caption.

6. Add the following `<tr>` elements. Notice in the example that follows that each of these `<tr>` elements contains four cells. The first cell is a row header because it uses the `<th>` tag. The next two cells are simple text in `<td>` elements, but in the fourth cell, the `<td>` element contains an `<img>` element (the .jpg files can be found on the book's website):

Click here to view code image

```
<tr>
  <th>Delicate Arch</th>
  <td>Utah</td>
  <td>March 1, 1872</td>
  <td><img src="/images/arch.jpg" width="100" />
</tr>
```

7. Save the document and open the following URL in the web browser to view the rendered results shown in Figure 3.9:

Click here to view code image

```
http://localhost/lesson03/tables.html
```

LISTING 3.5 tables.html HTML Generating a Table with Headers, Rows, and Columns

Click here to view code image

```
<!DOCTYPE html>
<html>
<head>
  <title>Tables</title>
  <meta Charset="UTF-8">
</head>
<body>
  <table border=1>
    <caption>Favorite World Sites</caption>
    <thead>
      <th>Monument</th>
      <th>location</th>
      <th colspan=2>century</th>
    </thead>
    <tr>
      <th>Delicate Arch</th>
      <td>Utah</td>
      <td>March 1, 1872</td>
      <td><img src="/images/arch.jpg" width="100" />
    </tr>
    <tr>
      <th>Washington Monument</th>
      <td>Washington D.C.</td>
      <td>March 1, 1872</td>
      <td><img src="/images/washington.jpg" width="100" />
    </tr>
    <tr>
      <th>Tikal</th>
      <td>Guatemala</td>
      <td>November 19, 1919</td>
      <td><img src="/images/pyramid.jpg" width="100" />
    </tr>
  </table>
</body>
</html>
```

Implementing Form Elements

Another extremely useful set of tags in HTML are the form tags. These tags provide the building blocks to render user input forms that enable you to gain input from the user. Browsers know how to render the form elements to display things like text boxes, buttons, and lists.

Following is a list of the more common form tags you will use in jQuery and JavaScript to create dynamic form elements:

- **<form>**—This is the root element of the form. All other elements are contained inside.
- **<fieldset>**—Acts as a container allowing you to group several form elements together. In HTML5, you can set the disabled attribute of the fieldset to disable it from user input. You can also use jQuery and JavaScript to access the children of the fieldset and modify them from code.
- **<legend>**—Tells the browser to add a caption to the fieldset.
- **<label>**—Allows you to tie a label element to an `<input>` element in the form by setting the `for` attribute to the `id` attribute value of the `<input>` element. This adds the advantage that the mouse events for the input element are also triggered for the `<label>`.
- **<input>**—The input tag is really a wrapper for several types of input components. The input that is rendered by the browser will depend on the value of the `type` attribute. The following is a list of type elements that can be specified in the input tag: `button`, `checkbox`, `color`, `date`, `datetime`, `email`, `file`, `hidden`, `image`, `month`, `number`, `password`, `radio`, `range`, `reset`, `search`, `submit`, `tel`, `text`, `time`, `url`, and `week`.
- **<textarea>**—Different from the other inputs, this provides a resizable area where the user can type in multiple lines of text. The `cols` and `rows` attributes define the initial size.
- **<select>**—Acts as a container for a drop-down list of items.
- **<option>**—Used inside the `<select>` element to define a single item in the list. The `value` attribute of the selected item is also the `value` of the `<select>` element.
- **<button>**—Defines a clickable button element. The `type` attribute of button tells the browser whether to use the button to `submit` the form, `reset` the values, or act as a simple input `button`.

The form elements have a different set of attributes that provide access to and modify the look and behavior of the element. The following are some of the more important ones:

- **name**—This option specifies a name that can be used when submitting the form.

- **value**—This option specifies the value of the form element. For elements such as `text <input>`, what you specify for `value` will be added as text in the rendered text box. The `value` attribute is also accessible via JavaScript, so you can get the value of any form element directly from JavaScript.
- **disabled**—When this attribute is added, the element will be displayed; however, it will appear disabled so the user cannot interact with the form element. When you enable and disable elements from JavaScript or jQuery, this value is modified dynamically. The `disabled` attribute is a Boolean, so you do not need to include a value. The following shows an example of how to include the `disabled` attribute:

Click here to view code image

```
<input type="text" disabled />
```

Try it Yourself: Adding Forms to Web Pages

In this example, you add a form with different elements to a web page. The finished example, shown in Listing 3.6, renders the web form shown in Figure 3.10 with several types of form elements. Using the basics you learn in the following example, you will be able to create just about any custom form you need:

1. Create a new HTML document in the lesson03 folder called forms.html (to match the filename on the book's website).
2. Add the appropriate `<html>`, `<head>`, and `<body>` tags.
3. Add a `<form>` element.
4. Inside the `<form>` element, add the following `<fieldset>` to request info. The `<fieldset>` includes a `<legend>` element that provides a caption; two basic `text <input>` tags for name, origin, destination, and data; and then a `<select>` element with four `<option>` children to provide a drop-down list of airlines. Figure 3.10 shows the rendered `<fieldset>` with a box around it and the elements.

Click here to view code image

```
<fieldset>
   <legend>Contact Info:</legend>
   Name: <input type="text" name="name">
     Origin: <input type="text" name="address"><br>
     Destination: <input type="text" name="city">
     Flight Date: <input type="text" name="zip"><br>
 <select name="Airline">
   <option value="DA">Delta</option>
   <option value="PJ">Private Jet</option>
```

```
    <option value="Heli">Helicopter</option>
    <option value="EG">Eagles</option>
  </select>
</fieldset>
```

5. Next, add the radio buttons to select one way or round trip. Add the following `<label>` and `<input>` fields to the form. The `type` attribute of the `<input>` elements is set to `radio`. Also, notice that the `<label>` elements include a `for` attribute that links them to the `radio <input>` elements. Because the `<label>` is linked to the `<input>`, when you click the label, it toggles the radio button.

Click here to view code image

```
<input id="One WayRB" type="radio"
    name="travelPlan" value="One way">
<label for="One WayRB">One Way</label>
<input id="Round TripRB" type="radio"
    name="travelPlan" value="round trip">
<label for="Round TripRB">Round Trip</label><br>
```

6. Add a comments element by adding the following `<textarea>` element. Notice that the `rows` and `cols` attributes define the size, and we include the word "`comments`" initially in the text area rather than adding a label outside. The `<br>` tag at the end causes the following form elements to be on a new line:

Click here to view code image

```
<textarea rows="10" cols="30">comments</textarea><br>
```

7. Provide a check box to enable frequent flier miles by adding the following `<input>` and `<label>`. The type of the `<input>` element is set to `checkbox` so the browser knows to render it as a check box. Also, the `<label>` element uses the `for` attribute so that the user can click the check box or the label to toggle it on or off:

Click here to view code image

```
<input di='newsCB' type="checkbox" name="news" value="news">
  <label for="newsCB">
    I would like to enroll into the frequent flyer miles program
  </label><br>
```

8. Add the following code to include two button elements: a `submit` button and a `reset` button. When the submit button is clicked, the browser submits the form. When the reset button is clicked, the values in the form are reset:

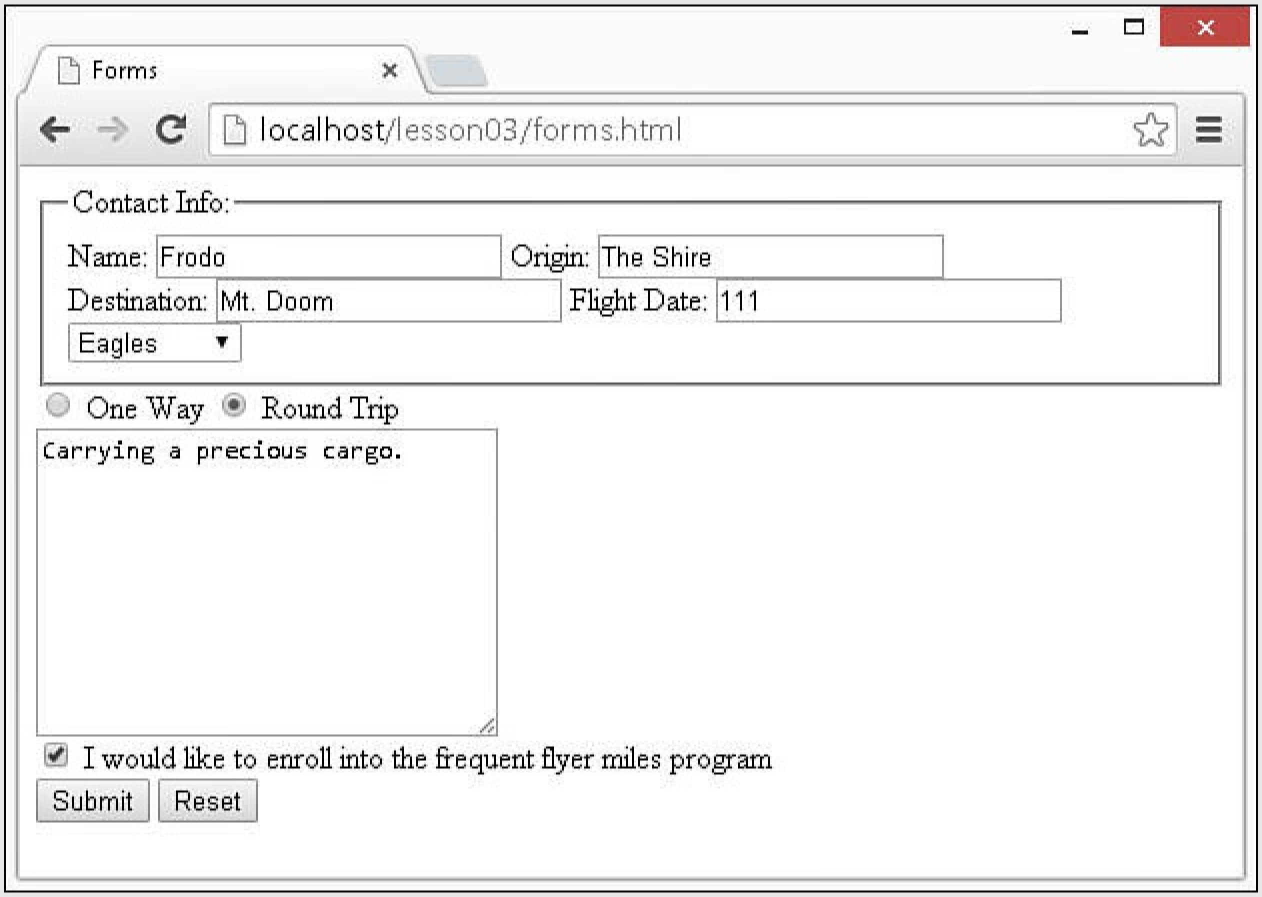
Click here to view code image

```
        <button type="submit" name="SubmitButton"
                value="Submit">Submit</button>
        <button type="reset" name="ResetButton"
                value="Reset">Reset</button>
```

9. Save the document and open the following URL in the web browser to view the rendered results shown in Figure 3.10:

Click here to view code image

```
http://localhost/lesson03/forms.html
```



Contact Info:

Name: Frodo Origin: The Shire

Destination: Mt. Doom Flight Date: 111

Eagles

One Way [✓ Round Trip]

Carrying a precious cargo.

[✓] I would like to enroll into the frequent flyer miles program

Submit Reset

FIGURE 3.10 An HTML form example showing the various form elements created by Listing 3.6.

LISTING 3.6 forms.html HTML Generating a Form with Text, Radio, and Select Inputs

Click here to view code image

```
<html>
  <head>
    <title>Forms</title>
    <meta charset="UTF-8">
  </head>
  <body>
```

```
    <form>
       <fieldset>
          <legend>Contact Info:</legend>
          Name: <input type="text" name="name">
            Origin: <input type="text" name="address"><br>
            Destination: <input type="text" name="city">
            Flight Date: <input type="text" name="zip"><br>
        <select name="Airliner">
          <option value="DA">Delta</option>
          <option value="PJ">Private Jet</option>
          <option value="Heli">Helicopter</option>
          <option value="EG">Eagles</option>
        </select>
      </fieldset>
      <input id="One WayRB" type="radio"
             name="travelPlan" value="One way">
      <label for="One WayRB">One Way</label>
      <input id="Round TripRB" type="radio"
             name="travelPlan" value="round trip">
      <label for="Round TripRB">Round Trip</label><br>
      <textarea rows="10" cols="30">comments</textarea><br>
      <input di='newsCB' type="checkbox" name="news" value="news">
        <label for="newsCB">
          I would like to enroll into the frequent flyer miles program
        </label><br>
      <button type="submit" name="SubmitButton"
              value="Submit">Submit</button>
      <button type="reset" name="ResetButton"
              value="Reset">Reset</button>
    </form>
  </body>
</html>
```

Adding Some Advanced HTML5 Elements

HTML5 adds several more advanced elements that you can use to enhance dynamic web pages. A few of the graphic and media elements are covered in the following sections because a direct relationship exists between those and JavaScript/jQuery programming.

Using HTML5 Graphical Elements

The two main HTML5 graphical elements are `<svg>` and `<canvas>`. These two elements can be added and manipulated via jQuery and JavaScript. This allows you to dynamically change complex graphics in your web pages by drawing directly to the browser window from your scripts.

That means you can dynamically add graphical elements in the browser without needing to load files from the server.

Creating SVG Graphics in HTML5

The `<svg>` element allows you to add scalable vector graphics to your web pages. Scalable vector graphics are composed of a series of lines, arcs, and fills that make up paths. When rendered by the browser, these paths can produce simple to complex graphics. The advantage to using vector graphics is that they retain their crisp edges even when scaled inside the browser. They look sharper onscreen and when printed.

Adding SVG graphics can get extremely complex; the next section gives you an idea of the HTML elements involved so that you can understand attributes and style properties. From JavaScript and jQuery, you will be able to access the properties for various reasons, such as creating and animating graphics in your web page.

Adding Basic Geometric Shapes

First, you'll create some basic geometric shapes. The following code from Listing 3.7 creates a simple series of ellipses and a circle, shown in Figure 3.11. In the `<ellipse>` element, you specify the center of the ellipse as `cx` and `cy` coordinates; then you specify the radius in both the horizontal and vertical directions using the `rx` and `ry` attributes. Notice that you also specify a `stroke` color and width, as well as the `fill` color:

Click here to view code image

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     height="200">
  <ellipse   cx="100" cy="100" rx="80" ry="50"
             stroke="black" stroke-width="2"
             fill="transparent" />
  <ellipse   cx="100" cy="100" rx="50" ry="80"
             stroke="black" stroke-width="2"
             fill="transparent"
             transform="rotate(30, 100, 100)" />
  <ellipse   cx="100" cy="100" rx="80" ry="50"
             stroke="black" stroke-width="2"
             fill="transparent"
             transform="rotate(60, 100, 100)" />
  <ellipse   cx="100" cy="100" rx="8" ry="2"
             stroke="crimson" stroke-width="2"
             fill="crimson"
             transform="rotate(45, 100, 100)" />
  <ellipse   cx="100" cy="100" rx="2" ry="8"
             stroke="crimson" stroke-width="2"
             fill="crimson"
             transform="rotate(45, 100, 100)" />
  <ellipse   cx="100" cy="100" rx="2" ry="8"
             stroke="crimson" stroke-width="2"
             fill="crimson" />
  <ellipse   cx="100" cy="100" rx="8" ry="2"
             stroke="crimson" stroke-width="2"
             fill="crimson" />
```

```
      <circle    cx="100" cy="100" r="6"
            stroke="red" stroke-width="4" fill="DarkRed"/>
    </svg>
```

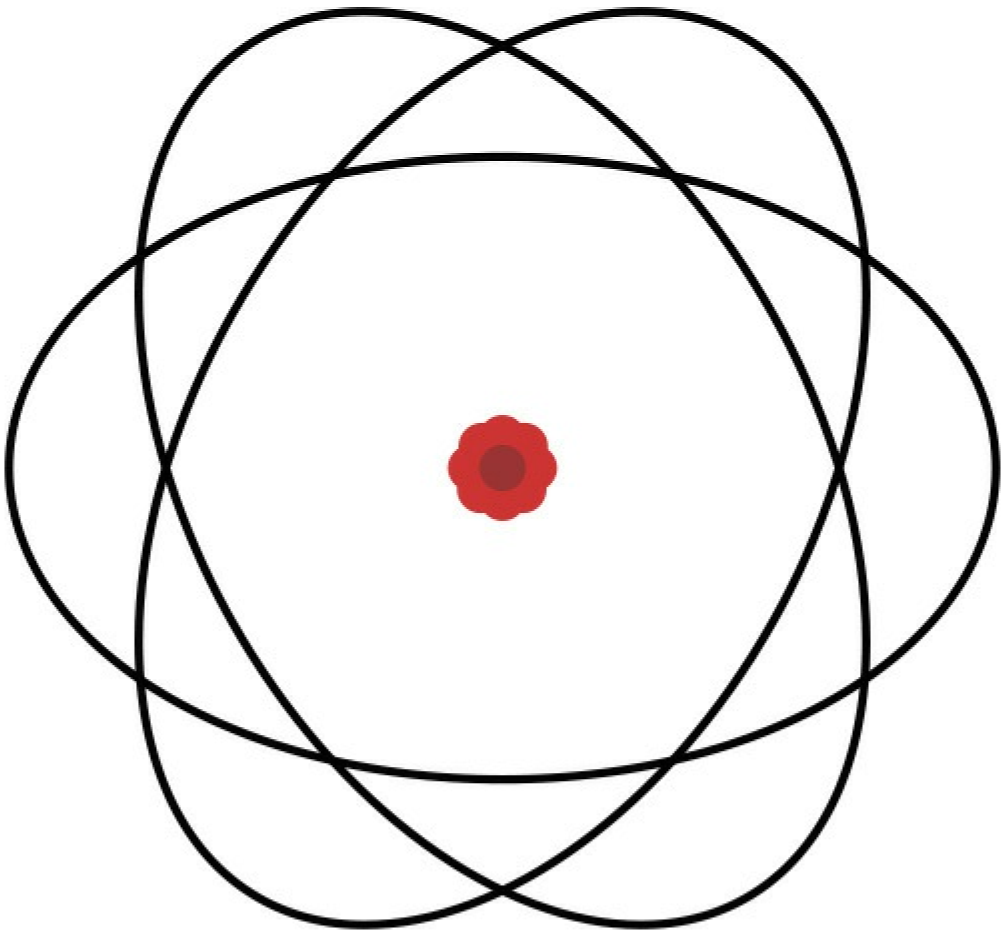


FIGURE 3.11 Using SVG graphics makes it easy to add basic geometric shapes to a web page.

Adding Paths

Another example is using paths to draw some more complex shapes. A path is a collection of lines that are all connected. The lines may be straight, parabolic arcs, or Bezier curves. To create a path, add a `<path>` element to the `<svg>` element and define the `d` attribute. The `d` attribute contains a series of commands that tells the browser how to render the path on the screen. The following is a list of the more common commands:

- **M x,y**—Specifies to move to coordinates `x,y`. Capital `M` specifies absolute coordinates; lowercase `m` specifies relative coordinates.
- **h n**—Specifies to draw a horizontal line `n` pixels. This value can be positive or negative. Negative means left. Capital `H` is absolute and lowercase `h` is relative.
- **v n**—Specifies to draw vertical line `n` pixels. This value can be positive or negative. Negative means up. Capital `V` is absolute and lowercase `v` is relative.

- **l x,y**—Draws a line from the current coordinates to the coordinate `x,y`. You can specify additional sets of coordinates separated by a space to add additional line segments. Capital `L` is absolute, and lowercase `l` is relative.
- **c x1 y1 y2 y2 x y**—Draws a Bezier curve from the current coordinates to `x,y` using `x1,y1` as a control point of the curve for the start and `x2,y2` as control points for the curve's end. Figure 3.12 illustrates how the control points work. Capital `C` is absolute and lowercase `c` is relative.

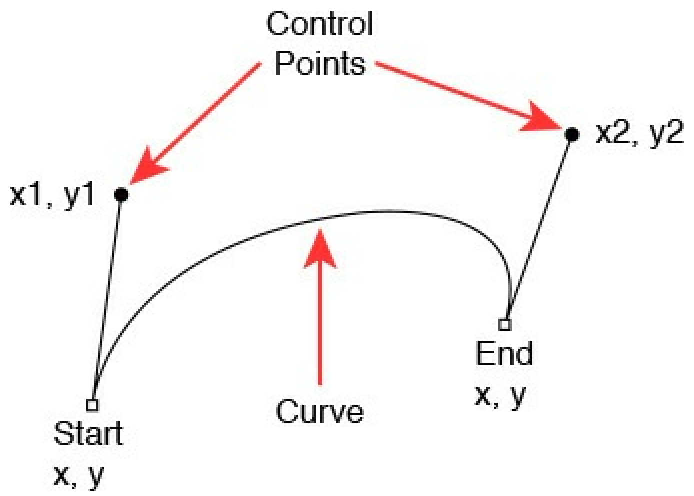


FIGURE 3.12 The control points define the shape of the curve.

- **a rx ry x-axis-rotation large-arc-flag sweep-flag x y**—Draws an arc from the current coordinates to `x,y`. The size and orientation of the ellipse are defined by two radii `rx, ry` as well as the `x-axis-rotation` value, which specifies the angle of the `x` axis of the arc. The `large-arc-flag` and `sweep-flag` are set to `0` or `1` and define which part of the parabolic curve is rendered to the screen. Capital `A` is absolute and lowercase `a` is relative.
- **z**—Close the path, which means that the last coordinate will be connected to the beginning of the path.

The following code in Listing 3.7 illustrates utilizing a couple of paths to create a disconnected pie graph, as shown in Figure 3.13. Notice that the `<path>` data begins with an `M` command to move to a specific absolute coordinate. Then an `h50` command draws a line to the right `50` pixels; next, an `a` command draws an arc with a `50` pixel radius in both directions up `50` pixels and to the right `50` pixels. Finally, the `z` command closes the arc, and you have the first piece of pie. The next `<path>` element draws the main portion of the pie in the same way:

Click here to view code image

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
    height="200">
  <path d="M100,90 h-50 a50,50 0 1,0 50,-50 z"
        fill="none" stroke="purple" stroke-width="1" />
  <path d="M90,80 v-50 a50,50 0 0,0 -50,50 z"
        fill="violet" stroke="none" stroke-width="2" />
</svg>
```

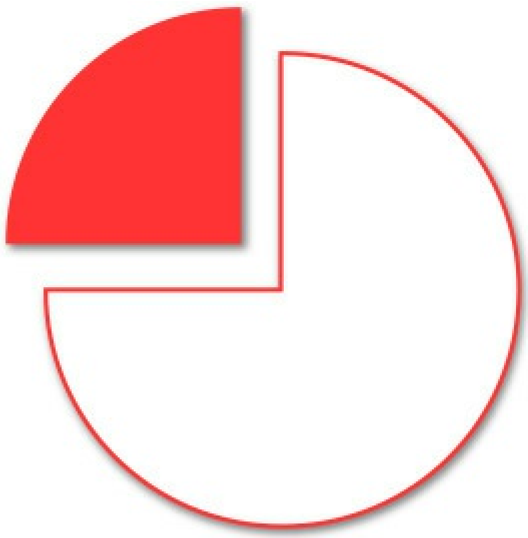


FIGURE 3.13 Using vector paths to create a pie chart.

In addition to using paths to create graphical elements, you can also use paths to create some cool textual effects. For example, the following code adds a path with no fill as an SVG definition in a `<defs>` container. The path draws a full circle using two arc segments. The `id` attribute is set to path1. Next, an SVG `<text>` element is created that defines some text.

Notice that inside the `<text>` element is a `<textPath>` element that references the path1 definition. The result is that the text is drawn on the path, as shown in Figure 3.14. We also added a path element to the `<svg>` tag that draws the hands using a couple of line segments:

Click here to view code image

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     height="350">
  <defs>
    <path id="path1"
          d="M 77,210 a 1,1 0 1,1 200,0 a 1,1 0 1,1 -200,1"/>
  </defs>
  <text x="10" y="10" style="fill:blue;font-size:31px;">
    <textPath xlink:href="#path1">
      Teach Yourself AngularJS JavaScript and jQuery
    </textPath>
  </text>
  <path d="M 175,130 v90 h60" stroke="black"
        stroke-width="5" fill="none"/>
</svg>
```

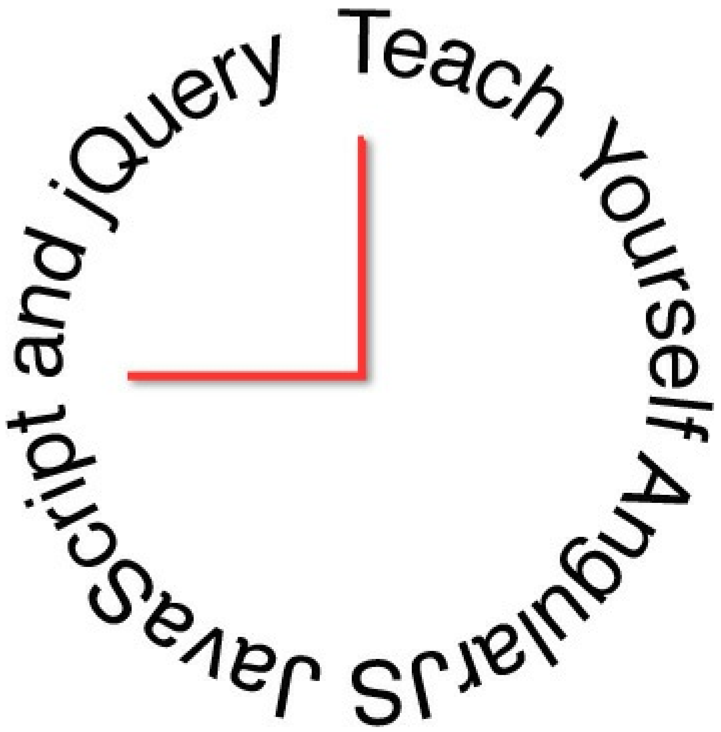


FIGURE 3.14 Using vector paths, you can link text to the path to create some cool visual effects.

The SVG examples can be found in the code/lesson03/html_svg.html file on the website. Using these techniques, you can create a vast amount of graphical shapes.

LISTING 3.7 JavaScript and HTML Code That Uses `<svg>` Elements to Create a Pie Graph and a Text Border Around a Clock

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>html 5 graphics</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
         height="200">
      <ellipse   cx="100" cy="100" rx="80" ry="50"
                 stroke="black" stroke-width="2"
                 fill="transparent" />
      <ellipse   cx="100" cy="100" rx="50" ry="80"
                 stroke="black" stroke-width="2"
                 fill="transparent"
                 transform="rotate(30, 100, 100)" />
      <ellipse   cx="100" cy="100" rx="80" ry="50"
                 stroke="black" stroke-width="2"
                 fill="transparent"
                 transform="rotate(60, 100, 100)" />
      <ellipse   cx="100" cy="100" rx="8" ry="2"
                 stroke="crimson" stroke-width="2"
                 fill="crimson"
                 transform="rotate(45, 100, 100)" />
      <ellipse   cx="100" cy="100" rx="2" ry="8"
                 stroke="crimson" stroke-width="2"
                 fill="crimson"
                 transform="rotate(45, 100, 100)" />
      <ellipse   cx="100" cy="100" rx="2" ry="8"
                 stroke="crimson" stroke-width="2"
                 fill="crimson" />
      <ellipse   cx="100" cy="100" rx="8" ry="2"
                 stroke="crimson" stroke-width="2"
                 fill="crimson" />
      <circle    cx="100" cy="100" r="6"
            stroke="red" stroke-width="4" fill="DarkRed"/>
    </svg>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
         height="200">
      <path d="M100,90 h-50 a50,50 0 1,0 50,-50 z"
            fill="none" stroke="purple" stroke-width="1" />
      <path d="M90,80 v-50 a50,50 0 0,0 -50,50 z"
            fill="violet" stroke="none" stroke-width="2" />
    </svg>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
         height="350">
      <defs>
        <path id="path1"
              d="M 77,210 a 1,1 0 1,1 200,0 a 1,1 0 1,1 -200,1"/>
      </defs>
      <text x="10" y="10" style="fill:blue;font-size:31px;">
        <textPath xlink:href="#path1">
          Teach Yourself AngularJS JavaScript and jQuery
        </textPath>
      </text>
      <path d="M 175,130 v90 h60" stroke="black"
            stroke-width="5" fill="none"/>
    </svg>
  </body>
</html>
```

This section only scratched the surface of SVG. If you would like to learn more about SVG graphics in HTML5, you can check out the docs here: www.w3.org/TR/SVG/Overview.html.

Adding a Canvas for Dynamic Design

The `<canvas>` element also allows you to dynamically add and manipulate image-type graphics to your web pages. The canvas allows you to paint color, lines, text, and images onto an area of the browser screen from JavaScript.

The difference between `<svg>` elements and `<canvas>` elements is that rather than being a path, the graphics are stored as a pixel by pixel map where each pixel is a different color and opaqueness.

Most of the work done with the `<canvas>` element will be done in jQuery and JavaScript scripts. In HTML, all you need to do is add the canvas element with an `id` that you can access from your scripts. To get an idea of the relationship between `<canvas>` elements and JavaScript, consider the code in Listing 3.8.

The canvas element only defines a container that can be drawn on. All the work is done in the `<script>` element. Inside the script, the first two lines get the `myCanvas` element and get a `2d` context object. The context object is the graphical object that provides methods to draw on the canvas. This code sets the line width and color that will be rendered by a `stroke()` call:

Click here to view code image

```
var c=document.getElementById("myCanvas");
var ctx=c.getContext("2d");
ctx.lineWidth="1";
ctx.strokeStyle="blue";
```

The rest of the JavaScript draws three sides of a cube onto the canvas. For each side, the `beginPath()` call starts a new path. Then you use `createLinearGradient()` and `addColorStop()` to create a gradient fill and set the context fill style:

Click here to view code image

```
var grd=ctx.createLinearGradient(100,50,100,5);
grd.addColorStop(0,"blue");
grd.addColorStop(1,"white");
ctx.fillStyle=grd;
```

Next, to build the cube side, begin with a `moveTo()` call to move to a specific coordinate in the canvas and then a series of `lineTo()` to add the lines. At this point, there is still nothing written to the canvas. To write the pixels to the canvas, you can use the `stroke()` call and/or `fill()` calls to draw lines and fill colors on the canvas. The results are shown in Figure 3.15.

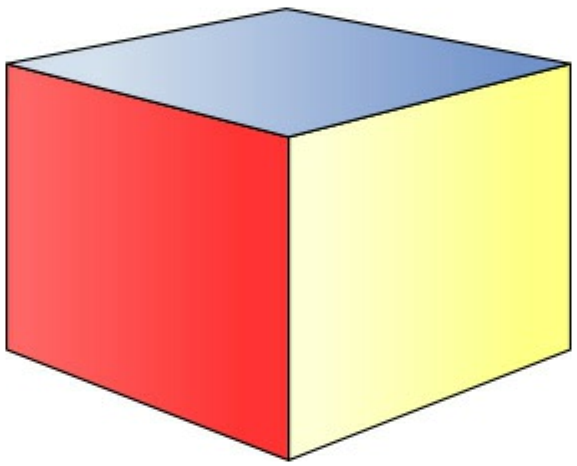


FIGURE 3.15 Using JavaScript to draw pixels on a canvas.

LISTING 3.8 JavaScript and HTML Code That Draws a Cube onto a `<canvas>` Element

[Click here to view code image]

```
<html>
  <head>
    <title>HTML 5 Canvas</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <canvas id="myCanvas" width="300" height="300">
      Sorry Your Browser Doesn't Support HTML5 Canvas
    </canvas>
    <script>
      var c=document.getElementById("myCanvas");
      var ctx=c.getContext("2d");
      ctx.lineWidth="1";
      ctx.strokeStyle="blue";
      //top
      ctx.beginPath();
      var grd=ctx.createLinearGradient(100,50,100,5);
      grd.addColorStop(0,"blue");
      grd.addColorStop(1,"white");
      ctx.fillStyle=grd;
      grd.addColorStop(0,"blue");
      ctx.moveTo(1,25);
      ctx.lineTo(100,5);
      ctx.lineTo(200,25);
      ctx.lineTo(100,50);
      ctx.fill();
      ctx.stroke();
      //left
      ctx.beginPath();
      var grd=ctx.createLinearGradient(75,100,60,25);
      grd.addColorStop(0,"red");
      grd.addColorStop(1,"white");
      ctx.fillStyle=grd;
      ctx.moveTo(1,25);
      ctx.lineTo(100,50);
      ctx.lineTo(100,165);
      ctx.lineTo(1,125);
      ctx.lineTo(1,25);
      ctx.fill();
      ctx.stroke();
      //right
      ctx.beginPath();
      var grd=ctx.createLinearGradient(200,50,125,175);
      grd.addColorStop(0,"yellow");
      grd.addColorStop(1,"white");
      ctx.fillStyle=grd;
      ctx.moveTo(100,50);
      ctx.lineTo(200,25);
      ctx.lineTo(200,125);
      ctx.lineTo(100,165);
      ctx.fill();
      ctx.stroke();
    </script>
  </body>
</html>
```

This section only scratched the surface of `<canvas>` elements. If you would like to learn more about canvas graphics in HTML5, you'll find some good docs and examples here: www.w3schools.com/tags/ref_canvas.asp.

Adding Media Elements

Some additional HTML5 elements that you should be aware of are the `<video>` and `<audio>` tags. These tags allow you to add media elements to web pages in the form of audio and video. Using jQuery and JavaScript, you can reference these elements and manipulate them dynamically. This allows you to change the size, notify the user when the media has loaded, or just control the playback.

The following code shows an example of the `<video>` and `<audio>` tags, and Figures 3.16 and 3.17 show the rendered components:

Click here to view code image

```
<video width="320" height="240" controls>
  <source src="images/movie.mp4" type="video/mp4">
  <source src="images/movie.ogg" type="video/ogg">
  Sorry, your browser does not support the video tag.
</video>
<audio controls>
  <source src="song.mp3" type="audio/mp3">
  Sorry, your browser does not support the audio element.
</audio>
```

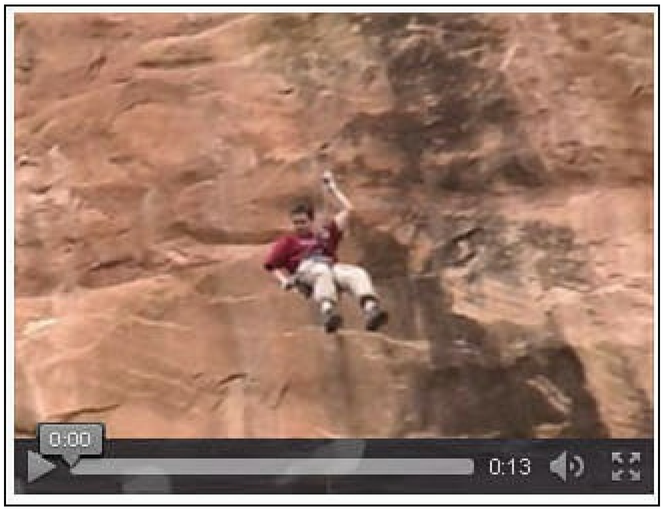


FIGURE 3.16 Rendered `<video>` element allows you to play back a movie.

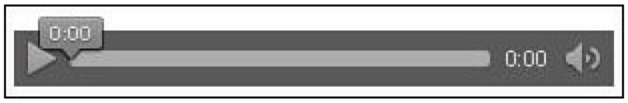


FIGURE 3.17 Rendered `<audio>` element allows you to play back songs or other audio.

Summary

In this lesson, you learned the basics of HTML web page development. You also learned how some of the elements can be dynamically accessed via jQuery and JavaScript so that you can better design interactions into your web pages. In addition, you learned the basics necessary to design your HTML elements to support implementing CSS layouts and styling.

Several Try It Yourself walkthroughs took you step by step through adding styled containers, tables, and web forms. In subsequent lessons, you use the basics learned in this lesson to implement dynamic jQuery and JavaScript.

Q&A

Q. Why is there HTML5 and HTML 4, and not just HTML? Which one should I use?

A. It takes web browsers some time to adopt the standards in the newer versions of HTML. Many of the HTML5 features will not work in older browsers and some newer browsers only support some features but not others. This causes errors on the HTML pages and a very poor user experience. Because of that, HTML 4 is kept as a different standard until HTML5 is fully supported on all web browsers. Which one you select to use depends on who will be viewing your web pages. If the web pages are for everyone on the Internet, you need to continue to support HTML 4 until the browsers fully support HTML5. If the web pages are only for users internal to your company, and you can require a certain level of web browser, you can choose a web browser and version that fully supports HTML5 and run with the new features.

Q. Why even bother creating HTML documents with elements when you can dynamically create them from JavaScript?

A. The best paradigm to use for dynamic web pages is to build the bare bones in HTML, style them in CSS, and then add interactions with JavaScript. The reason you should use HTML documents with the full bare bones is that it is much easier to understand and adjust the structure of the web pages via HTML rather than in JavaScript. Also, if you are localizing your web pages, it is often easier to localize an HTML document in contrast to dynamic JavaScript strings.

Workshop

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

Quiz

1. What HTML element attribute is used to change such things as the background color and size of elements?
2. What is the `<noscript>` element used for?
3. What is the difference between the `<canvas>` and `<svg>` elements?
4. True or false: You cannot have an HTML page link to a location on the same page.
5. What is the difference between a block and inline HTML elements?

Quiz Answers

1. `style`.
2. The `<noscript>` allows you to display a message on the HTML page if JavaScript is not active in the user's web browser.
3. The `<canvas>` element displays images as pixel-based graphics and supports images such as JPEG and PNG; `<svg>` supports vector-based graphics as a series of paths that remain crisp when scaled.
4. False. You can use the `<a href"#` syntax to link to a local anchor on the web page.
5. Block elements take up the full width of the screen so elements after them flow

below, whereas multiple inline elements can be displayed on the same line because they take up only the amount of room necessary for the content within them.

Exercises

1. Extend the example in Listing 3.5 to include some additional national parks.
2. Use the example in Listing 3.8 as a base and change the cube to a sphere by making all the top points in the path go to the same point (100,15). You will need to remove the portion that draws the top of the cube, change the moves, and remove some of the `lineTo()` functions.

Lesson 4. Adding CSS/CSS3 Styles to Allow Dynamic Design and Layout

What You'll Learn in This Lesson:

- Adding CSS to HTML documents
- How to use CSS selectors to apply styles to specific HTML elements
- How to apply color, images, and backgrounds to HTML elements
- Creating cool borders around HTML elements
- Using CSS to define the look and layout of web pages
- Designing CSS to be used by your AngularJS, jQuery, and JavaScript code

One of the most important aspects of dynamic web pages is their capability to dynamically adjust the design and layout of elements as users interact with the page. This lesson focuses on adding CSS styles to your HTML documents. CSS provides a way to easily apply style changes to HTML elements on the web page.

It is important to add an initially good design that can be easily altered in your AngularJS, jQuery, and JavaScript code. With a good design, simple style changes through AngularJS, jQuery, and JavaScript can dramatically alter the appearance and behavior of the HTML components, providing a rich user experience.

The following sections cover CSS syntax and using CSS to modify the look of HTML elements and the layout of the web page.

Adding CSS Styles to the Web Page

You can add CSS styles to web pages in a number of ways: as a separate file, in the HTML `<head>` element, in the HTML `<body>` element, inline inside of a specific HTML element, or even dynamically from AngularJS, jQuery, and JavaScript.

You can apply as many styles in as many ways as you would like. However, you need to keep in mind that the styles are applied in order, with the latest style overriding the values of previous styles. Styles are loaded in the following order:

1. `<link>` or `<style>` elements from the header. The lower elements override the ones above them.
2. `<style>` elements in the web page body. The lower elements override the ones above them.

Caution

If you define different CSS definitions for the same element(s) in two different CSS files or `<style>` sections, the ones that are loaded last will overwrite the previous ones. This is very useful; however, it can be a pain if it is not intended. For example, if you have a selector that changes several properties for `<p>` elements and then a subsequent selector that directly references the ID of a specific `<p>` element, you would likely want to place the general definition before the specific so that you retain the specific property values.

---

3. Styles applied to HTML elements directly via the style attribute of the element.
4. Styles applied dynamically via AngularJS, jQuery, or JavaScript. These are dynamically applied when the JavaScript is executed.

The following sections discuss how to add styles using each of the options and when to use them.

Loading CSS Styles from a File

Typically, the best method of loading CSS styles is from a separate file, which means you create a file with a .css extension and then add it to your website. This provides several advantages:

- You can link several HTML documents to the same CSS file.
- Your HTML files remain much cleaner because there is not a lot of extra CSS code encumbering them.
- It is easy to re-skin your website by changing out the CSS file for another.

To load CSS styles from a separate file, you need to add the .css file to your website and then add a `<link>` element to the HTML `<head>` element. When the web page is loaded, the browser parses the `<link>` element, requests the .css file from the web server, and applies the styles when rendering the HTML elements.

The following is an example of the syntax used in the `<link>` element. Notice that the `rel` attribute is set to `stylesheet`, the `type` to `text/css`, and the `href` points to the location of the .css file on the web server:

Click here to view code image

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

Adding CSS Styles to the Header

You can also add CSS styles directly to a web page in the `<head>` element using the `<style>` tag. All the text inside the `<style>` tag will be treated as CSS, read into the browser, and applied to the elements as they are rendered.

Applying CSS styles in the header does have some advantages, including the following:

- It is easy to apply them to the header to test out the web page without needing to manage an additional .css file.
- You can override global styles loaded from a .css file with some that apply specifically to that page.

The following is an example of a `<head>` element that includes a `<style>` element with CSS. Notice that the `type` attribute is set to `text/css`; this is not required, but it is good practice for a time when other style types are supported by browsers:

Click here to view code image

```
<html>
  <head>
    <meta charset="UTF-8">
    <style type="text/css">
      p{
        background-color:#C0C0C0;
        padding: 3px;
      }
      span{
        font-weight:bold;
      }
    </style>
  </head>
...
```

Using CSS Styles in the HTML Body

You can also use the `<script>` tag to add CSS style settings directly inside the HTML `<body>` element. This is not a good practice, because it can make the web page styles very difficult to change and fix. You should limit your use of this method to times when you want to quickly add a style for testing purposes. After testing, you should always move them to the `<head>` or a separate file.

Defining CSS Styles in HTML Elements

HTML elements provide a `style` attribute that allows you to directly set the CSS style inside the HTML statement. This provides advantages, but one huge disadvantage.

The advantage is that you can override the CSS styles that are applied globally to the website or web page. This makes it possible to customize the look and feel of a specific element without the need to include a special rule in another location.

The disadvantage is that if you do this very much, it makes it really difficult to update the style of your website when you decide on a different web design or branding.

The following line of code shows an example of adding a CSS `style` directly inside of a `<span>` element:

Click here to view code image

```
<span style="background-color:blue; color:white font-weight:bold">Styled
Text
</span>
```

Adding CSS Styles to HTML Elements

You have already seen some brief examples of CSS being applied in previous examples in this book. Now it is time to introduce you to the syntax and properties that you can apply to HTML elements via CSS. This section is by no means comprehensive; however, it will give you an understanding of what can be done to HTML elements via CSS.

As you read through and try out the examples, keep in mind that you will have access to the CSS properties via AngularJS, jQuery, and JavaScript and can therefore set them dynamically as the user interacts with the web page.

Understanding the Basic CSS Syntax

CSS is composed of a set of one or more rules that define values of properties that the web browser uses when rendering the HTML element. Each rule is started by a selector that defines which HTML element(s) to apply the style change to. Then inside the `{}` brackets are specific property settings. The property values are set using the `property:value` syntax. Each property setting is separated by a semicolon.

The following listing shows an example of a simple CSS rule that sets the font style, background color, and width of a `<p>` element:

```
p {
  font-style:italic;
  background-color:#DDDDDD;
  width:250px;
}
```

You should use a separate line for each of the elements to make the file more readable and clean. You can combine multiple elements on the same line as long as they are enclosed in the `{}` brackets. For example:

Click here to view code image

```
p { font-style:italic; background-color:#DDDDDD; width:250px; }
```

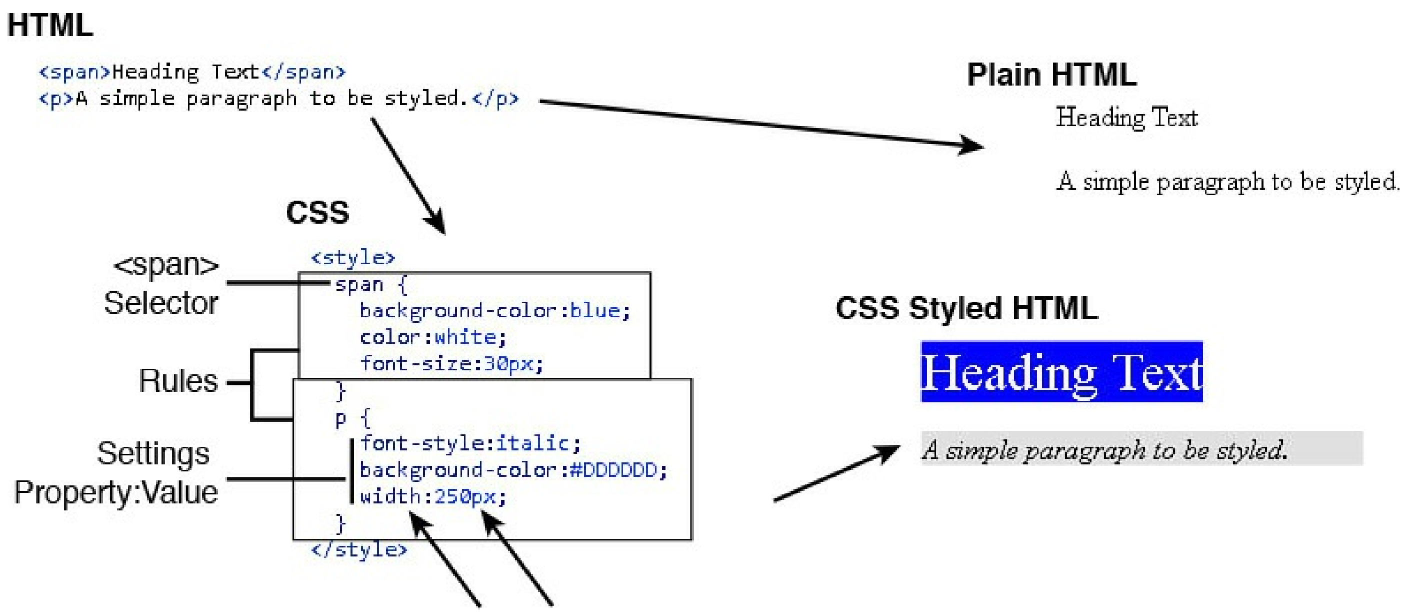
Note

Several of the CSS properties support multiple settings for a single property. This helps keep your CSS files a bit more concise and yet maintains the readability. For example, the following single CSS property setting sets the font to `bold`, `italic`, `12` pixels, and `Times New Roman`. The multiple settings are separated by spaces. Two typefaces are specified, `Times New Roman` and `serif`, so if the browser can't find the first, it will use the second. Notice that because `Times New Roman` includes spaces, it must be encapsulated in double or single quotes. Also notice that the two typefaces are separated by a comma indicating they are still part of the same setting value:

Click here to view code image

```
font:italic bold 12px "Times New Roman",serif;
```

Figure 4.1 shows the basic CSS to apply background and color changes to a `<p>` element and a `<span>` element. Notice that there are two rules—one for each element type. The property values listed in each rule are applied only to the element specified by the selector.



```
<span>Heading Text</span>
<p>A simple paragraph to be styled.</p>
```

```
<style>
  span {
    background-color:blue;
    color:white;
    font-size:30px;
  }
  p {
    font-style:italic;
    background-color:#DDDDDD;
    width:250px;
  }
</style>
```

FIGURE 4.1 CSS rules are composed of a selector, followed by a list of property values to be applied to HTML elements.

Note

Some of the CSS properties are supported only in specific types of browsers. CSS has a naming convention that allows for the browser engine to be prepended onto the property name to identify the browser that the property setting is intended for. This provides a couple of benefits: one is that you can specify different CSS property values for different browsers; the second is that it allows you to set the CSS property only for browsers that support the functionality. The prefixes are `-ms-` for Internet Explorer, `-moz-` for Firefox, `-webkit-` for Safari and Chrome, and `o-` for Opera. The following shows an example of using the prefixes to set the rotate property value:

Click here to view code image

```
    -ms-transform:rotate(5deg);          /* IE 9 */
    -moz-transform:rotate(5deg);          /* Firefox */
    -webkit-transform:rotate(5deg);      /* Safari and Chrome */
    -o-transform:rotate(5deg);          /* Opera */
```

Using CSS Selectors to Style HTML Elements

One of the most important pieces of styling HTML elements is the CSS selector. The CSS selector is used to define which HTML elements the CSS rule applies to. CSS selectors can seem a bit daunting at first; however, when you understand the basic concepts in some examples, the syntax falls into place.

The following series of examples will help you understand selectors.

To apply a CSS rule to all `<div>` elements, use the following (HTML elements are referred to by tag name):

```
div {...
```

To apply a CSS rule to a specific HTML element with the `id` attribute equal to `myDiv`, use the following (in CSS, the `id` attribute is designated using `#`):

```
#myDiv {...
```

To apply a CSS rule to a group of HTML elements all with the `class` attribute equal to `container`, use the following (in CSS, the `id` attribute is designated using `.`):

```
.container {...
```

To apply a CSS rule to `<span>` elements when the mouse is hovering over them, use the following (in CSS, states are designated by `:state`):

```
span:hover {...
```

To apply a CSS rule to `<a>` elements that have the target attribute set to `_blank`, use the following (in CSS, attributes are designated by `[]` brackets):

```
a[target=_blank]
```

You can also chain multiple selectors together using commas. For example, to apply a CSS rule to all `<div>`, `<span>`, `<p>`, and elements with `class="menu"`, you could use the following:

```
div, span, p, .menu {...
```

Also keep in mind that CSS stands for Cascading Style Sheets. You may have different selectors that apply to several groups of objects that overlap. For example, consider the following selectors. The first applies to all `<div>` elements, the second applies only to "`menu`" class elements—some of which are `<div>` elements—and the final applies only to "`menu`" class elements that are currently under the mouse cursor. Property settings in the `.menu` and `.menu:hover` rules will override settings made in the div rule:

```
div {...
.menu {...
.menu:hover{...
```

Table 4.1 provides a list of several types of selectors; it shows an example and describes how the selector works. You can do much more with selectors than what is shown in the preceding examples and in Table 4.1, but these should give you an idea of how we are using selectors in the rest of the exercises in this book.

Selector	Example	Description
`.class`	`.menu`	Selects all elements with `class="menu"`
`#id`	`#myClass`	Selects the element with `id="myClass "`
`*`	`*`	Selects all elements
`element`	`p`	Selects all `<p>` elements
`element,element`	`div, .menu`	Selects all `<div>` elements and all elements with `class="menu"`
`element element`	`div span`	Selects all `<span>` elements inside `<div>` elements
`element> element`	`div>span`	Selects all `<span>` elements where the parent is a `<div>` element
`element+ element`	`div+table`	Selects all `<table>` elements that are placed immediately after `<div>` elements
`element1~ element2`	`h1~ul`	Selects every `<ul>` element that is preceded by a `<h1>` element
`[attribute]`	`[target]`	Selects all elements with a target attribute
`[attribute=value]`	`[value= 1]`	Selects all elements with `value="1 "`
`[attribute~= value]`	`[src~= css]`	Selects all elements with a `src` attribute containing "`css`"

[attribute^= value]	a[src^="https"]	Selects the <a> elements with a src attribute value that begins with "https"
[attribute$= value]	a[src$=".png"]	Selects the <a> elements with a src attribute value ending with ".png"
[attribute*= value]	img[src*= "bkground"]	Selects every <img> element with a src attribute value containing "bkground"
:link	a:link	Selects all unvisited links
:visited	a:visited	Selects all visited links
:active	a:active	Selects the active link
:hover	a:hover	Selects links that the mouse is over
:focus	input:focus	Selects the currently focused input element
:empty	div:empty	Selects every <div> element that has no children
:enabled	input:enabled	Selects the enabled <input> elements
:disabled	input:disabled	Selects the disabled <input> elements
:checked	input:checked	Selects the checked <input> elements

TABLE 4.1 List of Some of the More Commonly Used CSS Selectors

Using CSS Design Properties

Most of the CSS properties are aimed at altering the appearance of the HTML items. Unfortunately, the out-of-the-box HTML elements look pretty bland. However, by adding color, borders, backgrounds, images, and other design properties, you can dramatically change the appearance of your HTML elements with only a little bit of CSS code.

The following sections cover the CSS properties that alter the look of HTML components.

CSS Colors

One of the most frequently used settings applied via CSS is the color property. Most elements have a color property that defines the color the browser uses to render them—for example, a table border, a list bullet, or text characters in a paragraph.

You can use several methods for setting the exact color to support the different backgrounds and situations of people who are defining the color. The color value can be specified via one of the following methods:

- **Name**—The CSS color name, such as `red`, `blue`, `green`, or `yellow`. There are 147 predefined color names, such as `aqua`, `crimson`, and `silver`. You can find a list of color names at www.w3.org/TR/css3-color/#svg-color.

- **Hex**—You can specify a hex number that represents the amount of red, green, and blue to include in the color. The syntax is `#RRGGBB`. A value of `00` represents none of that color, and a value of `FF` represents all of that color. For example, red is `#FF0000`, green is `#00FF00`, and blue-green is `#00FFFF`.
- **RGB**—Similar to Hex, except that you can specify values between 0 and 255. For example, blue-green is `rgb(0, 255, 255)`.
- **RGBA**—Same as RGB; however, you can also specify an alpha parameter that controls the opaqueness, with `0.0` being fully transparent and `1.0` being opaque. For example: `rgba(0, 255, 255, 0.5)` is blue-green that is 50% opaque.
- **HSL**—Enables you to specify the color based on the Hue/Saturation/Lightness color scheme. For example, a color with a hue of `100`, saturation of `45%`, and a lightness of `75%` would be `hsl(100, 45%, 75%)`.
- **HSLA**—Same as HSL, except you can also specify an alpha parameter that controls the opaqueness, with `0.0` being fully transparent and `1.0` being opaque. For example, an 80% transparent color would be `hsla(100, 45%, 75%, .20)`.

For example, the following CSS rules all define the text color of a `<p>` element as blue and are completely interchangeable:

Click here to view code image

```
p {color:blue;}
p {color:#0000FF;}
p {color:rgb(0,0,255);}
p {color:rgba(0,0,255,1.0);}
p {color:hsl(240,100%,50%);}
p {color:hsl(240,100%,50%,1.0);}
```

Try it Yourself: Applying Text Styles via CSS

CSS provides several properties that enable you to define the look of text in HTML elements. The following are some of the more common text properties you might be dealing with:

- **color**—Allows you to set the color and transparency of the text.
- **font**—Allows you to set the various properties of the font used to render the text. The properties that you can set are `font-style`, `font-variant`, `font-weight`, `font-size`, and `font-family`. The font property allows these to be set on a single line or as a series of separate properties. For example, the following lines are equivalent and produce the output shown in Figure 4.2:

Click here to view code image

```
#roman{font: italic bold 30px "Times New Roman", serif}
#roman{
  font-family:"Times New Roman", serif;
  font-style:italic;
  font-weight:bold;
  font-size: 30px;
  }
```

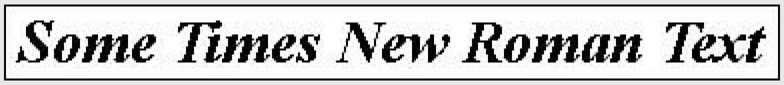


FIGURE 4.2 Changing the font style, weight, size, and face.

Note

Font families are applied in the order specified in the font-family property. The first font found is used. It is a good idea to always provide one of the common fonts, such as serif, as a backup in case the font you specify doesn’t exist on the browser’s system.

- **text-align**—Allows you to set the text alignment to `right`, `left`, or `centered`. For example, the following CSS settings render the output shown in Figure 4.3:

Click here to view code image

```
#align-left{text-align:left}
#align-center{text-align:center}
#align-right{text-align:right}
```

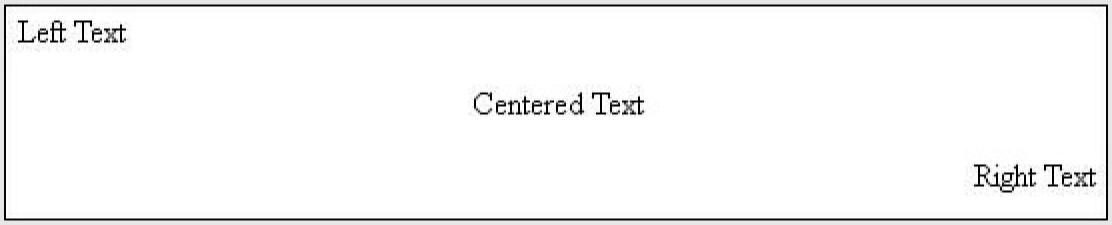


FIGURE 4.3 Changing the alignment repositions the text in the browser.

- **letter-spacing**—Allows you to specify an amount of spacing between the letters in the words. This value can be either positive or negative and can be specified as `px/pt/cm`. Figure 4.4 shows an example of using the following CSS setting to tighten the letter spacing by 1 pixel:

```
#tight{letter-spacing:-1px}
```

FIGURE 4.4 Changing the letter-spacing to 1px tightens the text.

- **word-spacing**—Enables you to specify an amount of spacing between the letters in the words. This value can be either positive or negative and can be specified as `px/pt/cm`.

- **line-height**—Enables you to control the amount of space between lines. You can specify the value as a number that is multiplied by the height of the text, a size amount using a number with an `em/px/pt/cm` suffix, or a % for percentage of the height of the line. Figure 4.5 shows an example of using the following CSS setting to tighten the line spacing to only 50% the normal height:

```
#half-height{line-height:50%}
```

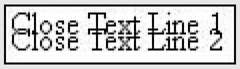


FIGURE 4.5 Changing the line height to 50% tightens the lines of text.

- **text-decoration**—Enables you to add a line below, above, or through the text using the `overline`, `underline`, or `line-through` values. You can also specify `blink` as the value to have the text blink in the browser. The following lines of CSS cause the element to render an underline and strikethrough, as shown in Figure 4.6:

Click here to view code image

```
#underline{text-decoration:underline;}
#strike-through{text-decoration:line-through;}
```

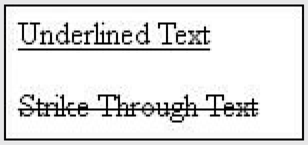


FIGURE 4.6 Changing the text to underline or strikethrough.

- **text-indent**—Enables you to specify an amount to indent the first line of the text either by a specific value, such as 20px, or as a percentage of the width of the element, such as 20%.
- **text-transform**—Enables you to change the capitalization of the text using capitalize to capitalize the first character in each word, uppercase to make the text all caps, or lowercase to remove all capital letters.
- **text-overflow**—Allows you to define what happens when the text is wider than the size allowed by the element. The options are `clip` to just cut off the text, `ellipsis` to add a ... representing the clipped text, or a string that will be added to replace the clipped text. For example, the following line of CSS will clip text if it is too long and append the string "`(more)`" at the end:

Click here to view code image

```
#overflow{text-overflow:" (more)";}
```

Use the following steps to create an HTML document and add CSS styles to text elements. The full versions of the files you are creating are in Listing 4.1 and Listing 4.2:

1. From Eclipse, create a folder named lesson04.
2. Add a subfolder to lesson04 called css.
3. Right-click the lesson04 folder and create a new HTML document named text_styles.html; then add the following basic HTML elements:

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Text Styles</title>
    <meta charset="UTF-8">
...
  </head>
  <body>
...
  </body>
</html>
```

4. Now add the following <p> elements that you will format using CSS. Notice that each has a different id property that you can use in the CSS selector to isolate just that element:

Click here to view code image

```
    <p id="plain">Plain Text</p>
    <p id="indent">Indented Text <br>Indented Line 2</p>
    <p id="blue">Blue Text</p>
    <p id="tight">Tight Text</p>
    <p id="half-height">Close Text Line 1<br>
                     Close Text Line 2</p>
    <p id="align-left">Left Text</p>
    <p id="align-center">Centered Text</p>
    <p id="align-right">Right Text</p>
    <p id="underline">Underlined Text</p>
    <p id="strike-through">Strike Through Text</p>
    <p id="first-cap">capitalize the first letter</p>
    <p id="uppercase">change to upper case</p>
    <p id="blackadder">Some BlackAdder Text</p>
    <p id="roman">Some Times New Roman Text</p>
```

5. Right-click the new css folder and add a new CSS file named text_styles.css.
6. Add the following CSS rules to the new file to define different property settings for the different textual elements created in step 4:

Click here to view code image

```
#blue{color:blue;}
#tight{letter-spacing:-1px}
#half-height{line-height:50%}
#align-left{text-align:left}
#align-center{text-align:center}
#align-right{text-align:right}
#indent{text-indent:50px;}
#underline{text-decoration:underline;}
#strike-through{text-decoration:line-through;}
#first-cap{text-transform:capitalize;}
#uppercase{text-transform:uppercase;}
```

7. Add the following `<link>` element to the header of text_styles.html to link to the new CSS document:

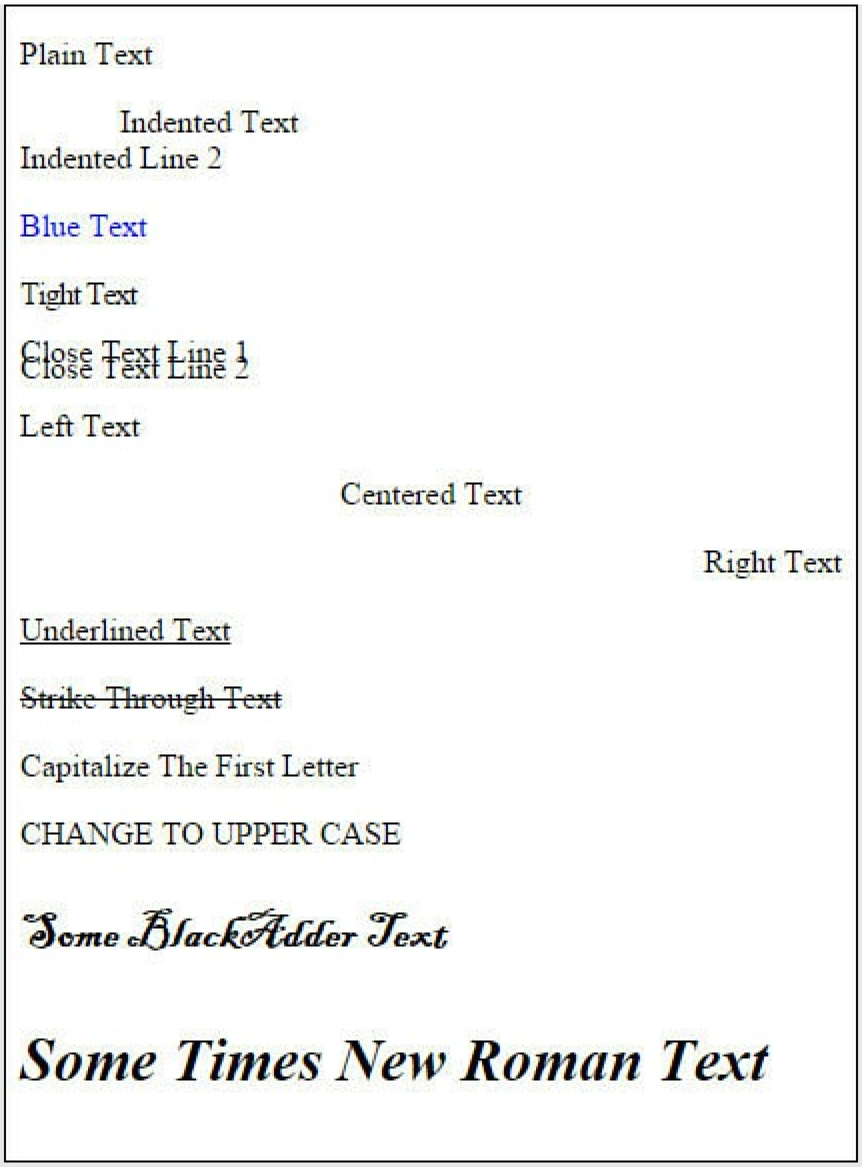
Click here to view code image

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

8. Save both files and open the following location in a browser. You should see output similar to Figure 4.7 with the various looks of each of the paragraphs altered:

Click here to view code image

```
http://localhost/lesson04/text_styles.html
```


Plain Text

Indented Text
Indented Line 2

Blue Text

Tight Text

Close Text Line 1
Close Text Line 2

Left Text

Centered Text

Right Text

Underlined Text

~~Strike Through Text~~

Capitalize The First Letter

CHANGE TO UPPER CASE

Some BlackAdder Text

Some Times New Roman Text

FIGURE 4.7 Applying CSS styles to paragraph elements enables you to completely change their look and behavior.

LISTING 4.1 text_styles.html HTML Code with Several Paragraph Elements to Be Stylized

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Text Styles</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="css/text_styles.css">
  </head>
  <body>
    <p id="plain">Plain Text</p>
    <p id="indent">Indented Text <br>Indented Line 2</p>
    <p id="blue">Blue Text</p>
    <p id="tight">Tight Text</p>
    <p id="half-height">Close Text Line 1<br>
                        Close Text Line 2</p>
    <p id="align-left">Left Text</p>
    <p id="align-center">Centered Text</p>
    <p id="align-right">Right Text</p>
    <p id="underline">Underlined Text</p>
    <p id="strike-through">Strike Through Text</p>
    <p id="first-cap">capitalize the first letter</p>
    <p id="uppercase">change to upper case</p>
    <p id="blackadder">Some BlackAdder Text</p>
    <p id="roman">Some Times New Roman Text</p>
  </body>
</html>
```

LISTING 4.2 text_styles.css CSS Code That Stylizes the Various Textual Elements by Adjusting the Color, Alignments, Adding Lines, and Adjusting the Spacing

[Click here to view code image]

```
#blue{color:blue;}
#tight{letter-spacing:-1px}
#half-height{line-height:50%}
#align-center{text-align:center}
#align-right{text-align:right}
#indent{text-indent:50px;}
#underline{text-decoration:underline;}
#strike-through{text-decoration:line-through;}
#first-cap{text-transform:capitalize;}
#uppercase{text-transform:uppercase;}
#roman{font: italic bold 30px "Times New Roman", serif}
#blackadder{
  font-family:"blackadder itc";
  font-size: 25px;
  font-weight:bold;
  }
```

Try it Yourself: Adding Backgrounds Via CSS

Altering the background properties of elements is an excellent way to completely change the look and feel. You should be aware of several background properties, such as the following:

- **background-attachment**—Can be set to scroll or fixed, which will scroll the background with the rest of the page or leave it fixed.

- **background-color**—Enables you to define the color of the background using the normal color commands; for example:

Click here to view code image

```
#top-left {background-color:rgb(255,255,0);}
```

- **background-size**—Enables you to specify the size of the background of the element. The values can be a width/height set in pixels or percentages, cover which will scale the background to fill the area of the element, or contain which will scale the image so that the entire image fits in the element. Keep in mind that with cover, some parts of the image may not be displayed in the element if the dimension ratios are different. Some examples that follow set the size to 200 pixels wide by 100 pixels high, 20% the width of the element by 200% the height, and contained fully within the size of the element:

Click here to view code image

```
#top-left {background-size:200px 100px;}
#top-left {background-position:20% 100%;}
#top-left {background-position:contain;}
```

- **background-position**—This sets the location the background is placed in the element, such as top left, bottom center, and so on. You can also specify a position coordinate of width and height in pixels or percentages for the top-left corner of the background image. For example, the following code places the background image in the middle, 10% to the right 50% down, and 100px to the right 20px down:

Click here to view code image

```
#top-left {background-position:center center;}
#top-left {background-position:10% 50%;}
#top-left {background-position:100px 20px;}
```

- **background-image (url)**—Enables you to specify the `url` to an image that will be applied as a background. For example:

Click here to view code image

```
#rocs{background-image:url("images/rocks.png");}
```

- **background-image (gradient)**—In CSS3, you can also specify a gradient that will fill the background with transitions to different colors. When specifying the gradient, you can set the position, size, and colors. Each of the browsers has its own specific property name. This is hard to describe, so a few examples follow.

Following is an example of CSS that defines a horizontal linear gradient. Figure 4.8 shows how the gradient is applied. Notice that the position is set to top. It starts with the color blue at the top, goes to white at 70%, and then to green at 100%, or the bottom:

Click here to view code image

```
background-image: -moz-linear-gradient(top, blue 0%, white 70%, green
100%);
```

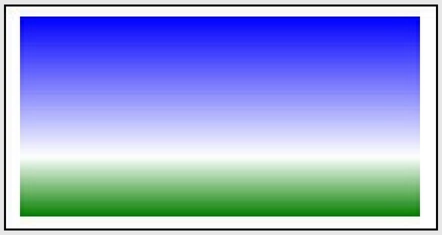


FIGURE 4.8 Creating a linear background image gradient.

- The following line of CSS defines the radial gradient shown in Figure 4.9. The center of the radiant is set to 50% from the left and 50% from the top; `circle` ensures that it isn't skewed into an ellipse, and `contain` scales it down to the right size. The color starts at white at the center, moves to blue at 75%, and then back to transparent white on the outside, using the `rgba()` color setting:

Click here to view code image

```
background-image: -moz-radial-gradient(50% 50%, circle
contain,white,blue
75%,rgba(255,255,255,0));
```

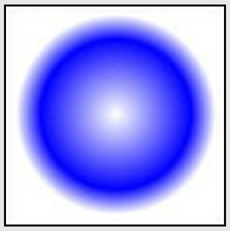


FIGURE 4.9 Creating a radial background image gradient.

- **background-repeat**—You can use this option to repeat the background image multiple times. The values are `repeat` to repeat in all directions, `repeat-y` to repeat in the vertical direction, and `repeat-x` to repeat in the horizontal direction. You can also specify `no-repeat` to display the image only once. Repeating images are used for borders and as a design element for bars and buttons. Figure 4.10 shows the results for the different `background-repeat` values in the following code:

Click here to view code image

```
body{
  background-size:50px 50px;
  margin:5px;
  background-image: -moz-radial-gradient(20px 20px, circle
contain,white,blue 75%,white);
  background-repeat:repeat;
}
```

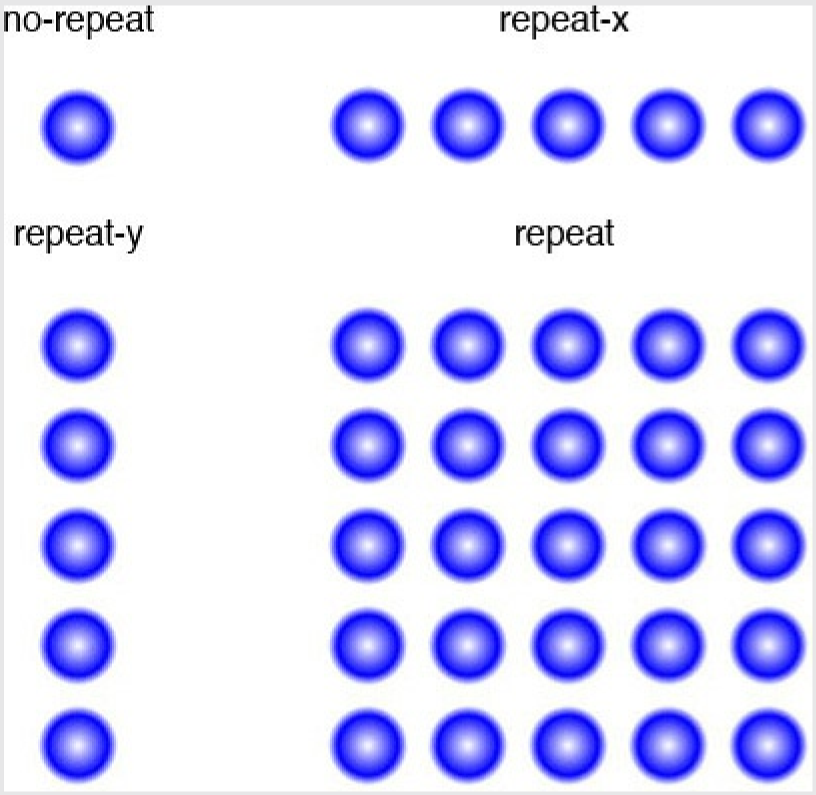


FIGURE 4.10 Using repeating images in an image.

Use the following steps to create an HTML document and add CSS styles to text elements. These are the basics with a few `<div>` and `<span>` elements. The full versions of the files you will be creating are in Listing 4.3:

1. Create a new HTML document named backgrounds.html in the lesson04 folder in Eclipse.
2. Add the following basic HTML elements:

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Backgrounds</title>
    <meta charset="UTF-8">
    <style>
...
```

```
  </head>
  <body>
    <div id="heading">jQuery Rules!</div>
    <div id="content">
      <div id="menu">
        <span>Home</span>
        <span>Info</span>
        <span>Examples</span>
      </div>
      <p>Page Content</p>
    </div>
  </body>
</html>
```

3. Save the file and open it in your browser using the `localhost` address. Notice that the text is very plain. All that is about to change with just a little bit of CSS.

4. Create a 200-pixel wide image named /images/rules.png or download the rules.png file from the book's website and place it there. This image will be a repeating image on the left side.

5. Add the following CSS rule to the `<style>` element in the header of the HTML file. This adds the image you just copied into the images folder to the web page. It repeats the image vertically:

Click here to view code image

```
      body {
        margin:0px;
        background-image:url("/images/ruler.png");
        background-repeat:repeat-y;
      }
```

6. Then add the following rule to style the text for the `heading` element and add a horizontal linear gradient as the background:

Click here to view code image

```
      #heading {
        background-image: -moz-linear-gradient(left, #294551 0%, #AACCFF 100%);
        background-image: -webkit-linear-gradient(left, #294551 0%, #AACCFF 100%);
        background-image: -ms-linear-gradient(left, #294551 0%, #AACCFF 100%);
        height:200px;
        font:150px bold;
        text-align: center;
        color:rgba(255,255,255,.4);
      }
```

7. The vertical repeating image we added to the body is 200 pixels wide, so change the left margin for the content container to 200px using the following rule so that the content will be to the right of the image:

Click here to view code image

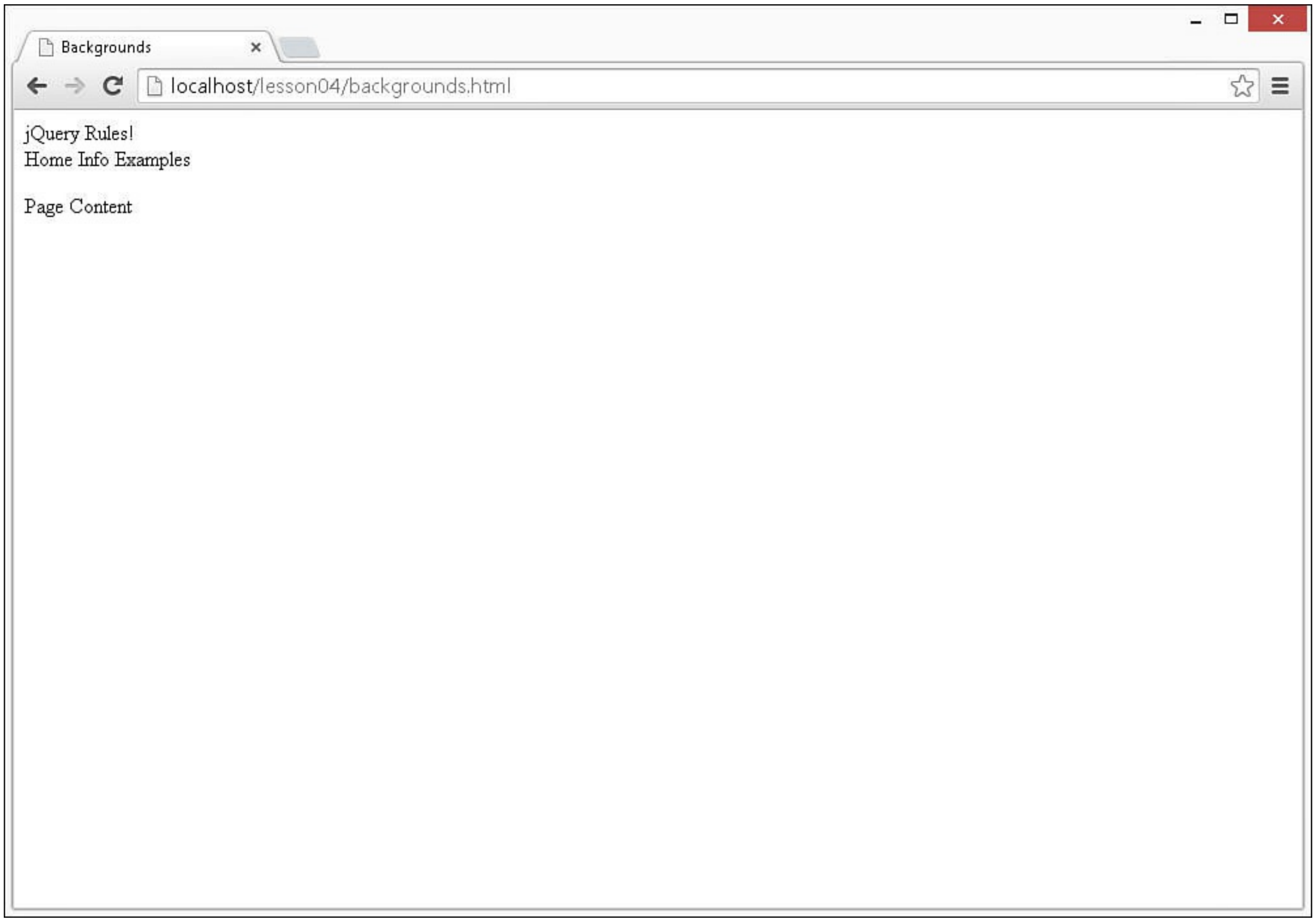
```
#menu{
  padding:2px;
  background-color:#294551;
}
```

8. Style the menu by changing the background color to `#555555` and adding a vertically linear gradient to the `<span>` elements that make up the options:

Click here to view code image

```
#menu{
  padding:3px;
  background-color:#555555;
}
span {
  padding:3px;
  background-image: -moz-linear-gradient(top, #294551 0%, #AACCFF 85%, #0022ff 100%);
  background-image: -webkit-linear-gradient(top, #294551 0%, #AACCFF 85%, #294551 100%);
  background-image: -ms-linear-gradient(top, #294551 0%, #AACCFF 85%, #294551 100%);
  font:20px bold;
  color:white;
}
```

9. Save the file and refresh the web page in your browser, as shown in Figure 4.11. Notice the difference that adding some background elements can have to some otherwise very plain HTML.



jQuery Rules!
Home Info Examples

Page Content

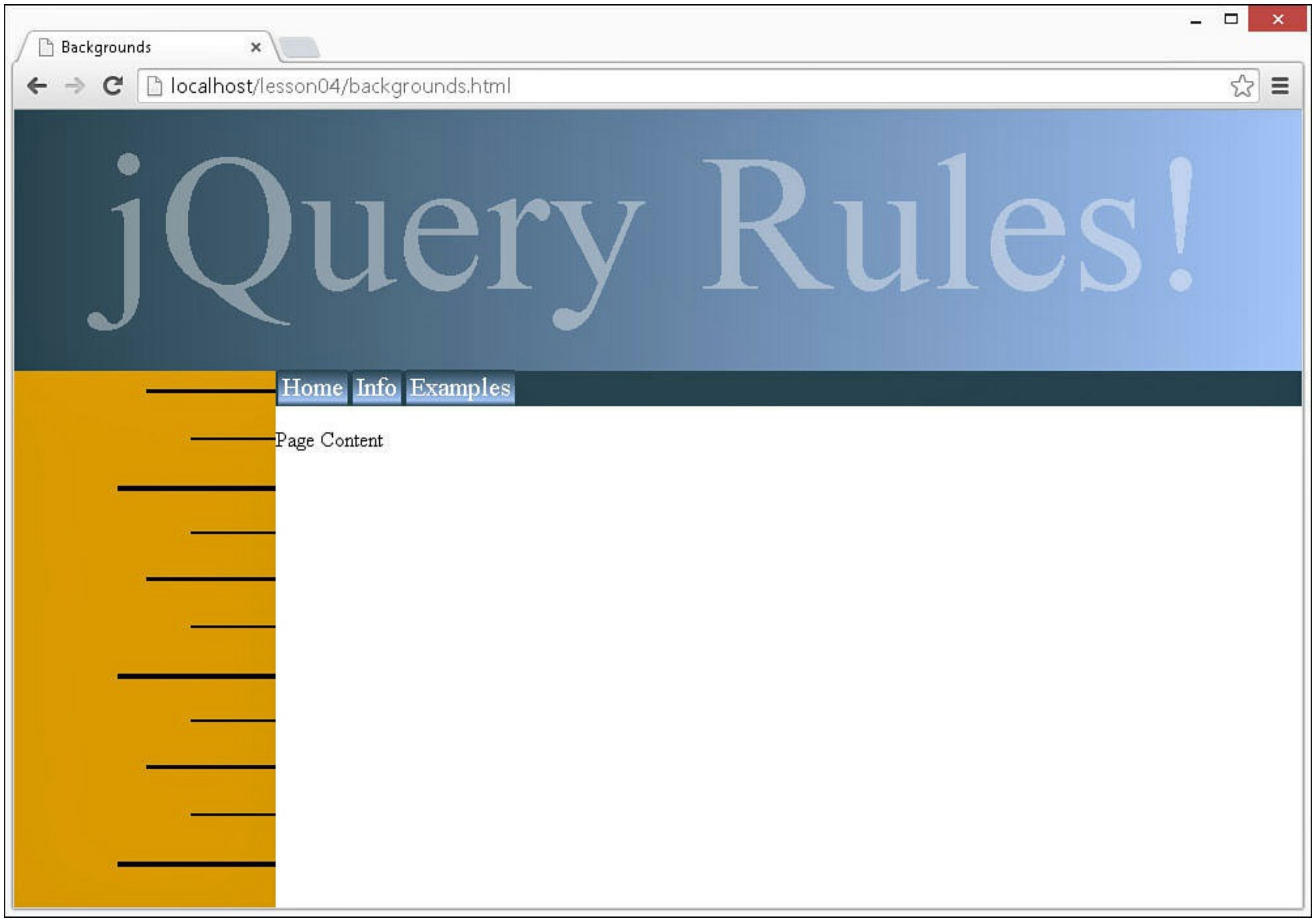


FIGURE 4.11 Adding CSS background styles greatly improves the appearance of elements.

LISTING 4.3 backgrounds.html HTML and CSS Code That Add Different Types of Background Styles to Elements

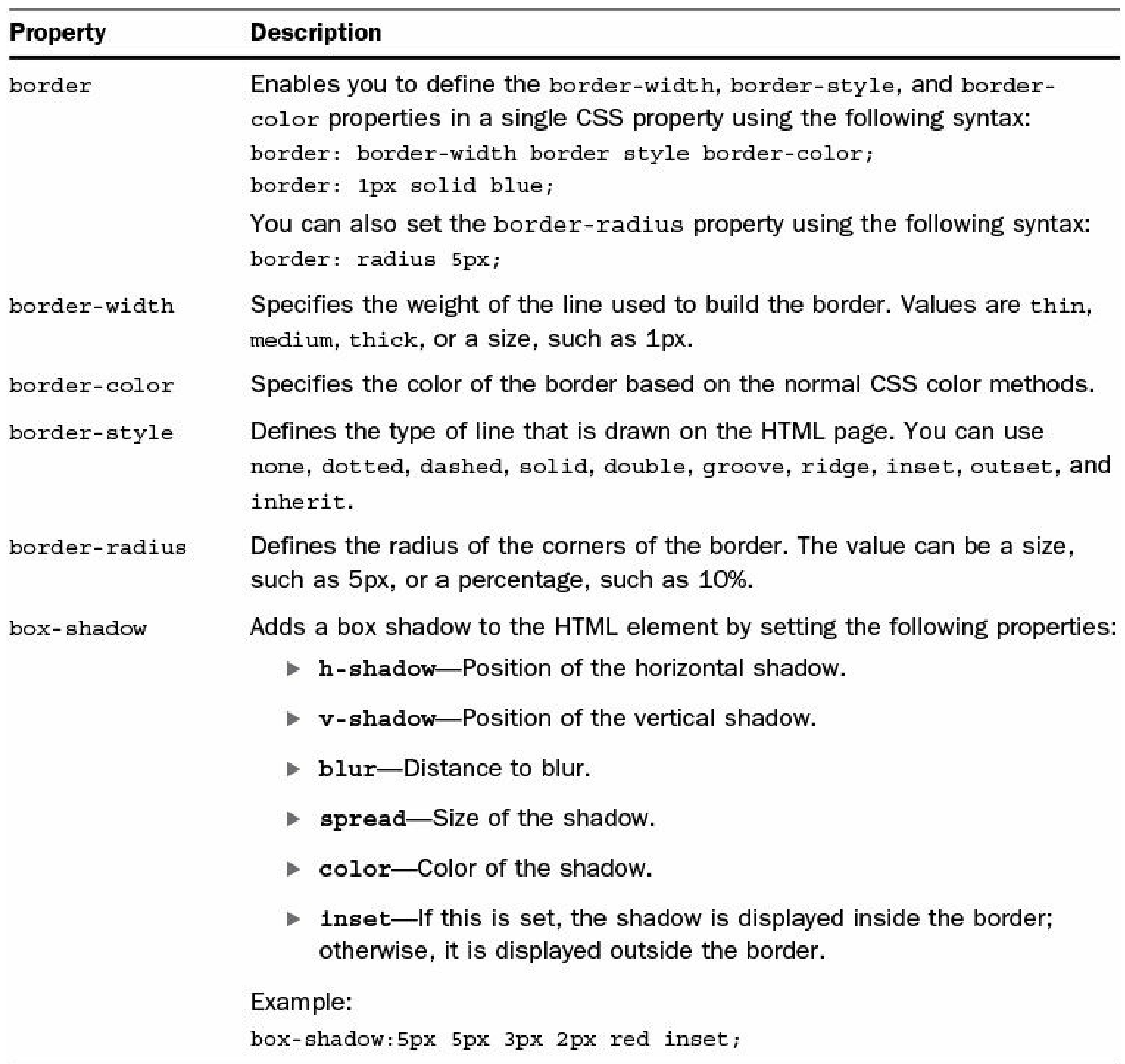
[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Backgrounds</title>
    <meta charset="UTF-8">
    <style>
      body {
        margin:0px;
        background-image:url("/images/ruler.png");
        background-repeat:repeat-y;
      }
      #heading {
        background-image: -moz-linear-gradient(left, #294551 0%,
#AACCFF 100%);
        background-image: -webkit-linear-gradient(left, #294551 0%,
#AACCFF 100%);
        background-image: -ms-linear-gradient(left, #294551 0%, #AACCFF 100%);
        height:200px;
        font:150px bold;
        text-align: center;
        color:rgba(255,255,255,.4);
      }
      #content {
        margin-left:200px;
      }
      #menu{
        padding:2px;
        background-color:#294551;
      }
      span {
        padding:3px;
        background-image: -moz-linear-gradient(top, #294551 0%, #AACCFF 85%, #0022ff 100%);
        background-image: -webkit-linear-gradient(top, #294551 0%, #AACCFF 85%, #294551 100%);
        background-image: -ms-linear-gradient(top, #294551 0%, #AACCFF 85%, #294551 100%);
        font:20px bold;
        color:white;
      }
    </style>
  </head>
  <body>
    <div id="heading">jQuery Rules!</div>
    <div id="content">
      <div id="menu">
        <span>Home</span>
        <span>Info</span>
        <span>Examples</span>
      </div>
      <p>Page Content</p>
    </div>
  </body>
</html>
```

Try it Yourself: Adding Borders to HTML Elements

Another CSS attribute that can be modified to completely change the look and feel of an HTML component is the border attribute. All the container elements, such as `<div>`, `<span>`, and `<p>`, enable you to define a border style through CSS.

Table 4.2 describes the some of the CSS properties that can be modified in CSS to define various styles of borders.



Property	Description

`border`	Enables you to define the `border-width`, `border-style`, and `border-color` properties in a single CSS property using the following syntax:
	`border: border-width border style border-color;`
	`border: 1px solid blue;`
	You can also set the `border-radius` property using the following syntax:
	`border: radius 5px;`

`border-width`	Specifies the weight of the line used to build the border. Values are `thin`, `medium`, `thick`, or a size, such as 1px.
`border-color`	Specifies the color of the border based on the normal CSS color methods.
`border-style`	Defines the type of line that is drawn on the HTML page. You can use `none`, `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, `outset`, and `inherit`.
`border-radius`	Defines the radius of the corners of the border. The value can be a size, such as 5px, or a percentage, such as 10%.
`box-shadow`	Adds a box shadow to the HTML element by setting the following properties: ▶ **`h-shadow`**—Position of the horizontal shadow. ▶ **`v-shadow`**—Position of the vertical shadow. ▶ **`blur`**—Distance to blur. ▶ **`spread`**—Size of the shadow. ▶ **`color`**—Color of the shadow. ▶ **`inset`**—If this is set, the shadow is displayed inside the border; otherwise, it is displayed outside the border. Example: `box-shadow:5px 5px 3px 2px red inset;`

TABLE 4.2 CSS Properties That Define Border Styles

You can specify different values of the `border-width`, `border-color`, and `border-style` attributes on a single property setting using the following syntax:

Click here to view code image

```
property: top right bottom left;
border-width: 1px 2px 1px 2px;
border-color: red blue red blue;
```

You can also set individual border component properties using the location properties, such as `border-top` or `border-top-width`. For example, the following CSS will style only the top and left components of the border:

```
border-top: 1px solid black;
border-left-color: blue;
border-left-style: solid;
border-left-width: 2px;
```

The following steps take you through the process of adding several types of borders to HTML `<div>` elements by creating the code in Listing 4.4 and Listing 4.5:

1. Create a new HTML document named borders.html in the lesson04 folder in Eclipse.
2. Add the code shown in Listing 4.4. The code links to an external style sheet named css/borders.css and then defines a series of `<div>` elements.
3. Create a new CSS document named borders.css in the lesson04/css folder in Eclipse.
4. Add the following line to the new CSS file to set basic spacing and size for the `<div>` elements:

Click here to view code image

```
01 div { width:200px; margin:5px; padding:2px; text-align:center}
```

5. Add the lines of code 2–8 from Listing 4.5 to define several border styles that apply to the different `<div>` elements.
6. Add the following CSS rules to create rounded borders on two of the `<div>` elements:

Click here to view code image

```
09 #round { border: 1px solid; border-radius:5px;}
10 #veryround { border: 1px solid; border-radius:50%;}
```

7. Add lines 11–20 from Listing 4.5. These lines of code create shadows on the `<div>` element. The first is an outset shadow and the second included the inset value that defines it as an inner shadow.
8. Add the following lines of code that create a mixed border. This is included to show you that you have a great deal of control on each side of the border. The top is `solid`, with a radius on the right side; the left is `dotted`, and the bottom is a `ridge` with a radius on the right side:

Click here to view code image

```
21 #mixed {
22   border-top: 1px solid;
23   border-top-left-radius:5px;
24   border-left: 1px dotted;
border-bottom: 5px ridge;
border-bottom-right-radius: 10px;
}
```

9. Add the following code that uses only the border with a large radius and a background color to create a good-looking button:

Click here to view code image

```
#button {
  width:150px;
  background-color: #2233FF;
  color: white;
  border:5px outset blue;
  border-radius:50%;
}
```

10. Save the files and open the HTML file in a web browser. You should see results similar to what is shown in Figure 4.12. Notice the different styles of borders applied with only a few CSS styling statements.

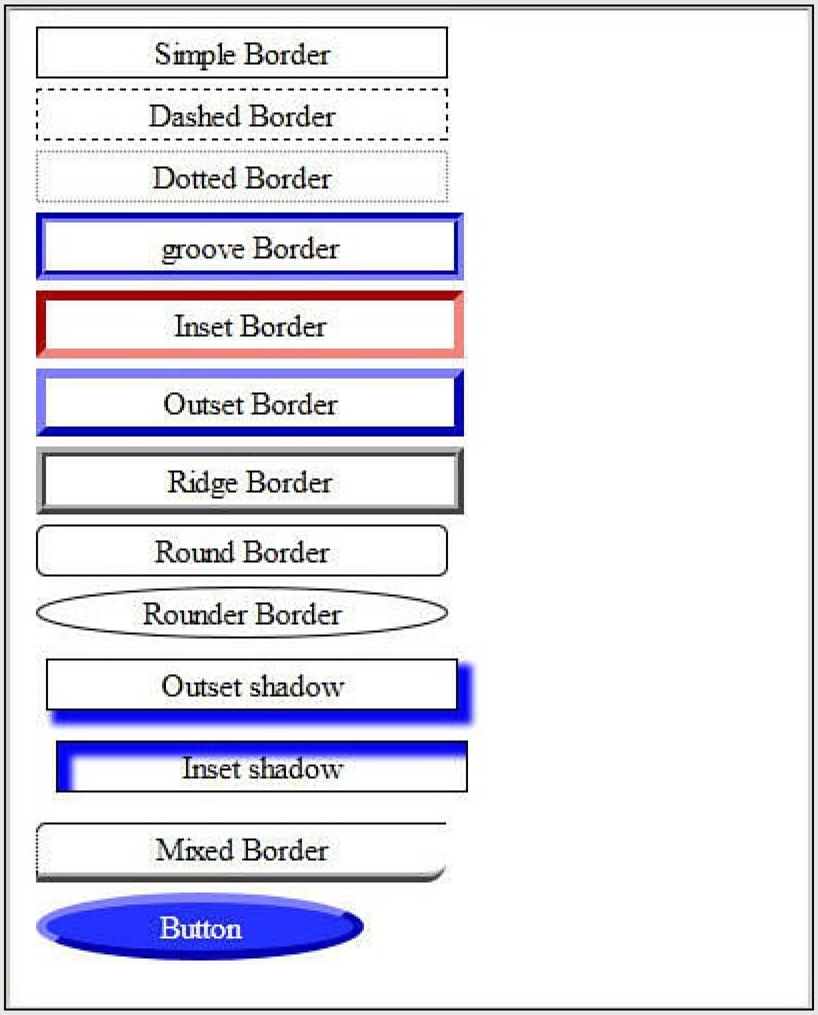


FIGURE 4.12 Adding CSS border elements can apply a variety of borders to HTML elements.

LISTING 4.4 HTML That Creates a Series of `<div>` Elements That Are Styled by Listing 4.5

Click here to view code image

```
<html>
  <head>
    <title>Borders</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="css/borders.css">
  </head>
  <body>
    <div id="simple">Simple Border</div>
    <div id="dashed">Dashed Border</div>
    <div id="dotted">Dotted Border</div>
    <div id="groove">groove Border</div>
    <div id="inset">Inset Border</div>
    <div id="outset">Outset Border</div>
    <div id="ridge">Ridge Border</div>
    <div id="round">Round Border</div>
    <div id="veryround">Rounder Border</div>
    <div id="shadow">Outset shadow</div>
    <div id="ishadow">Inset shadow</div>
    <div id="mixed">Mixed Border</div>
    <div id="button">Button</div>
  </body>
</html>
```

LISTING 4.5 CSS Rules That Define Several Border Styles

Click here to view code image

```
div { width:200px; margin:5px; padding:2px; text-align:center}
#simple { border:1px solid black; }
#dashed { border:1px dashed black; }
#dotted { border:1px dotted gray; }
#groove { border:5px groove blue; }
#inset { border:5px inset red; }
#outset { border:5px outset blue; }
#ridge { border:5px ridge black; }
#round { border: 1px solid; border-radius:5px;}
#veryround { border: 1px solid; border-radius:50%;}
#shadow {
  margin:10px;
  border:1px solid black;
  box-shadow: 5px 5px 3px 2px blue;
}
#ishadow {
  margin:15px;
  border:1px solid black;
  box-shadow: 5px 5px 3px 2px blue inset;
}
#mixed {
  border-top: 1px solid;
  border-top-left-radius:5px;
  border-left: 1px dotted;
  border-bottom: 5px ridge;
  border-bottom-right-radius: 10px;
}
#button {
  width:150px;
  background-color: #2233FF;
  color: white;
  border:5px outset blue;
  border-radius:50%;
}
```

Cursor

An important part of browser interaction with users is the mouse cursor. You can let the user know the purpose or behavior of an HTML element just by changing the way the cursor looks when the mouse is over it.

The `cursor` CSS property enables you to define what cursor should be used when the mouse is over a specific element. The following shows an example of setting the `cursor` to a pointer:

```
div { cursor:pointer; }
```

You can choose from several `cursor` values. Figure 4.13 shows an example of some of the `cursor` values that are available via the `cursor` CSS property.

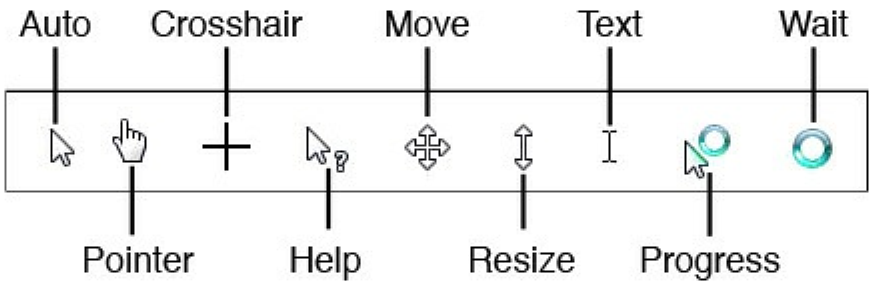


FIGURE 4.13 Examples of some of the cursor types available via the cursor CSS property.

Opacity

The `opacity` CSS property enables you to set the transparency of an HTML element as well as all elements contained within. The `opacity` is specified as a decimal value between 0 and 1, where 0 is transparent and 1 is opaque.

Changing the `opacity` is a great way to let users know that an HTML element is inactive. For example, the following CSS and HTML define two button definitions. The definition for `button2` includes an opacity of .4 to make it partially transparent. The results are shown in Figure 4.14.

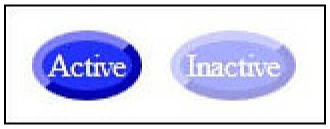


FIGURE 4.14 Two `<span>` elements styled as buttons, with one having an `opacity` set to .4.

CSS rules:

Click here to view code image

```
#button, #button2 {
  margin: 5px; padding:3px;
  text-align: center;
  background-color: #2233FF;
  color: white;
  border:5px outset blue;
  border-radius:50%;
}
#button2 { opacity:.4; }
```

HTML elements:

Click here to view code image

```
<span id="button">Active</span>
<span id="button2">Inactive</span>
```

Visibility

Another useful CSS property is `visibility`. You can set `visibility` to `hidden`, and the HTML element and all elements contained will not be rendered to the browser. Then setting it to `visible` again will display it in the browser window. This capability is very useful in jQuery and JavaScript to enable you to show and hide page elements dynamically.

Applying CSS Layout Properties

One of the biggest challenges with creating HTML web pages is that HTML does not lend itself to creating attractive-looking layouts very easily. That is why many of the CSS properties are aimed at altering the size, position, and relation to other HTML elements.

Using the CSS Layout properties, you will be able to position items correctly on your web page, and with JavaScript, jQuery, and AngularJS you can alter those positions efficiently to provide more of an application experience for your users, rather than a simple point-and-click web page. The following sections cover the CSS properties that alter the layout of HTML components.

Understanding the Box Model

HTML elements are rendered to the browser window using the box model. In the box model, the content of the HTML element is made up of the following four parts shown in Figure 4.15:

- **Content**—The HTML component content itself
- **Padding**—Space between the component and the border
- **Border**—Border surrounding the HTML element
- **Margin**—Space between the HTML element and other elements around it

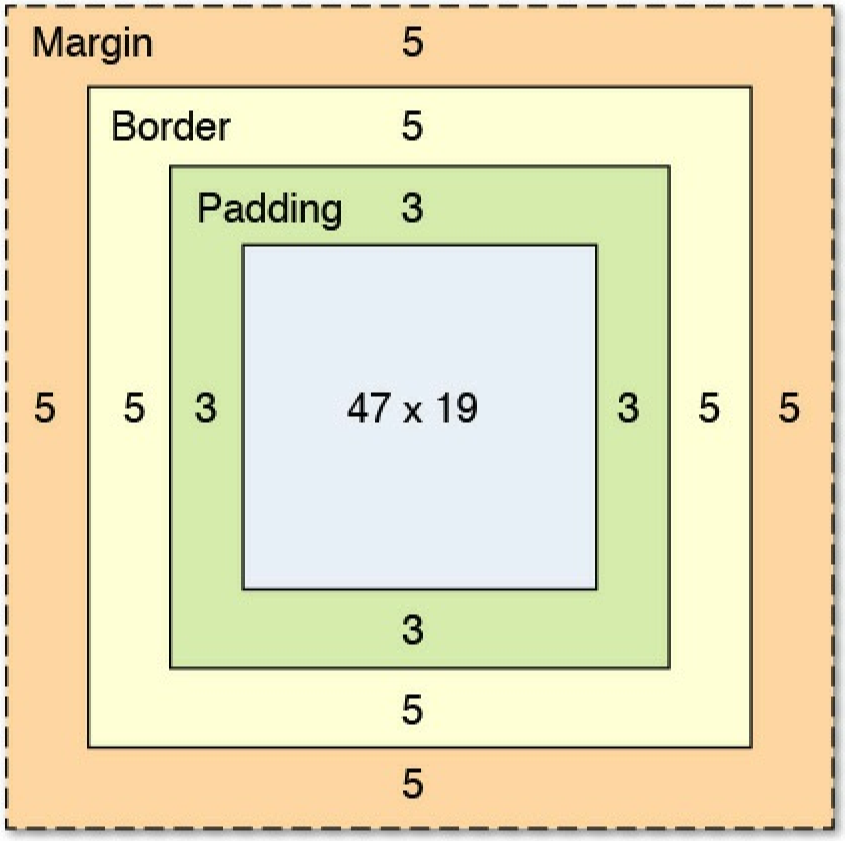


FIGURE 4.15 CSS Box Model includes four parts: content, padding, border, and margin.

The amount of space that each of these takes in the web browser can be set using CSS.

Setting the Content Size

The content is set by specifying a size value of the `height` and `width` CSS properties. The size value can be specified as `px`, `cm`, `auto`, `%`, and so on. You can also specify a maximum and minimum height and width using `max-height`, `max-width`, `min-height`, and `min-width` properties.

The following shows an example of setting the height and width properties of `<p>` element:

```
p {height:100px; width: 50px;}
```

Adding Padding Around HTML Content

Padding is an important part of element layout, especially if you are using borders. Padding keeps the content inside the HTML element from touching the border.

Padding can be added to all sides of the HTML element by specifying a `padding` value, such as the following:

```
padding:3px;
```

You can also specify the padding for each side of the element, for example:

```
padding:1px 2px 1px 2px;
```

Adding Margins Around HTML Elements

Margins work similar to padding, except they are outside the border of the element and keep other HTML elements away from touching the border. A margin can be added to all sides of the HTML element by specifying a `margin` value, such as the following:

```
margin:3px;
```

You can also specify the `margin` for each side of the element, for example:

```
margin:1px 2px 1px 2px;
```

The `margin` property also supports the value of `auto`. Using `auto` splits the `margin` values equally, based on the size constraints that are placed in the HTML element.

Modifying HTML Element Flow

Unless you specify otherwise, HTML elements flow to the top-left corner of the browser window.

As you learned in the previous lesson, some HTML elements are block elements and some are inline elements. The block elements take up the full width, and so items after them flow down to the next line. Inline elements take up only the amount necessary for the content.

Using the `display` CSS property, you can set elements to be `block` or `inline`. Another value of the display attribute worthy of mention is `inline-block`. `Inline-block` allows the element to be displayed inline; however, unlike inline elements, you can still set the block properties, such as `height` and `width` as well as `margin`. For example:

```
span {
  height: 200px; width:200px;
  display:inline-block;
}
```

Another way to modify the HTML element flow is to use the `float` property. The `float` property allows an element to float either to the `left` or `right` of the containing element. Other elements that are not floating will be displayed next to the floating element on the side away from the edge of the containing element.

When using the `float` attribute, you need to use the `clear` property to define which sides of the floating element other elements are not allowed to flow around. To illustrate this, consider the following CSS and HTML code. Notice that in Figure 4.16, the paragraph element that does not include the `clear:both` attribute is rendered next to the image that is floating to the left.

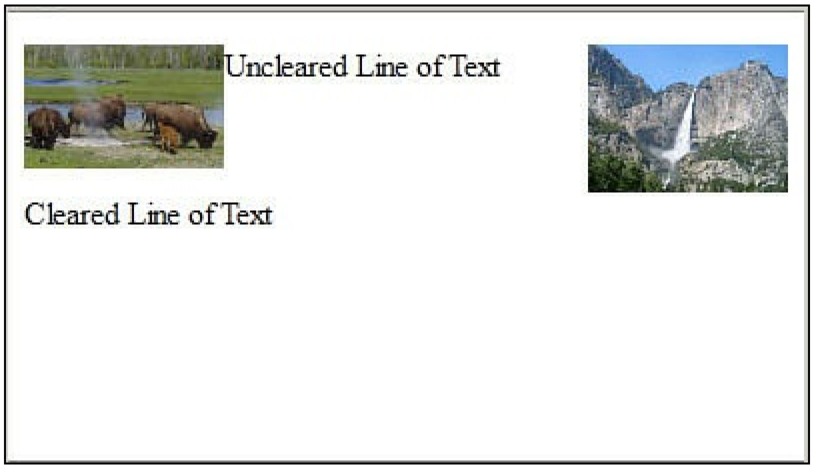


FIGURE 4.16 Floating an element to the left or right modifies the flow. However, you need to `clear` properties that you do not want to flow next to.

CSS rules:

```
#image1 {float:left;}
#image2 {float:right;}
#cleared {clear:both;}
```

HTML elements:

Click here to view code image

```
<img id="image1" src="images/lesson0302.jpg" width="100" />
<img id="image2" src="images/lesson0303.jpg" width="100" />
<p id="uncleared">Uncleared Line of Text</p>
<p id="cleared">Cleared Line of Text</p>
```

Positioning HTML Elements from CSS

The `position` property enables you to define how an element is positioned by the browser. By default, elements are static and flow with the other elements in the document. However, by changing the value of the `position` property, you can change that behavior.

The following list describes each of the nonstatic position values:

- **fixed**—Positions the element relative to the browser window.
- **absolute**—Positions the element relative to the first nonstatic container element. At least one of the element's containers must already be defined in a nonstatic position.
- **relative**—Positions the element relative to the normal position it would have been in.

After you change the position property to a nonstatic value, you can use the `top`, `left`, `bottom`, and `right` properties to set the positioning. For example, to set the top-left corner of an element to 50 pixels from the top and 100 pixels from the top-left edge of the browser window, you would use the following CSS code:

```
position:fixed;
top:50px;
left:100px;
```

Z-Index

When you begin positioning HTML elements using nonstatic positioning, you may find a situation where one element overlaps another. The `z-index` property allows you to specify which element should be in front. Elements with a higher `z-index` number are in front of elements with lower `z-index` values. The following CSS `<div>` elements would be in front of `<p>` paragraph elements:

```
div { z-index:2; }
p { z-index:1; }
```

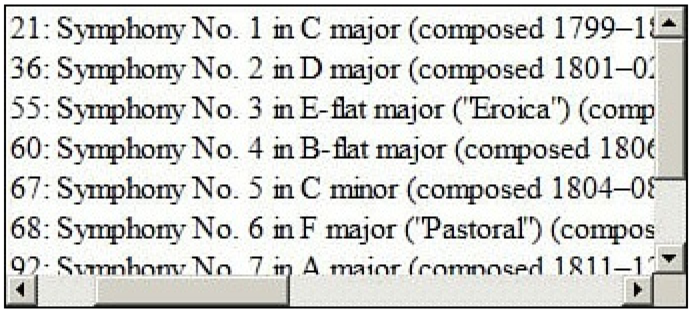
Overflow

The `overflow` CSS property is provided to solve the problem of trying to display an item that is too large for the area in the web page you have designed. The `overflow` property can be set to the following:

- **visible**—Default behavior.
- **hidden**—Will hide the element if the content is too big.
- **scroll**—Adds scrollbars on the bottom and right. If the content exceeds the size constraint, the scrollbars become active so you can view the full contents of the container without affecting the flow of the rest of the page.
- **auto**—Will add the scrollbars, if necessary, to keep the element to the size constraints you place on it.

Figure 4.17 shows an example of a list that gets constrained inside of a `<ul>` element that has the following CSS rule with the overflow `property`:

```
ul{
  border:1px solid;
  height:150px; width:300px;
  overflow:auto;
}
li{ width:800px; }
```



```
21: Symphony No. 1 in C major (composed 1799–1
36: Symphony No. 2 in D major (composed 1801–0
55: Symphony No. 3 in E-flat major ("Eroica") (comp
60: Symphony No. 4 in B-flat major (composed 180
67: Symphony No. 5 in C minor (composed 1804–0
68: Symphony No. 6 in F major ("Pastoral") (compos
92: Symphony No. 7 in A major (composed 1811–1
```

FIGURE 4.17 Adding the overflow property to a `<ul>` element results in the list being constrained within scrollbars.

Try it Yourself: Using CSS to Lay Out Web Page Components

In this example, you put into practice several of the layout properties. In the example, you create two `<div>` elements and place them in fixed positions in the web page using the following steps. The full HTML and CSS code can be found in Listing 4.6 and Listing 4.7:

1. Create a new HTML document named web_layout.html in the lesson04 folder in Eclipse.
2. Add the code shown in Listing 4.6. The code links to an external style sheet named css/web_layout.css and then defines two `<div>` elements: one that contains a set of images and the second that contains `<p>` elements to display song info.
3. Copy the image files listed in Listing 4.6 from the book's website under /images to the /images folder in your Eclipse project.
4. Create a new CSS document named web_layout.css in the lesson04/css folder in Eclipse.
5. Add the following lines to the new CSS file to set basic spacing and size for the `<img>` elements. The `margin` property gives them space around each other and in the div element. The `float` property allows them to float to the left of their container:

```
img{
  width:48px;
  margin:-5px;
  float:left;
}
```

6. Add the following lines of code to define the look and feel of both `<div>` elements. The `position` is set to `fixed`, but no positioning is specified yet. We also add `padding` to give the items inside the `<div>` room away from the border:

```
#buttons, #songInfo{
  position:fixed;
  padding:5px;
  display:inline-block;
  border:8px ridge blue;
  border:radius 5px;
}
```

7. Add the following CSS rules to position each of the `#songiInfo` and `#buttons <div>` elements individually. Remember that they keep all the values from the previous step. Notice that you set the elements' position by specifying the `top` and `left` positions in the browser window. Also notice that you give the buttons a higher `z-index` number to ensure that it will always be on top of the `songInfo`:

Click here to view code image

```
#songInfo{
  background-color: black;
  top:70px;
  left:100px;
  height:150px;width:300px;
  z-index:1;
}
#buttons{
  background-color: white;
  top:200px;
  left:120px;
  z-index:2;
}
```

8. Add the following rule for the title and year elements. Notice that you set the `position` to `relative` so that you can position them relative to where they would normally flow inside of the `<div>` container. Then when you set the `top` and `left` values, they are relative to the `<p>` items' normal position. By specifying a negative number, you can move the item up toward the top:

```
#title{
  color:lime;
  font-size:15px;
  position:relative;
  left:30px;
  top:25px;
}
#year{
  color:yellow;
  font-size:15px;
  position:relative;
  left:30px;
  top:-40px;
}
```

9. Save the files and open the HTML file in a web browser. You should see results similar to what is shown in Figure 4.18. Notice that the CSS layout styles have positioned the elements in specific positions in the browser.

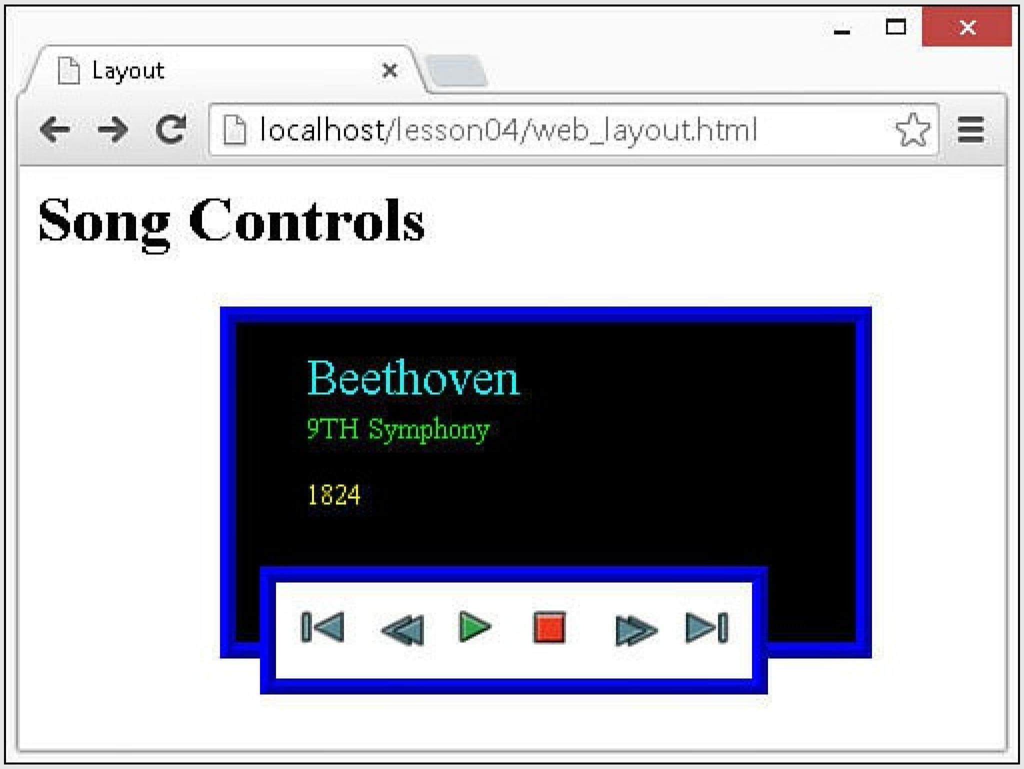


FIGURE 4.18 Adding CSS layout properties allows you to position elements at specific locations in the browser.

LISTING 4.6 web_layout.html HTML That Creates a Pair of `<div>` Elements That Are Styled by Listing 4.7 to Be Song Info and a Set of Playback Controls

Click here to view code image

```
<html>
  <head>
    <title>Layout</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="css/web_layout.css">
  </head>
  <body>
```

```
    <div id="buttons">
      <img src="../images/skipBack.png" />
      <img src="../images/seekBackward.png" />
      <img src="../images/play.png" />
      <img src="../images/stop.png" />
      <img src="../images/seekForward.png" />
      <img src="../images/skipForward.png" />
    </div>
    <div id="songinfo">
    <p id="title">9TH Symphony</p>
    <p id="artist">Beethoven</p>
    <p id="Year">1824</p>
  </div>
    <h1>Song Controls</h1>
  </body>
</html>
```

LISTING 4.7 web_layout.css CSS Code Used to Apply a Page Layout That Places Elements in Fixed Positions

[Click here to view code image]

```
img{
  width:48px;
  margin:-5px;
  float:left;
}
#buttons, #songInfo{
  position:fixed;
  padding:5px;
  display:inline-block;
  border:8px ridge blue;
  border:radius 5px;
}
#songInfo{
  background-color: black;
  top:70px;
  left:100px;
  height:150px;width:300px;
  z-index:1;
}
#buttons{
  background-color: white;
  top:200px;
  left:120px;
  z-index:2;
}
#artist{
  color:cyan;
  font-size:25px;
  position:relative;
  left:30px;
  top:-50px;
}
#title{
  color:lime;
  font-size:15px;
  position:relative;
  left:30px;
  top:25px;
}
#year{
  color:yellow;
  font-size:15px;
  position:relative;
  left:30px;
  top:-40px;
}
```

Preparing CSS Styles for Dynamic Design

One of the coolest features of AngularJS, jQuery, and JavaScript is the capability to adjust CSS styles dynamically. A little bit later in the book, you learn how to do that to create some cool effects. In this section, you are introduced to the concept of preparing for dynamic styles while CSS is still fresh in your mind.

As you define your CSS styling rules, keep in mind that there are a few methods to applying styles dynamically. You can design the CSS in such a way that the dynamic code will be much simpler to implement. The following sections describe some of the methods that you will learn later.

Preparing to Add Classes to HTML Elements Dynamically

The simplest way to dynamically adjust the style of an HTML element is to change which CSS rules are associated with the element. The easiest way to change CSS rules is to use `class` selectors for the different rules that you want to use. That way, from AngularJS, jQuery, and JavaScript, you can add a `class` to apply a style or remove a `class` to remove the style.

This provides two main advantages. First, dynamic code is much simpler because it is easy to add and remove classes. The second is that it is easy to make adjustments to the CSS styles that will be applied, because they will be done inside a .css file or a `<style>` tag.

Using dynamic class rules requires that the rules for the class selectors be completely defined somewhere in the CSS code loaded with the web page. It also requires some thinking about how to implement each CSS rule.

For example, think about adding dynamic interaction with a `<span>` element that is styled to be used as a button. You want the button to be able to have four different states of up, down, inactive, and state changing. You can create a different CSS rule with a selector for the class of those button states. When the button is up, the `class` attribute assigned to the `<span>` would be up; when the user clicks the button, you could change the `class` to state changing, and then when the mouse is released, change the `class` to down.

Preparing to Directly Adjust CSS Properties

You can also directly access and modify specific CSS properties of an element or a set of elements via AngularJS, jQuery, and JavaScript. This provides the advantage of being able to create static CSS definitions in .css files or `<style>` tags and then dynamically change a subset of the property values.

A great example of using JavaScript with CSS property values is moving items around the web page. For instance, suppose you were to define an HTML image element for a scrolling banner that was supposed to scroll horizontally across the screen. You would define the static CSS style that styles the banner as a fixed position in the browser window. Then, because the CSS already defines all the necessary properties to render the banner fixed, all you need to do from AngularJS, jQuery, and JavaScript is to adjust the CSS position properties to scroll the banner.

Summary

This lesson discussed using CSS to quickly and easily alter the look and feel of your web pages. Understanding these concepts is critical to implementing JavaScript, jQuery, and AngularJS to create dynamic web pages and rich Internet applications.

You learned how the CSS rules are structured and how to define selectors that will apply a CSS rule to a specific set of HTML elements. You were introduced to some of the common design and layout styles of CSS.

You learned how to style text, apply color and background images, and create cool borders around HTML elements. You also learned how to understand and control the flow of the HTML elements on the page as well as how to apply fixed positioning.

Q&A

Q. Is it better to use a fixed size and position for HTML elements, or should I try to use more of the HTML flow based on the content size?

A. That is a tough question. For most web pages, consider using more of a flow with relative positioning within elements as much as possible. The reason for that is because of the wide range of displays that may be looking at the web pages. If you need to support mobile displays up to 36-inch monitors, the flow allows for the content to more easily match the browser size. However, if you are designing a dynamic web page as more of a web application, use more fixed sizes that support an average size monitor and then implement a mobile version using the jQuery Mobile techniques you learn later in this book.

Q. Why bother with CSS—can't I just apply the CSS styles directly from jQuery and JavaScript?

A. Sure, everything that you can do with CSS can be applied from JavaScript. However, it is more difficult to maintain the changes, especially when more than one person is involved in the design. Often, some of the best web page designers won't know any jQuery or JavaScript. Also, think about how easy it is to keep multiple versions of a CSS file and tweak them to apply different skins to the web page.

Workshop

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

Quiz

1. What selector would you use to apply a CSS rule to all `<div>` elements?
2. What CSS property allows an element to float past other elements either to the right or to the left?
3. How would you apply a border with rounded corners around a `<div>` element?
4. True or false: You cannot specify exact coordinates in the browser to position an HTML element.
5. What selector would you use to apply a CSS rule only to an HTML element with the class attribute set to "`myClass`" and the id set to "`mainContainer`"?

Quiz Answers

1. `div {...`
2. `float:right` or `float:left`
3. `border:1px solid;border-radius:5px;`
4. False. You can set the `display:fixed` property and then set the top, left, bottom, or right pixel offset to place the HTML element, regardless of where other elements are on the page.
5. `#mainContainer {...`

Exercises

1. Modify the code that you applied in Listing 4.3 to include the following CSS rule that will apply to the span you defined as buttons. The rule should change the style of the HTML element and consequently the look of the menu item when the mouse hovers over it:

```
span:hover {
  width:150px;
  background-color: #2233FF;
  color: white;
  border:5px outset blue;
  border-radius:50%;
}
```

2. Modify the code in Listing 4.6 and Listing 4.7 to wrap the heading in its own border and position it above the other `<div>` elements. You will need to change the position of the existing `<div>` elements to do so.

Lesson 5. Jumping into jQuery and JavaScript Syntax

What You'll Learn in This Lesson:

- Ways to add jQuery and JavaScript to your web pages
- Creating and manipulating arrays of objects
- Adding code logic to JavaScript
- Implementing JavaScript functions for cleaner code

Throughout the book, you'll see several examples of using jQuery and JavaScript to perform various dynamic tasks. jQuery doesn't replace JavaScript; it enhances it by providing an abstract layer to perform certain common tasks, such as finding elements or values, changing attributes and properties of elements, and interacting with browser events.

AngularJS uses JavaScript and jQuery syntax to provide its functionality. It is important for you to understand the jQuery and JavaScript syntax before getting into AngularJS. That is why these are covered first and AngularJS is covered in later lessons.

In this lesson, you learn the basic structure and syntax of JavaScript and how to use jQuery to ease some of the development tasks. The purpose of this lesson is to help you become familiar with the JavaScript language syntax, which is also the jQuery language syntax.

Adding jQuery and JavaScript to a Web Page

Browsers come with JavaScript support already built in to them. That means all you need to do is add your own JavaScript code to the web page to implement dynamic web pages. jQuery, on the other hand, is an additional library, and you will need to add the jQuery library to your web page before adding jQuery scripts.

Loading the jQuery Library

Because the jQuery library is a JavaScript script, you use the `<script>` tag to load the jQuery into your web page. jQuery can either be downloaded to your code directory and then hosted on your web server, or you can use the hosted versions that are available at jQuery.com. The following statement shows an example of each; the only difference is that the first loads it from the jQuery CDN source and the second loads it from the web server:

Click here to view code image

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

Caution

Remember that you need to place the `<script>` element to load the jQuery library before any script elements that are using it. Otherwise, those libraries will not be able to link up to the jQuery code.

The jQuery library downloads can be found at the following location:

http://jquery.com/download/

The jQuery library hosted links can be found at the following location:

http://code.jquery.com/

Implementing Your Own jQuery and JavaScript

JQuery code is implemented as part of JavaScript scripts. To add jQuery and JavaScript to your web pages, first add a `<script>` tag that loads the jQuery library, and then add your own `<script>` tags with your custom code.

The JavaScript code can be added inside the `<script>` element, or the `src` attribute of the `<script>` element can point to the location of a separate JavaScript document. Either way, the JavaScript will be loaded in the same manner.

The following is an example of a pair of `<script>` statements that load jQuery and then use it. The `document.write()` function just writes text directly to the browser to be rendered:

Click here to view code image

```
<script src="http://code.jquery.com/jquery-latest.min.js"></script>
<script>
  function writeIt(){
    document.write("jQuery Version " + $().jquery + " loaded.");
  }
</script>
```

Note

The `<script>` tags do not need to be added to the `<head>` section of the HTML document; they can also be added in the body. It’s useful to add simple scripts directly inline with the HTML elements that are consuming them.

Accessing HTML Event Handlers

So after you add your JavaScript to the web page, how do you get it to execute? The answer is that you tie it to the browser events. Each time a page or element is loaded, the user moves or clicks the mouse or types a character, an HTML event is triggered.

Each supported event is an attribute of the object that is receiving the event. If you set the attribute value to a JavaScript function, the browser will execute your function when the event is triggered.

For example, the following will execute the `writeIt()` function when the body of the HTML page is loaded:

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

Try it Yourself: Implementing JavaScript and jQuery

Those are the basic steps. Now it is time to try it yourself. Use the following steps to add jQuery to your project and use it dynamically in a web page:

1. In Eclipse, create a source folder named lesson05.
2. In the same folder as the lesson05 folder, add an additional directory called js.
3. Now create a source file named jquery_version.html in the lesson05 folder.
4. Add the usual basic elements (html, head, body).
5. Inside the `<head>` element, add the following line to load the library you just downloaded:

Click here to view code image

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
```

6. Now you can add your own `<script>` tag with the following code to print out the jQuery version to the browser windows:

Click here to view code image

```
<script>
  function writeIt(){
    document.write("jQuery Version " + $().jquery + " loaded.");
  }
</script>
```

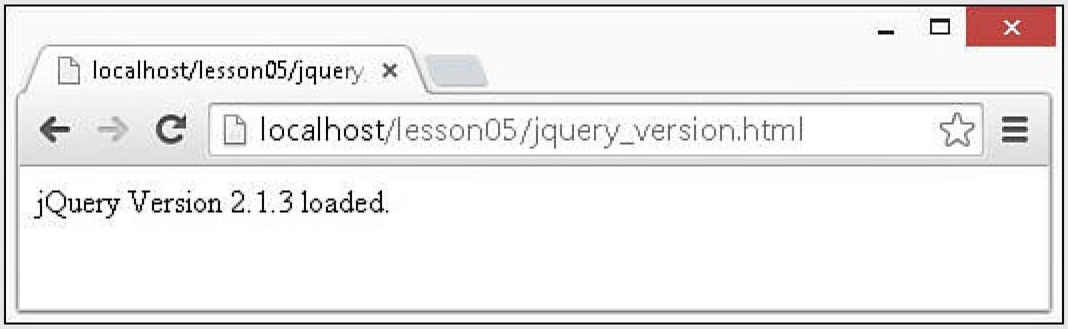
7. To have your script execute when the document is loaded, tie the `writeIt()` function to the `<body> onload` event using the following line:

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

8. Save the file and view it in your web browser at the following location. The output should be similar to Figure 5.1:

Click here to view code image

```
http://localhost/lesson06/jquery_version.html
```


localhost/lesson05/jquery
localhost/lesson05/jquery_version.html

jQuery Version 2.1.3 loaded.

FIGURE 5.1 The function `writeIt()` is executed when the body loads and writes the jQuery version to the browser.

LISTING 5.1 jquery_version.html Very Basic Example of Loading Using jQuery in a Web Page to Print Out Its Own Version

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery Version</title>
    <meta charset="utf-8" />
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script>
      function writeIt(){
        document.write("jQuery Version " + $().jquery + " loaded.");
      }
    </script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

Accessing the DOM

One of the most important aspects of JavaScript, and especially jQuery, is the capability to access and manipulate the DOM. Accessing the DOM is how you make the web page dynamic by changing styles, size, position, and values of elements.

In the following sections, you learn about accessing the DOM through traditional methods via JavaScript and the much improved methods using jQuery selectors. These sections are a brief introduction. You will get plenty of practice as the lessons roll on.

Using Traditional JavaScript to Access the DOM

Traditionally, JavaScript uses the global `document` object to access elements in the web page. The simplest method of accessing an element is to directly refer to it by `id`. For example, if you have a paragraph with the `id="question"`, you can access it via the following JavaScript `getElementById()` function:

Click here to view code image

```
var q = document.getElementById("question");
...
<p id="question">Which method to you prefer?</p>
```

Another helpful JavaScript function that you can use to access the DOM elements is `getElementsByTagName()`. This returns a JavaScript array of DOM elements that match the tag name. For example, to get a list of all the `<p>` elements, use the following function call:

Click here to view code image

```
var paragraphs = document.getElementsByTagName("p");
```

Using jQuery Selectors to Access HTML Elements

Accessing HTML elements is one of jQuery's biggest strengths. jQuery uses selectors that are very similar to CSS selectors to access one or more elements in the DOM; hence the name jQuery. jQuery returns back either a single element or an array of jQuerified objects. jQuerified means that additional jQuery functionality has been added to the DOM object, allowing for much easier manipulation.

The syntax for using jQuery selectors is `$(`**selector**`).`**action**`()`, where **selector** is replaced by a valid selector and **action** is replaced by a jQuerified action attached to the DOM element(s).

For example, the following command finds all paragraph elements in the HTML document and sets the CSS `font-weight` property to `bold`:

Click here to view code image

```
$("p").css('font-weight', 'bold');
```

Try it Yourself: Using jQuery and JavaScript to Access DOM Elements

Now to solidify the concepts, you'll run through a quick example of accessing and modifying DOM elements using both jQuery and JavaScript. Use the following steps to build the HTML document shown in Listing 5.2:

1. Create a source file named dom_elements.html in the lesson05 folder.
2. Add the usual basic elements (html, head, body).
3. Inside the `<head>` element, add the following line to load the library you just

downloaded:

Click here to view code image

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js">
</script>
```

4. Add the following `<script>` element that accesses the DOM using both the JavaScript and jQuery methods. Notice that with jQuery, two actions are chained together. The first sets the CSS `font-weight` property and the second changes text contained in element. With JavaScript, you use the `getElementById()` method, and then you set the `innerHTML` property directly in the DOM to change the text displayed in the browser:

Click here to view code image

```
<script>
  function writeIt(){
    $("#heading").css('font-weight', 'bold').html("jQuery");
    var q = document.getElementById("question");
    q.innerHTML = "I Prefer jQuery!";
  }
</script>
```

5. To have your script execute when the document is loaded, tie the `writeIt()` function to the `<body> onload` event using the following line:

Click here to view code image

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

6. Add the following `<p>` elements to the `<body>` to provide containers for the JavaScript code to access:

Click here to view code image

```
<p id="heading">jQuery or JavaScript</p>
<p id="question">Which method to you prefer?</p>
```

7. Save the file and view it in a web browser. The output should be similar to Figure 5.2.

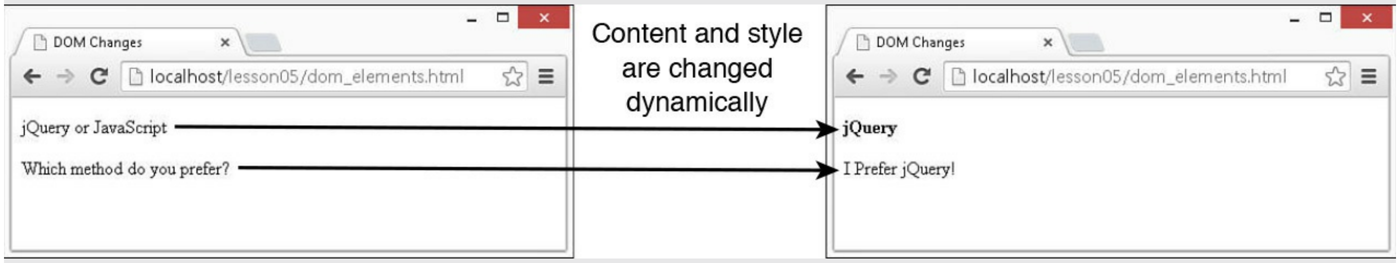

DOM Changes
localhost/lesson05/dom_elements.html
jQuery or JavaScript
Which method do you prefer?
Content and style are changed dynamically
DOM Changes
localhost/lesson05/dom_elements.html
jQuery
I Prefer jQuery!

FIGURE 5.2 The function `writeIt()` is executed when the body loads and changes the content and appearance of the text.

LISTING 5.2 Very Basic Example of Using JavaScript and jQuery to Access DOM Elements

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Changes</title>
    <meta charset="utf-8" />
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script>
      function writeIt(){
        $("#heading").css('font-weight', 'bold').html("jQuery");
        var q = document.getElementById("question");
        q.innerHTML = "I Prefer jQuery!";
      }
    </script>
  </head>
  <body onload="writeIt()">
    <p id="heading">jQuery or JavaScript</p>
    <p id="question">Which method do you prefer?</p>
  </body>
</html>
```

Understanding JavaScript Syntax

Like any other computer language, JavaScript is based on a rigid syntax where specific words mean different things to the browser as it interprets the script. This section is designed to walk you through the basics of creating variables, working with data types, and using looping and functions in JavaScript to manipulate your web pages.

Tip

For the simple JavaScript examples in this lesson, you can test them by starting Node.js using the `node` command from a console prompt to bring up the Node.js interpreter. From the interpreter, you can type in JavaScript code and have it execute as you type each line.

Creating Variables

The first place to begin with in JavaScript is variables. Variables are a means to name data so that you can use that name to temporarily store and access data from your JavaScript files. Variables can point to simple data types, such as numbers or strings, or they can point to more complex data types, such as objects.

To define a variable in JavaScript, you must use the `var` keyword and then give the variable a name; for example:

```
var myData;
```

You can also assign a value to the variable in the same line. For example, the following line of code creates a variable `myString` and assigns it the value of "Some Text":

```
var myString = "Some Text";
```

This works as well as the following:

```
var myString;
myString = "Some Text";
```

After you have declared the variable, you can use the name to assign the variable a value and access the value of the variable. For example, the following code stores a string into the `myString` variable and then uses it when assigning the value to the `newString` variable:

Click here to view code image

```
var myString = "Some Text";
var newString = myString + " Some More Text";
```

Your variable names should describe the data that is stored in them so that it is easy to use them later in your program. The only rule for creating variable names is that they must begin with a letter, `$`, or `_`, and they cannot contain spaces. Also remember that variable names are case sensitive, so using `myString` is different from `MyString`.

Understanding JavaScript Data Types

JavaScript uses data types to determine how to handle data that is assigned to a variable. The variable type will determine what operations you can perform on the variable, such as looping or executing. The following list describes the most common types of variables that we will be working with through the book:

- **String**—Stores character data as a string. The character data is specified by either single or double quotes. All the data contained in the quotes will be assigned to the string variable. For example:

Click here to view code image

```
var myString = 'Some Text';
var anotherString = "Some Other Text";
```

- **Number**—Stores the data as a numerical value. Numbers are useful in counting, calculations, and comparisons. Some examples are as follows:

```
var myInteger = 1;
var cost = 1.33;
```

- **Boolean**—Stores a single bit that is either true or false. Booleans are often used for flags. For example, you might set a variable to false at the beginning of some code and then check it on completion to see whether the code execution hit a certain spot. The following shows an example of defining a true and a false variable:

```
var yes = true;
var no = false;
```

- **Array**—An indexed array is a series of separate distinct data items all stored under a single variable name. Items in the array can be accessed by their zero-based index using the `[index]`. The following is an example of creating a simple array and then accessing the first element, which is at index 0:

Click here to view code image

```
var arr = ["one", "two", "three"]
var first = arr[0];
```

- **Associative Array/Objects**—JavaScript does support the concept of an associative array, meaning accessing the items in the array by a name instead of an index value. However, a better method is to use an object literal. When you use an object literal, you can access items in the object using `object.property` syntax. The following example shows how to create and access an object literal:

Click here to view code image

```
var obj = {"name":"Brad", "occupation":"Hacker", "age", "Unknown"};
var name = obj.name;
```

- **Null**—At times, you do not have a value to store in a variable, either because it hasn't been created or you are no longer using it. At this time, you can set a variable to `null`. That way, you can check the value of the variable in your code and use it only if it is not `null`:

```
var newVar = null;
```

Note

JavaScript is a typeless language, meaning you do not need to tell the browser what data type the variable is; the interpreter will automatically figure out the correct data type for the variable.

Using Operators

JavaScript operators provide the capability to alter the value of a variable. You are already familiar with the = operator because you used it several times in the book already. JavaScript provides several operators that can be grouped into two types—arithmetic and assignment.

Arithmetic Operators

Arithmetic operators are used to perform operations between variable and direct values. Table 5.1 shows a list of the arithmetic operations along with the results that get applied.

Operator	Description	Example	Resulting x	Resulting y
`+`	Addition	`x=y+5`	`9"45"`	`4`
		`x=y+"5"`	`"Four44"`	`4`
		`x="Four"+y+"4"`		`4`
`-`	Subtraction	`x=y-2`	`2`	`4`
`++`	Increment	`x=y++`	`4`	`5`
		`x=++y`	`5`	`5`
`--`	Decrement	`x=y--`	`4`	`3`
		`x=--y`	`3`	`3`
`*`	Multiplication	`x=y*4`	`16`	`4`
`/`	Division	`x=10/y`	`2.5`	`4`
`%`	Modulous (remainder of Division)	`x=y%3`	`1`	`4`

TABLE 5.1 Table Showing JavaScripts' Arithmetic Operators as Well as Results Based on y=4 to Begin With

Tip

The + operator can also be used to add strings or strings and numbers together. This allows you to quickly concatenate strings and add numerical data to output strings. Table 5.1 shows that when adding a numerical value and a string value, the numerical value is converted to a string, and then the two strings are concatenated.

Assignment Operators

Assignment operators are used to assign a value to a variable. You are probably used to the = operator, but there are several forms that allow you to manipulate the data as you assign the value. Table 5.2 shows a list of the assignment operations along with the results that get applied.

Operator	Example	Equivalent Arithmetic Operators	Resulting x
=	x=5	x=5	5
+=	x+=5	x=x+5	15
-=	x-=5	x=x-5	5
*=	x*=5	x=x*5	50
/=	x/=5	x=x/5	2
%=	x%=5	x=x%5	0

TABLE 5.2 JavaScripts' Assignment Operators as Well as Results Based on x=10 to Begin With

Applying Comparison and Conditional Operators

Conditionals are a way to apply logic to your applications so that certain code will be executed only under the correct conditions. This is done by applying comparison logic to variable values. The following sections describe the comparisons available in JavaScript and how to apply them in conditional statements.

Comparison Operators

A comparison operator evaluates two pieces of data and returns true if the evaluation is correct or false if the evaluation is not correct. Comparison operators compare the value on the left of the operator against the value on the right.

The simplest way to help you understand comparisons is to provide a list with some examples. Table 5.3 shows a list of the comparison operators along with some examples.

Operator	Example	Example	Result
`==`	Is equal to (value only)	`x==8`	false
		`x==10`	true
`===`	Both value and type are equal	`x===10`	true
		`x==="10"`	false
`!=`	Is not equal	`x!=5`	true
`!==`	Both value and type are not equal	`x!=="10"`	true
		`x!==10`	false
`>`	Is greater than	`x>5`	true
`>=`	Is greater than or equal to	`x>=10`	true
`<`	Is less than	`x<5`	false
`<=`	Is less than or equal to	`x<=10`	true

TABLE 5.3 JavaScripts' Comparison Operators as Well as Results Based on x=10 to Begin With

You can chain multiple comparisons together using logical operators. Table 5.4 shows a list of the logical operators and how to use them to chain comparisons together.

Operator	Description	Example	Result
`&&`	and	`(x==10 && y==5)(x==10 && y>x)`	true false
`||`	or	`(x>=10 || y>x)(x<10 && y>x)`	true false
`!`	not	`!(x==y)!(x>y)`	true false
	mix	`(x>=10 && y<x || x==y)((x<y || x>=10) && y>=5)(!(x==y) && y>=10)`	true true false

TABLE 5.4 JavaScripts' Comparison Operators as Well as Results Based on x=10 and y=5 to Begin With

If

An `if` statement enables you to separate code execution based on the evaluation of a comparison. The syntax is shown in the following lines of code where the conditional operators are in `()` parenthesis and the code to execute if the conditional evaluates to true is in `{}` brackets:

```
if(x==5){
  do_something();
}
```

In addition to executing code only within the `if` statement block, you can specify an else block that will get executed only if the condition is false. For example:

```
if(x==5){
  do_something();
} else {
  do_something_else();
}
```

You can also chain `if` statements together. To do this, add a conditional statement along with an `else` statement. For example:

```
if(x<5){
  do_something();
} else if(x<10) {
  do_something_else();
} else {
  do_nothing();
}
```

switch

Another type of conditional logic is the `switch` statement. The `switch` statement allows you to evaluate an expression once and then, based on the value, execute one of many sections of code.

The syntax for the switch statement is the following:

Click here to view code image

```
switch(expression){
  case value:
    <code to execute>
    break;
  case value2:
    <code to execute>
    break;
  default:
    <code to execute if not value or value2>
}
```

This is what is happening. The `switch` statement will evaluate the expression entirely and get a value. The value may be a string, a number, a Boolean, or even an object. The `switch` value is then compared to each value specified by the `case` statement. If the value matches, the code in the `case` statement is executed. If no values match, the `default` code is executed.

Note

Typically, each `case` statement will include a break command at the end to signal a break out of the `switch` statement. If no break is found, code execution will continue with the next `case` statement.

Try it Yourself: Applying If Conditional Logic in JavaScript

To help you solidify using JavaScript conditional logic, use the following steps to build conditional logic into the JavaScript for a dynamic web page. The final version of the HTML document is shown in Listing 5.3:

1. Create a source file named if_logic.html in the lesson05 folder.
2. Create a folder under lesson05 named images.
3. Add your own images for day.png and night.png to the ./images folder in your project or download the ones from the book's website.
4. Add the usual basic elements (html, head, body).
5. Add the following `<script>` element that gets the lesson value using the `Date().getLessons()` JavaScript code. The code uses `if` statements to determine the time of day and does two things: it writes a greeting onto the screen and sets the value of the `timeOfDay` variable:

Click here to view code image

```
<script>
  function writeIt(){
    var lesson = new Date().getLessons();
    var timeOfDay;
    if(lesson>=7 && lesson<12){
      document.write("Good Morning!");
      timeOfDay="morning";
    } else if(lesson>=12 && lesson<18) {
      document.write("Good Day!");
      timeOfDay="day";
    } else {
      document.write("Good Night!");
      timeOfDay="night";
    }
  }
</script>
```

6. Now add the following `switch` statement that uses the value of `timeOfDay` to determine which image to display in the web page:

Click here to view code image

```
    switch(timeOfDay){
      case "morning":
      case "day":
        document.write("<img src='images/day.png' />");
```

```
          break;
        case "night":
          document.write("<img src='images/night.png' />");
          break;
        default:
          document.write("<img src='images/day.png' />");
    }
```

7. Save the file and view it in a web browser. The output should be similar to Figure 5.3, depending on what time of day it is.

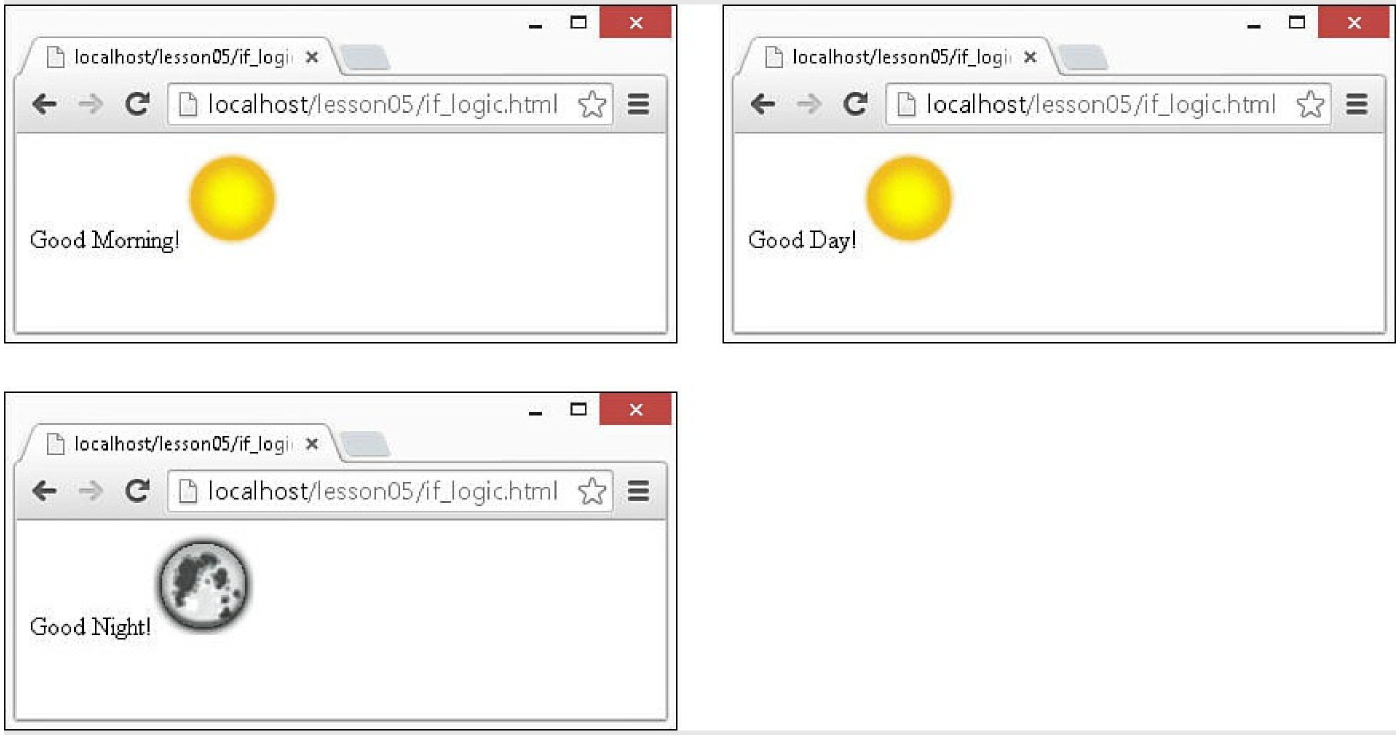


FIGURE 5.3 The function `writeIt()` is executed when the body loads and changes the greeting and image displayed on the web page.

LISTING 5.3 if_logic.html Simple Example of Using Conditional Logic Inside JavaScript

Click here to view code image

```
<!DOCTYPE html>
<html>
 <head>
  <title>If Logic</title>
  <meta charset="utf-8" />
  <script>
   function writeIt(){
    var hour = new Date().getHours();
    var timeOfDay;
    if(hour>=7 && hour<12){
     document.write("Good Morning!");
     timeOfDay="morning";
    } else if(hour>=12 && hour<18) {
     document.write("Good Day!");
     timeOfDay="day";
    } else {
     document.write("Good Night!");
     timeOfDay="night";
    }
    switch(timeOfDay){
     case "morning":
     case "day":
      document.write("<img src='../images/day.png' />");
      break;
     case "night":
      document.write("<img src='../images/night.png' />");
      break;
     default:
      document.write("<img src='../images/day.png' />");
    }
   }
  </script>
 </head>
 <body onload="writeIt()">
 </body>
</html>
```

Implementing Looping

Looping is a means to execute the same segment of code multiple times. This is extremely useful when you need to perform the same tasks on a set of DOM objects, or if you are dynamically creating a list of items.

JavaScript provides functionality to perform `for` and `while` loops. The following sections describe how to implement loops in your JavaScript.

while Loops

The most basic type of looping in JavaScript is the `while` loop. A `while` loop tests an expression and continues to execute the code contained in its `{}` brackets until the expression evaluates to false.

For example, the following `while` loop executes until the value of `i` is equal to 5:

[Click here to view code image]

```
var i = 1;
while (i<5){
  document.write("Iteration " + i + "<br>");
  i++;
}
```

The resulting output to the browser is as follows:

```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
```

do/while Loops

Another type of `while` loop is the `do/while` loop. This is useful if you always want to execute the code in the loop at least once and the expression cannot be tested until the code has executed at least once.

For example, the following `do/while` loop executes until the value of day is equal to Wednesday:

Click here to view code image

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var i=0;
do{
  var day=days[i++];
  document.write("It's " + day + "<br>");
} while (day != "Wednesday");
```

The resulting output to the browser is as follows:

```
It's Monday
It's Tuesday
It's Wednesday
```

for Loops

The JavaScript `for` loop allows you to execute code a specific number of times by using a `for` statement that combines three statements into one using the following syntax:

Click here to view code image

```
for (statement 1; statement 2; statement 3;){
  code to be executed;
}
```

The `for` statement uses those three statements as follows when executing the loop:

- **statement 1**—Executed before the loop begins and not again. This is used to initialize variables that will be used in the loop as conditionals.
- **statement 2**—Expression that is evaluated before each iteration of the loop. If the expression evaluates to true, the loop is executed; otherwise, the `for` loop execution ends.
- **statement 3**—Executed each iteration after the code in the loop has executed.

This is typically used to increment a counter that is used in statement 2.

To illustrate a `for` loop, check out the following example. The example not only illustrates a basic `for` loop, it also illustrates the capability to nest one loop inside another:

Click here to view code image

```
for (var x=1; x<=3; x++){
  for (var y=1; y<=3; y++){
    document.write(x + " X " + y + " = " + (x*y) + "<br>");
  }
}
```

The resulting output to the web browser is as follows:

```
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
```

for/in Loops

Another type of for loop is the `for/in` loop. The `for/in` loop executes on any data type that can be iterated on. For the most part, you will use the `for/in` loop on arrays and objects. The following example illustrates the syntax and behavior of the `for/in` loop on a simple array:

Click here to view code image

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  document.write("It's " + days[idx] + "<br>");
}
```

Notice that the variable `idx` is adjusted each iteration through the loop from the beginning array index to the last. The resulting output is as follows:

```
It's Monday
It's Tuesday
It's Wednesday
It's Thursday
It's Friday
```

Interrupting Loops

When working with loops, at times you need to interrupt the execution of code inside the code itself without waiting for the next iteration. There are two ways to do this using the `break` and `continue` keywords.

The `break` keyword stops execution of the `for` or `while` loop completely. The `continue` keyword, on the other hand, stops execution of the code inside the loop and continues on with the next iteration. Consider the following examples:

Using a `break` if the `day` is `Wednesday`:

Click here to view code image

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  if (days[idx] == "Wednesday")
    break;
  document.write("It's " + days[idx] + "<br>");
}
```

When the value is `Wednesday`, loop execution stops completely:

```
It's Monday
It's Tuesday
```

Using a `continue` if the `day` is `Wednesday`:

Click here to view code image

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  if (days[idx] == "Wednesday")
    continue;
  document.write("It's " + days[idx] + "<br>");
}
```

Notice that the write is not executed for `Wednesday` because of the continue; however, the loop execution did complete:

```
It's Monday
It's Tuesday
It's Thursday
It's Friday
```

Creating Functions

One of the most important parts of JavaScript is making code that is reusable by other code. To do this, you combine your code into functions that perform specific tasks. A function is a series of code statements combined in a single block and given a name. The code in the block can then be executed by referencing that name.

Defining Functions

Functions are defined using the keyword `function` followed by a function name that describes the use of the function, list of zero or more arguments in `()` parentheses, and a block of one or more code statements in `{}` brackets. For example, the following is a function definition that writes “Hello World” to the browser:

Click here to view code image

```
function myFunction(){
  document.write("Hello World");
}
```

To execute the code in `myFunction()`, all you need to do is add the following line to the main JavaScript or inside another function:

```
myFunction();
```

Passing Variables to Functions

Frequently, you will need to pass specific values to functions that they will use when executing their code. Values are passed in comma-delimited form to the function. The function definition will need a list of variable names in the `()` parentheses that match the number being passed in. For example, the following function accepts two arguments, a `name` and `city`, and uses them to build the output string:

Click here to view code image

```
function greeting(name, city){
  document.write("Hello " + name);
  document.write(". How is the weather in " + city);
}
```

To call the `greeting()` function, we need to pass in a `name` value and a `city` value. The value can be a direct value or a previously defined variable. To illustrate this, the following code will execute the `greeting()` function with a `name` variable and a direct string for the `city`:

```
var name = "Brad";
greeting(name, "Florence");
```

Returning Values from Functions

Often, functions will need to return a value to the calling code. Adding a `return` keyword followed by a variable or value will return that value from the function. For example, the following code calls a function to format a string, assigns the value returned from the function to a variable, and then writes the value to the browser:

Click here to view code image

```
function formatGreeting(name, city){
  var retStr = "";
  retStr += "Hello <b>" + name + "</b><br>";
  retStr += "Welcome to " + city + "!";
 return retStr;
}
var greeting = formatGreeting("Brad", "Rome");
```

```
document.write(greeting);
```

You can include more than one `return` statement in the function. When the function encounters a `return` statement, code execution of the function is stopped immediately. If the `return` statement contains a value to return, that value is returned. The following example shows a function that tests the input and returns immediately if it is zero:

Click here to view code image

```
function myFunc(value){
  if (value == 0)
    return;
  code_to_execute_if_value_nonzero;
}
```

Try it Yourself: Creating JavaScript Functions

To help solidify functions, use the following steps to integrate some functions into a JavaScript application. The following steps take you through the process of creating a function, calling it to execute code, and then handling the results returned:

1. Create a source file named js_functions.html in the lesson05 folder.
2. Add the usual basic elements (html, head, body).
3. Add a `<script>` tag to the `<head>` element to house the JavaScript.
4. Insert the following object literal definition at the beginning of the script. The object will have planet names for attributes, and each hero name is a reference to an array of villains:

Click here to view code image

```
var superData = {"Super Man":["Lex Luther"],
                 "Bat Man":["Joker", "Riddler",],
                 "Spider Man":["Green Goblin",
                               "Vulture", "Carnage"],
                 "Thor":["Loki", "Frost Giants"]};
```

5. Add the following function that will be called by the `onload` event. In this function, you use a nested `for/in` loop to iterate through the `superData` object attributes. The outer loop gets the hero `name` and the inner loop loops through the index of the `villains` array:

Click here to view code image

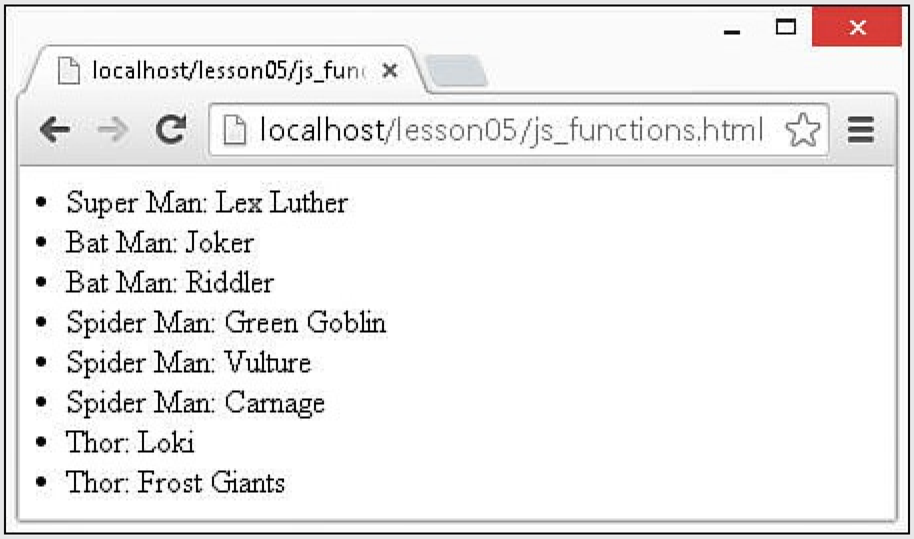
```
function writeIt(){
  for (hero in superData){
    var villains = superData[hero];
    for (villainIdx in villains){
      var villain = villains[villainIdx];
      var listItem = makeListItem(hero, villain);
18              document.write(listItem);
19            }
20          }
21        }
```

6. Notice that on line 16 of the `writeIt()` function is a call to `makeListItem()`. That function needs to return a value that can be used in line 17 to write to the document. Add the following code to create the function. The function takes two arguments: a `name` and a `value`, then generates an HTML string to create a `<li>` element and returns the string:

Click here to view code image

```
22        function makeListItem(name, value){
23          var itemStr = "<li>" + name + ":&nbsp;" + value + "</li>";
24          return itemStr;
25        }
```

7. Save the file and open it in a web browser. You should see the results shown in Figure 5.4. You have just created two JavaScript functions: one that takes no arguments and does not return a value and the other that takes two arguments and returns a formatted HTML string containing the argument strings.


localhost/lesson05/js_fun
localhost/lesson05/js_functions.html

- Super Man: Lex Luther
- Bat Man: Joker
- Bat Man: Riddler
- Spider Man: Green Goblin
- Spider Man: Vulture
- Spider Man: Carnage
- Thor: Loki
- Thor: Frost Giants

FIGURE 5.4 The function `writeIt()` is executed, which iterates through the `moonData` object and makes calls to the `makeListItem()` function to format the planet and moon names as an HTML `<li>` element.

LISTING 5.4 js_functions.html Simple Example of JavaScript Functions

Click here to view code image

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <title>JavaScript Functions</title>
```

```
    <meta charset="utf-8" />
    <script>
      var superData = {"Super Man":["Lex Luther"],
                       "Bat Man":["Joker", "Riddler"],
                       "Spider Man":["Green Goblin",
                                     "Vulture", "Carnage"],
                       "Thor":["Loki", "Frost Giants"]};
      function writeIt(){
        for (hero in superData){
          var villains = superData[hero];
          for (villainIdx in villains){
            var villain = villains[villainIdx];
            var listItem = makeListItem(hero, villain);
            document.write(listItem);
          }
        }
      }
      function makeListItem(name, value){
        var itemStr = "<li>" + name + ":&nbsp;" + value + "</li>";
        return itemStr;
      }
    </script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

Understanding Variable Scope

After you start adding conditions, functions, and loops to your JavaScript applications, you need to understand variable scoping. Variable scope is simply this: "what is the value of a specific variable name at the current line of code being executed."

JavaScript enables you to define both a global and a local version of the variable. The global version is defined in the main JavaScript, and local versions are defined inside functions. When you define a local version in a function, a new variable is created in memory. Within that function, you will be referencing the local version. Outside that function, you will be referencing the global version.

To understand variable scoping a bit better, consider the following code:

Click here to view code image

```
<script>
  var myVar = 1;
  function writeIt(){
    var myVar = 2;
    document.write(myVar);
    writeMore();
  }
  function writeMore(){
```

```
09     document.write(myVar);
10   }
11 </script>
```

The global variable `myVar` is defined on line 2. Then on line 4, a local version is defined within the `writeIt()` function. So, line 5 will write to the document. Then in line 6, `writeMore()` is called. Because there is no local version of `myVar` defined in `writeMore()`, the value of the global `myVar` is written in line 9.

Adding Error Handling

An important part of JavaScript coding is adding error handling for instances where there may be problems. By default, if a code exception occurs because of a problem in your JavaScript, the script fails and does not finish loading. This is not usually the desired behavior.

Try/Catch Blocks

To prevent your code from totally bombing out, use `try/catch` blocks that can handle problems inside your code. If JavaScript encounters an error when executing code in a `try/catch` block, it will jump down and execute the `catch` portion instead of stopping the entire script. If no error occurs, all of the `try` will be executed and none of the `catch`.

For example, the following `try/catch` block will execute any code that replaces `your_code_here`. If an error occurs executing that code, the error message followed by the string ": happened when loading the script" will be written to the document:

Click here to view code image

```
try {
    your_code_here
} catch (err) {
  document.write(err.message + ": happened when loading the script");
}
```

Throw Your Own Errors

You can also throw your own errors using a `throw` statement. The following code illustrates how to add throws to a function to `throw` an error, even if a script error does not occur:

Click here to view code image

```
<script>
  function sqrRoot(x) {
    try {
      if(x=="")    throw "Can't Square Root Nothing";
      if(isNaN(x)) throw "Can't Square Root Strings";
      if(x<0)      throw "Sorry No Imagination";
```

```
   return "sqrt("+x+") = " + Math.sqrt(x);
  } catch(err){
   return err;
  }
 }
 function writeIt(){
  document.write(sqrRoot("four") + "<br>");
  document.write(sqrRoot("") + "<br>");
  document.write(sqrRoot("4") + "<br>");
  document.write(sqrRoot("-4") + "<br>");
 }
</script>
```

The function `sqrRoot()` accepts a single argument `x`. It then tests `x` to verify that it is a positive number and returns a string with the square root of `x`. If `x` is not a positive number, the appropriate error is thrown and returned to `writeIt()`.

Using **finally**

Another valuable tool in exception handling is the `finally` keyword. A `finally` keyword can be added to the end of a `try/catch` block. After the `try/catch` blocks are executed, the `finally` block is always executed. It doesn't matter if an error occurs and is caught or if the `try` block is fully executed.

Following is an example of using a `finally` block inside a web page:

Click here to view code image

```
function testTryCatch(value){
 try {
  if (value < 0){
   throw "too small";
  } else if (value > 10){
   throw "too big";
  }
  your_code_here
 } catch (err) {
  document.write("The number was " + err.message");
 } finally {
  document.write("This is always written.");
 }
}
```

Summary

In this lesson, you learned the basics of adding jQuery and JavaScript to web pages. The basic data types that are used in JavaScript and, consequently, jQuery were described. You learned some of the basic syntax of applying conditional logic to JavaScript applications. You also learned how to compartmentalize your JavaScript applications into functions that can be reused in other locations. Finally, you learned some ways to handle JavaScript errors in your script before the browser receives an exception.

Q&A

Q. When should you use a regular expression in string operations?

A. That depends on your understanding of regular expressions. Those who use regular expressions frequently and understand the syntax well would almost always rather use a regular expression because they are so versatile. If you are not familiar with regular expressions, it takes time to figure out the syntax, and so you will want to use them only when you need to. The bottom line is that if you need to manipulate strings frequently, it is absolutely worth it to learn regular expressions.

Q. Can I load more than one version of jQuery at a time?

A. Sure, but there really isn't a valid reason to do that. The one that gets loaded last will overwrite the functionality of the previous one. Any functions from the first one that were not overwritten may be completely unpredictable because of the mismatch in libraries. The best bet is to develop and test against a specific version and update to a newer version only when there is added functionality that you want to add to your web page.

Workshop

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

Quiz

1. What is the difference between == and === in JavaScript?
2. What is the difference between the `break` and `continue` keywords?
3. When should you use a `finally` block?
4. What is the resulting value when you add a string "1" to a number 1, ("1"+1)?

Quiz Answers

1. == compares only the relative value; === compares the value and the type.
2. `break` will stop executing the loop entirely, whereas `continue` will only stop executing the current iteration and then move on to the next.
3. When you have code that needs to be executed even if a problem occurs in the `try` block.

4. The string "11" because the number is converted to a string and then concatenated.

Exercises

1. Open js_functions.html and modify it to create a table instead of a list. You will need to add code to the `writeIt()` function that writes the `<table>` open tag before iterating through the planets and then the closing tag after iterating through the planets. Then modify the `makeListItem()` function to return a string in the form of:

Click here to view code image

```
<tr><td>planent</td><td>moon</td></tr>
```

2. Modify if_logic.html to include some additional times with different messages and images. For example, between 8 and 9, you could add the message "go to work" with a car icon, and between 5 and 6, you could add the message "time to go home" with a home icon. You will need to add some additional cases to the switch statement and set the `timeOfDay` value accordingly.

Lesson 6. Understanding and Using JavaScript Objects

What You'll Learn in This Lesson:

- How to access JavaScript objects
- Creating and manipulating strings
- Sorting JavaScript arrays
- Searching arrays and strings
- How to create your own custom objects

Much of the code in JavaScript revolves around objects. Objects are a convenient and easy way to group functionality and data together for a variety of purposes. Objects allow you to more easily write code that is clear and easy to implement.

There are four main types of objects that you will be working with in jQuery and JavaScript: DOM, built-in, user-defined, and jQuery. You will also be working with these types of objects in AngularJS along with some additional objects that are described in later lessons in the AngularJS section.

The following sections define object syntax, take you through the most common built-in objects, and show you how to create your own custom objects. DOM objects and jQuery objects are covered in upcoming lessons.

Tip

For the simple JavaScript examples in this lesson, you can test them by starting Node.js using the `node` command from a console prompt to bring up the Node.js interpreter. From the interpreter, you can type in JavaScript code and have it execute as you type each line.

Using Object Syntax

To use objects in JavaScript effectively, you need to have an understanding of their structure and syntax. An object is really a container to group multiple values and, in some instances, functions together. The values of an object are called *properties*, and functions are called *methods*. Object syntax is very straightforward—you use the object name, then a dot, then the property or method name.

Creating a New Object Instance

To use a JavaScript object, you must first create an instance of the object. Object instances are created using the `new` keyword with the object constructor name. For example, to create a Number object, you use the following line of code:

```
var x = new Number("5");
```

Accessing Object Properties

Almost all JavaScript objects have values that are accessible via a standard dot-naming syntax. For example, consider an object with the variable name `user` that contains a property `firstName`. The following statement accesses the value of the `firstName` property of the user object:

```
user.FirstName
```

Accessing Object Methods

Many JavaScript objects also have methods. An object method is a function that is attached to the object as a name. The function can be called by using the dot syntax to reference the method name. For example, if the object named `user` had a method called `getFullName()`, that function can be called by the following statement:

```
user.getFullName()
```

Assigning New Values and Methods to Objects

One of the coolest things about objects in JavaScript is that you can assign new property and method values to them at any point using the dot syntax. It doesn’t matter if it is a built-in object or one of your own custom objects. That value can then be accessed later using the same dot syntax.

The following is an example of adding a method and property to the main document object that we have been using for several examples. Notice that we define a simple function that writes the `document.me` property to the browser. Then we assign a value to `document.me` and the function `writeMe()` without the `()` as `document.writeMe`. At that point, we can call `document.writeMe` and access the property `document.me`:

Click here to view code image

```
<script>
  function writeMe(){
    document.write(document.me);
  }
  document.me = "Brad Dayley";
  document.writeMe = writeMe;
  document.writeMe();
</script>
```

Caution

If you assign a new value to an object, the existing properties or methods with the same name will be overwritten. When assigning values to objects, be careful that you do not accidentally overwrite an existing property or method.

Understanding Built-in Objects

JavaScript has several built-in objects that provide a specific set of functionality. Using these built-in objects will save time because they provide already coded and tested methods to handle data. The following sections don't include all the JavaScript objects, but cover the ones that you need for this book.

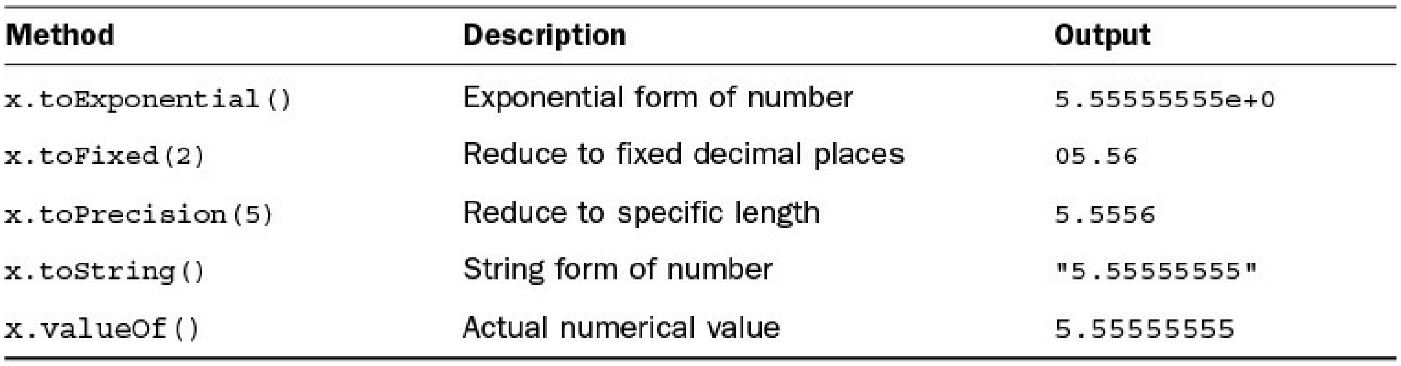
Number

The `Number` object provides functionality that is useful when dealing with numbers. Creating a number object is different from just assigning a number to a variable. When you create a `Number` object, you also get a set of methods that can be used with it.

The `Number` object provides the methods listed in Table 6.1. Follow the output results for each of these methods in Table 6.1 based on the following object creation:

Click here to view code image

```
var x = new Number("5.55555555");
```



Method	Description	Output
`x.toExponential()`	Exponential form of number	`5.55555555e+0`
`x.toFixed(2)`	Reduce to fixed decimal places	`05.56`
`x.toPrecision(5)`	Reduce to specific length	`5.5556`
`x.toString()`	String form of number	`"5.55555555"`
`x.valueOf()`	Actual numerical value	`5.55555555`

TABLE 6.1 Using the Built-In Number Object Methods

We chose to pass the number `5.555555555` in as a string instead of a number to illustrate that `Number()` can accept a string, a number, or a variable representing a string or a number. The string must be a number string or the value of the `Number` objects will be `NaN` (not a number).

It is a good idea to test values before creating the Number object. To test a value and determine whether it is a number, use the `isNaN()` function. `isNaN()` returns `false` if the value is a number or a string that can be a number.

Note

You can use a hexadecimal string when checking for numbers. This is very useful if you are working with hex color values, such as #e0ffff. The string format to use in JavaScript is “0xe0ffff”.

String

The String object is by far the most commonly used in JavaScript. JavaScript automatically creates a String object anytime you define a variable that has a string data type. For example:

```
var myStr = "Teach Yourself jQuery & JavaScript in 24 Lessons";
```

When you create a string, several special characters can’t be directly added to the string. For these characters, JavaScript provides a set of escape codes described in Table 6.2.

Escape	Description	Example	Output String
`\'`	Single quote mark	`"couldn\'t be"`	couldn’t be
`\"`	Double quote mark	`"I \"think\" I \"am\""`	I “think” I “am”
`\\`	Backslash	`"one\\two\\three"`	one\two\three
`\n`	New line	`"I am\nI said"`	I am I said
`\r`	Carriage return	`"to be\ror not"`	to be or not
`\t`	Tab	`"one\ttwo\tthree"`	one two three
`\b`	Backspace	`"correction\b\b\bion"`	correction
`\f`	Form feed	`"Title A\fTitle B"`	Title A then Title B on a new page (mostly for printing)

TABLE 6.2 Escape Codes Used in JavaScript Strings

To get the length of the string, you can use the length property of the String object, for example:

```
var numOfChars = myStr.length;
```

The String object has several functions that allow you to access and manipulate the string in various ways. The methods for string manipulation are described in Table 6.3.

Method	Description
`charAt(index)`	Returns the character at the specified index.
`charCodeAt(index)`	Returns the unicode value of the character at the specified index.
`concat(str1, str2, ...)`	Joins two or more strings and returns a copy of the joined strings.
`fromCharCode()`	Converts unicode values to actual characters.
`indexOf(subString)`	Returns the position of the first occurrence of a specified `subString` value. Returns –1 if the substring is not found.
`lastIndexOf(subString)`	Returns the position of the last occurrence of a specified `subString` value. Returns –1 if the substring is not found.
`match(regex)`	Searches the string and returns all matches to the regular expression.
`replace(subString/regex, replacementString)`	Searches the string for a match of the `subString` or regular expression, and replaces the matched substring with a new substring.
`search(regex)`	Searches the string based on the regular expression and returns the position of the first match.
`slice(start, end)`	Returns a new string that has the portion of the string between the `start` and `end` positions removed.
`split(sep, limit)`	Splits a string into an array of substrings based on a `separator` character or regular expression. The optional `limit` argument defines the maximum number of splits to make, starting from the beginning.
`substr()`	Extracts the characters from a string, beginning at a specified start position, and through the specified number of characters.
`substring(from, to)`	Returns a substring of characters between the `from` and `to` index.
`toLowerCase()`	Converts the string to lowercase.
`toUpperCase()`	Converts a string to uppercase.
`valueOf()`	Returns the primitive string value.

TABLE 6.3 `String` Object Methods Used to Manipulate JavaScript Strings

To get you started on using the functionality provided in the `String` object, the following sections describe some of the common tasks that can be done using `String` object methods.

Combining Strings

Multiple strings can be combined either by using a + operation or by using the `concat()` function on the first string. For example, in the following code, `sentence1` and `sentence2` will be the same:

Click here to view code image

```
var word1 = "Today ";
var word2 = "is ";
var word3 = "tomorrows\' ";
var word4 = "yesterday.";
var sentence1 = word1 + word2 + word3 + word4;
var sentence2 = word1.concat(word2, word3, word4);
```

Searching a String for a Substring

To tell if a string is a substring of another, you can use the `indexOf()` method. For example, the following code writes the string to the browser only if it contains the word “think”:

Click here to view code image

```
var myStr = "I think, therefore I am.";
if (myStr.indexOf("think") != -1){
  document.write(myStr);
}
```

Replacing a Word in a String

Another common `String` object task is replacing one substring with another. To replace a word/phrase in a string, use the `replace()` method. The following code replaces the text `"<username>"` with the value of the variable username:

Click here to view code image

```
var username = "Brad";
var output = "<username> please enter your password: ";
output = output.replace("<username>", username);
```

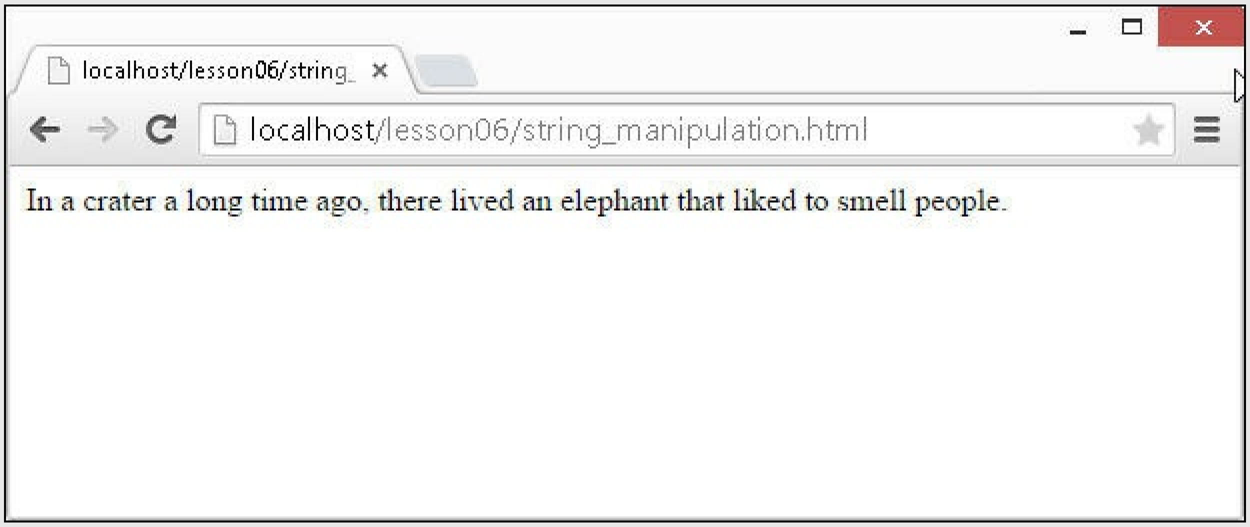
Splitting String into an Array

A common task with strings is to split them into arrays using a separator character. For example, the following code splits a time string into an array of its basic parts using the `split()` method on the `":"` separator:

```
var t = "12:10:36";
var tArr = t.split(":");
var lesson = tArr[0];
var minute = tArr[1];
var second = tArr[2];
```

Try it Yourself: Manipulating Strings in JavaScript

In this section, you follow step by step the process of manipulating some strings to produce a mini-madlib. It is just a basic example, but it should help solidify `String` objects a bit for you. The full code is in Listing 6.1, and the results are displayed in Figure 6.1.


localhost/lesson06/string_
localhost/lesson06/string_manipulation.html

In a crater a long time ago, there lived an elephant that liked to smell people.

FIGURE 6.1 Outputted madlib text.

1. In Eclipse, create a source folder named lesson06.
2. Create a source file named string_manipulation.html in the lesson06 folder.
3. Add the usual basic elements (html, head, body).
4. Inside the `<head>` element, add a `<script>` element with a function named `writeIt()` and link it to the `onload` event of the `<body>` element, as shown next:

Click here to view code image

```
<script>
  function writeIt(){
...
  }
</script>
  </head>
  <body onload="writeIt()">
```

5. Add the following three string definitions that will house each line of the madlib text:

Click here to view code image

```
    var line1 = "In a [place] a long time ago, ";
    var line2 = "there lived an [animal] that liked to ";
    var line3 = "[action] people.";
```

6. Add the following three string definitions be used to populate items in the

madlib:

Click here to view code image

```
var place = "crater";
var animal = "elephant";
var action = "smell";
```

7. Add the following line to combine the lines of text as a single string:

Click here to view code image

```
var madlib = line1.concat(line2, line3);
```

8. Fill in the blanks in the madlib string by replacing them with the appropriate variable value. The following lines use the `replace()` method to replace the value of the variable with the correct location in the string:

Click here to view code image

```
madlib = madlib.replace("[place]", place);
madlib = madlib.replace("[animal]", animal);
madlib = madlib.replace("[action]", action);
```

9. Add the following command to write the fully processed madlib string to the browser:

Click here to view code image

```
document.write(madlib);
```

10. Open the page in the browser, and the string should be formatted as shown in Figure 6.1.

LISTING 6.1 string_manipulation.html Example of Combining Multiple Lines of Text into a Single String and Using `replace()` to Fill in Specifically Formatted Sections of the String with Variable Values

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>String Manipulation</title>
    <meta charset="utf-8" />
<script>
  function writeIt(){
    var line1 = "In a [place] a long time ago, ";
    var line2 = "there lived an [animal] that liked to ";
    var line3 = "[action] people.";
    var place = "crater";
    var animal = "elephant";
    var action = "smell";
    var madlib = line1.concat(line2, line3);
    madlib = madlib.replace("[place]", place);
    madlib = madlib.replace("[animal]", animal);
    madlib = madlib.replace("[action]", action);
    document.write(madlib);
  }
</script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

Array

The `Array` object provides a means of storing and handling a set of other objects. Arrays can store numbers, strings, or other JavaScript objects. You can use a couple of methods to create JavaScript arrays; for example, the following statements create three identical versions of the same array:

Click here to view code image

```
var arr = ["one", "two", "three"];
var arr2 = new Array();
arr2[0] = "one";
arr2[1] = "two";
arr3[2] = "three";
var arr3 = new Array();
arr3.push("one");
arr3.push("two");
arr3.push("three");
```

To get the number of elements in the array, you can use the `length` property of the `Array` object. For example:

```
var numOfItems = arr.length;
```

Arrays are a zero-based index, meaning that the first item is at index 0, and so on. You can access the array backward by subtracting using the `length` attribute. For example, in the following code, the value of variable first will be Monday and the value of variable last will be Friday:

Click here to view code image

```
var week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var first = week[0];
var last = week[week.length-1];
```

The `Array` object has several built-in functions that allow you to access and manipulate the array in various ways. Table 6.4 describes the method attached to the

`Array` object that enables you to manipulate the array contents.

Method	Description
`concat(arr1, arr2, ...)`	Returns a joined copy of the array and the arrays passed as arguments.
`indexOf(value)`	Returns the first index of the `value` in the array or –1 if the item is not found.
`join(separator)`	Joins all elements of an array separated by the `separator` into a single string. If no separator is specified, a comma is used.
`lastIndexOf(value)`	Returns the last index of the `value` in the array or –1 if the value is not found.
`pop()`	Removes the last element from the array and returns that element.
`push(item1, item2, ...)`	Adds one or more new elements to the end of an array, and returns the new length.
`reverse()`	Reverses the order of all elements in the array.
`shift()`	Removes the first element of an array and returns that element.
`slice(start, end)`	Returns the elements between the `start` and `end` index.
`sort(sortFunction)`	Sorts the elements of the array. The `sortFunction` is optional.
`splice(index, count, item1, item2...)`	At the `index` specified, `count` number items are removed, and then any optional items passed in as arguments are inserted at index.
`toString()`	Returns the string form of the array.
`unshift()`	Adds new elements to the beginning of an array and returns the new length.
`valueOf()`	Returns the primitive value of an `Array` object.

TABLE 6.4 `Array` Object Methods Used to Manipulate JavaScript Arrays

To get you started on using the functionality provided in the `Array` object, the following sections describe some of the common tasks that can be done using `Array` object methods.

Combining Arrays

You can combine arrays the same way that you combine `String` objects, using + statements or using the `concat()` method. In the following code, `arr3` ends up being the same as `arr4`:

Click here to view code image

```
var arr1 = [1,2,3];
var arr2 = ["three", "four", "five"]
var arr3 = arr1 + arr2;
var arr4 = arr1.concat(arr2);
```

Tip

You can combine an array of numbers and an array of strings. Each item in the array will keep its own object type. However, as you use the items in the array, you need to keep track of arrays that have more than one data type so that you do not run into problems.

Iterating Through Arrays

You can iterate through an array using a `for` or a `for/in` loop. The following code illustrates iterating through each item in the array using each method:

Click here to view code image

```
var week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var i=0; i<week.length; i++){
  document.write("<li>" + week[i] + "</li>");
}
for (dayIndex in week){
  document.write("<li>" + week[dayIndex] + "</li>");
}
```

Converting an Array into a String

A useful feature of `Array` objects is the capability to combine the elements of a string to make a `String` object separated by a specific separator using the `join()` method. For example, the following code results in the time components being joined together into the format 12:10:36:

Click here to view code image

```
var timeArr = [12,10,36];
var timeStr = timeArr.join(":");
```

Checking to See Whether an Array Contains an Item

Often, you will need to check to see whether an array contains a certain item. You can do this by using the `indexOf()` method. If the item is not found in the list, a –1 will be returned. The following function writes a message to the browser if an item is in the week array:

Click here to view code image

```
function message(day){
  var week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
  if (week.indexOf(day) != -1){
    document.write("Happy " + day);
  }
}
```

Adding Items to and Removing Items from Arrays

You can use several methods to add items or remove them from `Array` objects using the various built-in methods. Table 6.5 gives you some ideas on the various methods used in this book.

Statement	Value of x	Value of arr
`var arr = [1,2,3,4,5];var x = 0;`	`0`	`1,2,3,4,5`
`x = arr.unshift("zero");`	`6 (length)`	`zero,1,2,3,4,5`
`x = arr.push(6,7,8);`	`9 (length)`	`zero,1,2,3,4,5,6,7,8`
`x = arr.shift();`	`zero`	`1,2,3,4,5,6,7,8`
`x = arr.pop()`	`8`	`1,2,3,4,5,6,7`
`x = arr.splice(3,3,"four","five","six")`	`4,5,6`	`1,2,3,four,five,six,7`
`x = arr.splice(3,1)`	`four`	`1,2,3,five,six,7`
`x = arr.splice(3)`	`five,six,7`	`1,2,3`

TABLE 6.5 `Array` Object Methods Used to Add or Remove Elements from Arrays

Try it Yourself: Creating and Manipulating Arrays

In this exercise, you learn the creation and manipulation of arrays using various methods attached to the `Array` object. Use the following steps to build the file in Listing 6.2:

1. Create a source file named array_manipulation.html in the lesson06 folder.
2. Add the usual basic elements (html, head, body).
3. Inside the `<head>` element, add a `<script>` element with a function named `writeIt()` and link it to the `onload` event of the `<body>` element, as shown next:

Click here to view code image

```
<script>
  function writeIt(){
...
  }
</script>
  </head>
  <body onload="writeIt()">
```

4. Add the following JavaScript function that accepts a message and an array argument as `msg` and `arr`. The array is converted to a string using the `join()` method. The method also writes the message and joined array string to the browser with some HTML formatting:

Click here to view code image

```
07   function writeArray(msg, arr){
08     var arrString = arr.join(" | ");
09     document.write("<b>"+ msg + ": </b>" + arrString + "<br><br>");
10   }
```

5. Add the following lines of code to the `writeIt()` function to create a couple of arrays and call the `writeArray()` function to write them to the browser. The `weekDays` array is created by the declaration; the `weekEnd` array is generated by creating a blank array and then pushing items into it:

Click here to view code image

```
12     var weekDays = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday"];
13     writeArray("Week Days", weekDays);
14     var weekEnd = new Array();
15     weekEnd.push("Saturday");
16     weekEnd.push("Sunday");
17     writeArray("Weekend", weekEnd);
```

6. Use the following lines to create a new array with the full week array by combining the `weekEnd`. In line 18, the `concat([])` is a way of creating a copy of `weekDays`. We want Sunday at the first and Saturday at the end, so we use `unshift()` to push Sunday on to the front of the array and `push()` to append Saturday to the end:

Click here to view code image

```
18     var week = weekDays.concat([]);
19     week.unshift(weekEnd[1]);
20     week.push(weekEnd[0]);
21     writeArray("Week", week);
```

7. Create a `midWeek` array using the `slice()` function, as follows:

Click here to view code image

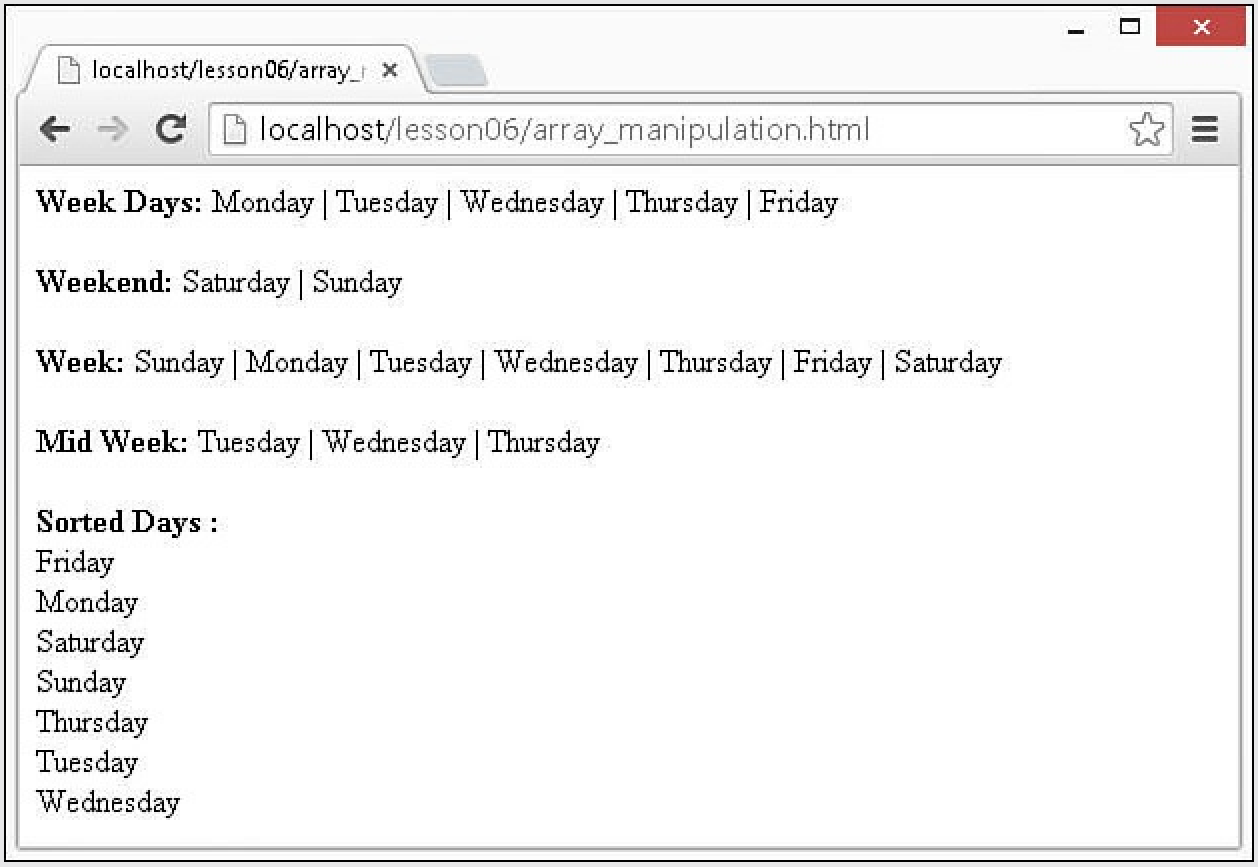
```
22     var midWeek = week.slice(2,5);
23     writeArray("Mid Week", midWeek);
```

8. Use the following code to create a sorted version of the full week and then iterate through each item in the `sortedWeek` array and write them out to the browser:

Click here to view code image

```
    var sortedWeek = week.sort();
    document.write("<b>Sorted Days :</b> <br>");
    for (dayIndex in sortedWeek){
      document.write(sortedWeek[dayIndex] + "<br>");
    }
```

9. Open the page in the browser, and the output of the arrays that you created should be displayed, as shown in Figure 6.2.


localhost/lesson06/array_
localhost/lesson06/array_manipulation.html

```
Week Days: Monday | Tuesday | Wednesday | Thursday | Friday

Weekend: Saturday | Sunday

Week: Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday

Mid Week: Tuesday | Wednesday | Thursday

Sorted Days :
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
```

FIGURE 6.2 Output of Listing 6.2.

LISTING 6.2 array_manipulation.html Example of Creating and Manipulating `Array` **Objects in JavaScript**

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Array Manipulation</title>
    <meta charset="utf-8" />
<script>
  function writeArray(msg, arr){
    var arrString = arr.join(" | ");
    document.write("<b>"+ msg + ": </b>" + arrString + "<br><br>");
  }
  function writeIt(){
    var weekDays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
    writeArray("Week Days", weekDays);
    var weekEnd = new Array();
    weekEnd.push("Saturday");
    weekEnd.push("Sunday");
    writeArray("Weekend", weekEnd);
    var week = weekDays.concat([]);
    week.unshift(weekEnd[1]);
    week.push(weekEnd[0]);
    writeArray("Week", week);
    var midWeek = week.slice(2,5);
    writeArray("Mid Week", midWeek);
    var sortedWeek = week.sort();
    document.write("<b>Sorted Days :</b> <br>");
    for (dayIndex in sortedWeek){
      document.write(sortedWeek[dayIndex] + "<br>");
    }
  }
</script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

Date

The Date object provides access to the current time on the browser's system. This can be useful in a lot of ways, such as displaying time on the page, comparing time with the server via AJAX, timing certain events, and the like.

To get the current time on the system, use the following syntax:

```
var cDate = new Date();
document.write(cDate);
```

The string version of the date will be similar to the following:

Click here to view code image

```
Mon Dec 10 2012 17:30:27 GMT-0700 (Mountain Standard Time)
```

You can create a Date object using a string or a set of values based on the following syntax:

Click here to view code image

```
Date(year, month, day, lessons, minutes, seconds, milliseconds)
```

For example, the following will create the same Date object:

Click here to view code image

```
var d1 = Date("2012, 12, 12, 12, 12, 12, 00");
var d2 = Date("December 12, 2012 12:12:12");
```

JavaScript enables you to compare dates using the normal logical comparisons (<, >, ==, and so on). For example, you could use the following code to compare the current time on the browser with what has been sent from the server as a timestamp: "December 12, 2012 12:12:12":

Click here to view code image

```
var currentTime = new Date();
var serverTime = new Date("December 12, 2012 12:12:12");
if (currentTime>serverTime){
    alert("Mayans Wrong?");
}
```

Math

The `Math` object is really an interface to a mathematical library that provides a ton of time-saving functionality. The math library is much too large to go over in this book, but Table 6.6 introduces you to some of the more useful functions.

Property/Method	Description	Output
`Math.PI`	Returns PI	`3.14...`
`Math.E`	Returns Euler's number	`2.718...`
`Math.LN10`	Returns the natural logarithm	`2.302...`
`Math.floor(5/2)`	Round down always	`2`
`Math.ceil(5/2)`	Round up always	`3`
`Math.round(5/2)`	Round up >= .5	`3`
`Math.abs(1-10)`	Absolute value	`9`
`Math.exp(5)`	Returns e^x	`148.4131...`
`math.pow(2,16)`	Returns x^y	`65536`
`math.sqrt(16)`	Returns square root	`4`
`Math.floor((Math.random()*100)+1)`	Random number	`0<?<100`
`Math.min(2,8,7,3,1)`	Minimum of set	`1`
`Math.max(2,8,7,3,1)`	Maximum of set	`8`
`Math.sin(1)`	Returns sine	`0.84147...`
`Math.cos(1)`	Returns cosine	`0.54030...`

TABLE 6.6 Using the Math Object in JavaScript

RegExp

When dynamically processing user input or even data coming back from the web server, an important tool is regular expressions. Regular expressions allow you to quickly match patterns in text and then act on those patterns.

JavaScript enables you to create a `RegExp` object that can be used to match patterns in strings using the following syntax:

Click here to view code image

```
var re = new RegExp(pattern,modifiers);
```

Or, more simply:

```
var re =/pattern/modifiers;
```

The pattern is a standard regular expression pattern, and the modifiers define the scope to apply the expression. Following is a list of the available modifiers in JavaScript:

- **i**—Perform matching that is not case sensitive.
- **g**—Perform a global match on all instances rather than just the first.
- **m**—Perform multiline match rather than stopping on the first `LF/CR` character.

The following shows an example of using a regular expression with a string `replace()` function to do a search for “yourself” that is not case sensitive and replace it with “Your Friends”:

Click here to view code image

```
var myStr = "Teach Yourself jQuery & JavaScript in 24 Lessons";
var re = /yourself/i;
var newStr = myStr.replace(re, "Your Friends");
```

The value of `newStr`:

Click here to view code image

```
Teach Your Friends jQuery & JavaScript in 24 Lesson
```

If you are not familiar with regular expressions, consider doing some research into it. There are some good books on the topic and several resources on the Web.

Creating Custom-Defined Objects

As you have seen so far, using the built-in JavaScript objects has several advantages. As you begin to write code that uses more and more data, you will find that you want to build your own custom objects with specific properties and methods. The following sections take you through the process of building custom JavaScript objects.

Defining JavaScript Objects

JavaScript objects can be defined using a couple of ways. The simplest is the on-the-fly method, meaning that you create a generic object and then add properties to it as you need it.

For example, to create a user object and assign a first and a last name, you use the following code:

```
var user = new Object();
user.first="Brad";
user.last="Dayley";
```

You could also accomplish the same effect through a direct assignment using the following syntax where the object is enclosed in `{}` brackets and the properties are defined using `property:value` syntax, as shown next:

Click here to view code image

```
var user = {'first':'Brad','last':'Dayley'};
```

These first two options work very well for simple objects that you do not need to reuse later. A better method is to enclose the object inside its own function block. This has the advantage of enabling you to keep all the code pertaining to the object local to the object itself. For example:

Click here to view code image

```
function User(first, last){
  this.first = first;
  this.last = last;
}
var user = new User("Brad", "Dayley");
```

The end result of these methods is essentially the same. You have an object with properties that that can be referenced using dot syntax, as shown next:

Click here to view code image

```
document.write(user.first + " " + user.last);
```

Adding Methods to JavaScript Objects

If there is code specific to a custom object, you want to attach that code as methods to the object itself. There are a couple of ways to do this depending on how the object was created.

The first way is to define a static function and then assign it as a property of the object. Following is an example of assigning a static function when defining objects on-the-fly:

Click here to view code image

```
var user = new Object();
```

```
user.first="Brad";
user.last="Dayley";
user.getFullName = makeFullName;
var user2 = {'first':'Brad', 'last':'Dayley',
             'getFullName':makeFullName};
function makeFullName(){
  return this.first + " " + this.last;
}
document.write(user.getFullName());
```

The function `makeFullName()` combines and returns the first and last properties of the object as a string. Notice that the function accesses the properties using the `this` keyword. `this` refers to the current instance of the object. Also notice that the assignment of the `makeFullName()` to `getFullName` omits the `()` parentheses; this sets `user.getFullName` equal to the function itself instead of the value it returns.

If you created the object using the better enclosed method, the function can be defined inside the object using the following syntax. Notice that `getFullName` is set to a newly defined function that returns the value in the same way that the static function `makeFullName()` did:

Click here to view code image

```
function User(first, last){
  this.first = first;
  this.last = last;
  this.getFullName = function(){
      return this.first + " " + this.last;
    };
}
```

Using a Prototyping Object Pattern

An even more advanced method of creating objects is using a prototyping pattern. The prototyping pattern is implemented by defining the functions inside the prototype attribute of the object instead of the object itself. The reason prototyping is better is that the functions defined in the prototype are created only once when the JavaScript is loaded, instead of each time a new object is created.

The following example shows the code necessary to implement the prototyping pattern. Notice that you define the object `UserP` and then you set `UserP.prototype` to include the `getFullName()` function. You can include as many functions in the `prototype` as you like. Each time a new object is created, those functions will be available:

Click here to view code image

```
function UserP(first, last){
  this.first = first;
```

```
    this.last = last;
  }
  UserP.prototype = {
    getFullName: function(){
        return this.first + " " + this.last;
      }
  };
```

To call the `getFullName()` function, you would then use something similar to the following:

[Click here to view code image]

```
  var me = UserP("Brad", "Dayley");
  var myFullName = me.getFullName();
```

Try it Yourself: Creating and Using Custom Objects

In this section, you use the prototyping pattern to create and use an array of custom JavaScript objects. Use the following steps to build the file in Listing 6.3:

1. Create a source file named custom_objects.html in the lesson06 folder.
2. Add the usual basic elements (html, head, body).
3. Inside the `<head>` element, add a `<script>` element with a function named `writeIt()` and link it to the `onload` event of the `<body>` element, as shown next:

[Click here to view code image]

```
<script>
  function writeIt(){
...
  }
</script>
  </head>
  <body onload="writeIt()">
```

4. Add the following function below `writeIt()`. This function will be used to create custom `Character` objects. The function accepts a `first` and `last` name, the `land` where the character is from, and the `race`:

[Click here to view code image]

```
  function Character(first, last, land, race){
    this.first = first;
    this.last = last;
    this.race = race;
    this.land = land;
  }
```

5. Add the following prototype definition that provides two functions, `getFullName()` and `getDetails()`:

Click here to view code image

```
33   Character.prototype = {
34     getFullName: function(){
35         return this.first + " " + this.last;
36     },
37     getDetails: function(){
38         return "is a " + this.race + " from the " + this.land;
39       }
40   };
```

6. Inside the `writeIt()` function, add the following lines of code that create the characters array and populate it with character objects. The objects are created using the `new` keyword:

Click here to view code image

```
08     var characters = new Array();
09     characters.push(new Character("John 117", "Master Chief",
10                                   "Earth", "Spartan"));
11     characters.push(new Character("vadam", "The Arbitor",
12                                   "High Charity", "Elite"));
13     characters.push(new Character("Gravemind", "", "Precursors",
14                                   "Flood"));
```

7. Add the following code to create the `sgtJohnson` object. Notice that we pass in only the `first` name. Lines 14 and 15 assign the additional `land` and `race` values. This provides an example of another method to assign values to objects:

Click here to view code image

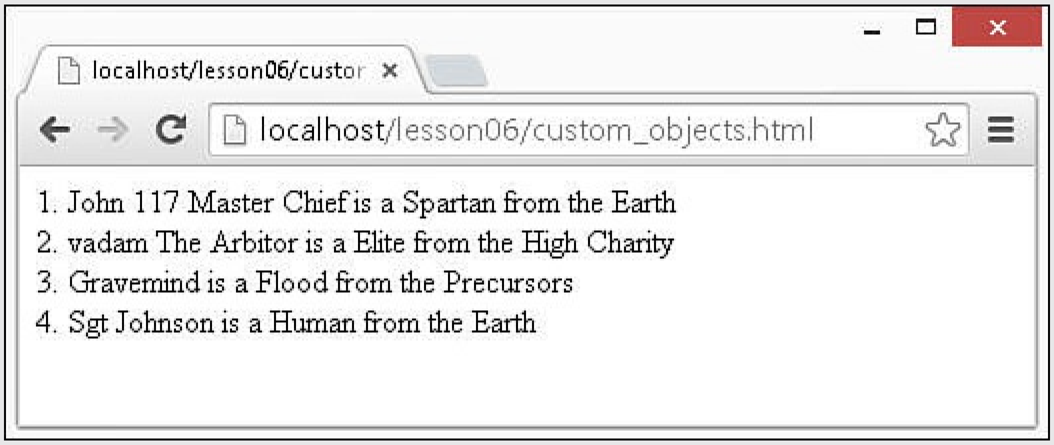
```
15     sgtJohnson = new Character("Sgt Johnson", "", "", "");
16     sgtJohnson.land = "Earth";
17     sgtJohnson.race = "Human";
18     characters.push(sgtJohnson);
```

8. Add the following `for` loop that iterates through each element of the characters array and writes information about the character to the browser. Line 19 assigns a new property value of `number` to the `Character` object. Notice that after it is assigned, we are able to access `character.number` to write it out as part of the string:

Click here to view code image

```
19     for (var i=0; i<characters.length; i++){
20       var character = characters[i];
21       character.number = i+1;
22       document.write(character.number + ". " +
23                      character.getFullName() + " " +
24                      character.getDetails() + "<br>");
25     }
```

9. Open the page in the browser, and the output of the objects that you created should be displayed, as shown in Figure 6.3.


localhost/lesson06/custom_objects.html

```
1. John 117 Master Chief is a Spartan from the Earth
2. vadam The Arbitor is a Elite from the High Charity
3. Gravemind is a Flood from the Precursors
4. Sgt Johnson is a Human from the Earth
```

FIGURE 6.3 Output of Listing 6.3.

LISTING 6.3 custom_objects.html Example of Creating and Manipulating Custom Objects in JavaScript

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Custom Objects</title>
    <meta charset="utf-8" />
<script>
  function writeIt(){
    var characters = new Array();
    characters.push(new Character("John 117", "Master Chief",
                                  "Earth", "Spartan"));
    characters.push(new Character("vadam", "The Arbitor",
                                  "High Charity", "Elite"));
    characters.push(new Character("Gravemind", "", "Precursors",
                                  "Flood"));
    sgtJohnson = new Character("Sgt Johnson", "", "", "");
    sgtJohnson.land = "Earth";
    sgtJohnson.race = "Human";
    characters.push(sgtJohnson);
    for (var i=0; i<characters.length; i++){
      var character = characters[i];
      character.number = i+1;
      document.write(character.number + ". " +
                     character.getFullName() + " " +
                     character.getDetails() + "<br>");
    }
  }
  function Character(first, last, land, race){
    this.first = first;
    this.last = last;
  this.race = race;
  this.land = land;
 }
 Character.prototype = {
  getFullName: function(){
    return this.first + " " + this.last;
  },
  getDetails: function(){
    return "is a " + this.race + " from the " + this.land;
   }
 };
</script>
 </head>
 <body onload="writeIt()">
 </body>
</html>
```

Summary

In this lesson, we discussed how JavaScript objects provide a way to create much cleaner and more efficient code. You learned how objects support attaching properties and methods to a single variable name.

You also were introduced to several built-in objects such as Array and String. The built-in objects provide properties and methods that provide quick functionality with little code—for example, using the Date object to get the system time, compare times, and create time-based output to the browser.

Finally, you got a chance to create and manipulate your own JavaScript objects with properties and methods. Creating custom objects will help you organize your scripts to be more efficient.

Q&A

Q. What is the difference between built-in, custom, DOM objects, and jQuery objects?

A. Only how they are defined. Built-in objects are automatically defined and available as part of the language sometimes; as with arrays or strings, they are automatically instantiated during an assignment statement. DOM objects are also built in to the JavaScript language. You can create an instance of them in JavaScript, but they are also created based on the tags in the HTML file. jQuery objects are defined and created inside the jQuery library. At the low level, they are all just JavaScript objects. Each has properties and methods that can be used to simplify JavaScript tasks.

Q. The new keyword creates a new instance of an object, but how do I get rid

of an instance of an object to free up memory and such?

A. JavaScript automatically cleans up memory for you when a variable goes out of scope. To clean up an instance of a JavaScript object, make it go out of scope. For example, if it is part of an array, you can us the `pop()` function to remove it, or, if you want all of an array of objects to be freed up, set the array variable to null.

Workshop

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

Quiz

1. What is the `new` keyword for?
2. How do you find the length of a JavaScript array?
3. True or false: The following statement creates a copy of the `Array` object named `myArr`:

   ```
   var newArr = myArr.concat([])
   ```

4. True or false: You cannot assign new properties to an existing JavaScript object.

Quiz Answers

1. The `new` keyword is used to create a new instance of an object.
2. The length of JavaScript `String` and `Array` objects can be found using the `length` property. For example:

   ```
   myArr.length
   ```

3. True.
4. False.

Exercises

1. Extend the code in Listing 6.1 to include a much more complete madlib. Add several attributes that should be included in the final string.
2. Use the code in Listing 6.2 as an outline to create and display arrays that contain the summer months, winter months, autumn months, and spring months. Then combine the arrays into a single array containing all the months of the year.
3. Extend Listing 6.3 to add additional characters. Also include additional attributes

such as weapon, age, or height.

Part II: Implementing jQuery and JavaScript in Web Pages

Lesson 7. Accessing DOM Elements Using JavaScript and jQuery Objects

What You'll Learn in This Lesson:

- ▶ The difference between jQuery and JavaScript DOM objects
- ▶ How to tell if an object is jQuery or DOM
- ▶ How to get a jQuery object from a DOM object
- ▶ How to get a DOM object from a jQuery object
- ▶ How to use jQuery selectors to quickly find DOM elements

The most important part of dynamic web development is the capability to access the DOM elements quickly and efficiently. JavaScript inherently provides functionality to access the DOM elements. It can be useful at times; however, this is the area where jQuery really stands out and you understand the need for jQuery on top of JavaScript.

In the following sections, you learn about the basic structure of jQuery and JavaScript DOM objects, how to determine whether an object is jQuery or DOM, and how to switch between them. The rest of the lesson discusses methods you can use to find and access the DOM elements using jQuery and JavaScript.

Understanding DOM Objects Versus jQuery Objects

You need to understand the difference between DOM objects and jQuery objects and how to manipulate and use both to be effective when creating dynamic web pages. In this section, you learn about these types of objects and how to navigate between them.

Introducing JavaScript DOM Objects

DOM objects are the objects that the web browser is using to render elements on the web page. Consequently, DOM objects have a great deal of functions and attributes. The advantage with working with DOM objects is that you have direct access to everything you need to manipulate the HTML element.

The disadvantage of DOM objects is that most of the attached functions and attributes are things that the browser needs and are not necessarily useful when you're working with JavaScript. That means that the DOM object has a lot of properties and methods that encumber your debugging environment, such as the Chrome development tools or other JavaScript consoles.

Table 7.1 provides a list of some of the important attributes and methods of DOM objects that you should be aware of. The list is not comprehensive; it is intended to give you an idea of things that you can access from a DOM object.

Attribute/Method	Description
`parentNode`	Because DOM objects are part of a tree, they always have a link to their direct parent.
`childNodes`	As with parent nodes, if the DOM object has children, this attribute allows you to access them as an array.
`click()`	This enables you to execute and/or change the event handler that gets called when the HTML element is clicked.
`innerHTML`	Access to the content text that is placed between the elements' starting and ending tags. For example, the following line of code changes the inner HTML of an object. Notice that you are able to include DOM tags in with the HTML text. These will be rendered with the page: `obj.innerHTML = "This is <span>SOME</span> text";`
`outerHTML`	Access to the full text, including tags of the HTML element.
`value`	Some elements such as options and inputs contain a `value` attribute.
`id`	Direct access to the `id` attribute of the HTML element.
`class`	Direct access to the `class` attribute of the HTML element.
`style`	Direct access to the CSS `style` of the element. For example, to set the color, you could use the following: `obj.style.color="red";`
`getAttribute(attribute)`	Gets the `attribute` value in the DOM object. For example, to get the `name` attribute of an object, use the following: `obj.getAttribute("name")`
`setAttribute(attribute, value)`	Sets the `value` of `attribute` in the DOM object. For example, to set the `href` attribute of an object, you could use the following: `obj.setAttribure("href", "http://dayleycreations.com");`
`appendChild(object)`	Appends a child DOM element to the object. For example, to add a new `<option>` element to a `<select>` object, you could use the following: `var o = new Option("New", 1, false, false);` `selectObj.appendObject(o);`

TABLE 7.1 Some of the More Commonly Used Attributes/Methods Attached to DOM Objects

Introducing jQuery Objects

jQuery objects are basically wrapper objects around a set of DOM elements. The jQuery objects are still JavaScript objects and provide access to the DOM elements—however, in a much different, much easier, and often much more effective way.

The most important thing to remember is that a jQuery object may represent a single DOM object, or it may represent a set of many DOM objects. So if you apply an operation on the jQuery object, it may apply to many DOM objects.

The biggest advantage to jQuery objects is how easy it is to search HTML elements in a web page. Another advantage is that the jQuery library wraps methods and attributes specifically to make it easier for JavaScript and jQuery developers to manipulate and work with different groups of objects. Calling a method on jQuery objects can apply to one or many DOM elements without the need to iterate through a list.

Table 7.2 provides a list of some of the important methods of jQuery objects that you should be aware of. The list is not comprehensive; it is intended to give you an idea of things that you can access from a jQuery object.

Method	Description
`html([newHTML])`	Gets the inner HTML text of the object or sets it to the optional `newHTML`.
`val([newValue])`	Gets the value of the object or sets it to the optional `newValue` argument if one is passed. For example, to set the value of an element, use the following: `$("#myInput").val("test");`
`attr(attribute, [value])`	Enables you to get the value of an attribute or set it to the optional value argument if one is passed. For example, to set the value of the `href` attribute, use the following: `$("#mainLink").attr("href", "www.dayleycreations.com");`
`addClass(class)`	Enables you to add a `class` attribute value that can result in new CSS rules being applied to the object.
`css(property, [value])`	Enables you to get or set a CSS `property value` for the jQuery object. For example, to set the `background-color` for all `<div>` elements, use the following: `$("div").css("backgorund-color", "yellow");`
`click([function])`	Enables you to get the `onclick` hander for the jQuery object or set a function definition for a new `onclick` handler.
`height([value])`	Gets or sets the height of the DOM elements. If the value is specified, the object's height is set; otherwise, the value of the object's height is returned.
`width([value])`	Gets or sets the width of the DOM elements.
`hide()`	Enables you to hide the DOM objects represented by the jQuery object.
`show()`	Enables you to unhide the DOM objects represented by the jQuery object.

TABLE 7.2 Some of the More Commonly Used Methods Attached to jQuery Objects

Determining Whether an Object Is DOM or jQuery

Occasionally, you may find yourself in a situation where you have an object and you are not sure whether it is a jQuery object, a DOM object, or some other JavaScript object. There is a simple way to tell the difference.

To tell whether an object is a jQuery object, use the following `if` syntax to check to see whether the object has the `jquery` attribute:

```
if( obj.jquery ) {
  ...
```

To tell whether an object is a DOM object, use the following `if` syntax to check whether the object has the `nodeType` attribute:

```
if( obj.nodeType ) {
  ...
```

Changing an Object from DOM to jQuery and Back

What do you do if you have a jQuery object but you want to use code intended for DOM objects, or vice versa? The answer is to convert the object to the other type.

The `.get()` method returns the JavaScript version of the jQuery element set in the form of an array of DOM objects. Use the following code to call the `get()` function that returns the DOM object of the HTML element represented by the jQuery object:

Click here to view code image

```
var domObj = jqueryObj.get();
```

Conversely, the `$()` or `jquery()` method creates a new jQuery object from JavaScript. Use the following code to execute the `$()` function to wrap the DOM object as a jQuery object:

```
var jqueryObj = $(domObj);
```

Accessing DOM Objects from JavaScript

To be able to manipulate HTML elements from JavaScript, you first need to gain access to the DOM object. You can use a few methods to accomplish that.

Finding DOM Objects by ID

The simplest is to find an HTML element using the value of the `id` attribute using the `getElementById(id)` function. `getElementById(id)` searches the DOM for an object with a matching `id` attribute.

For example, the following code searches for the HTML element with `id="container"`:

Click here to view code image

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

Finding DOM Objects by Class Name

You can also search for HTML elements by their `class` attribute using the `getElementsByClassName(class)`. This function returns a list of DOM objects with matching `class` attributes. You can then iterate over that list using a

JavaScript loop and apply changes to each DOM element.

For example, the following retrieves a list of HTML elements with `class="myClass"` and then iterates through them:

Click here to view code image

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

Finding DOM Objects by Tag Name

Another way to search for HTML elements is by their HTML tag, using the `getElementsByTagName(tag)`. This function returns a list of DOM objects with matching HTML tags. You can then iterate over that list using a JavaScript loop and apply changes to each DOM element.

For example, the following code retrieves a list of the `<div>` HTML elements and then iterates through them:

Click here to view code image

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

Try it Yourself: Using JavaScript to Access DOM Objects

In this example, you use JavaScript to find different elements and read values and write values to them. This is a basic example designed to give you a chance to use JavaScript to access DOM objects. The full scripts used in the example are shown in Listings 7.1, 7.2, and 7.3:

1. In Eclipse, create a source folder named lesson07. Inside the lesson07 folder, create a js and css folder.
2. Create a source file named dom_objects.html in the lesson07 folder.
3. Add the usual basic elements (html, head, body).
4. Inside the `<head>` element, add the following `<script>` and `<link>` elements, shown in Listing 7.1, which will be used to load the JavaScript and CSS rules that will be defined later:

Click here to view code image

```
06     <script type="text/javascript" src="js/dom_objects.js">
</script>
```

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

5. Add the following `<input>` elements to define a text box and a button. Notice that on the button, you need to add the `onclick` handler, which allows you to run your dynamic script when the button is clicked. The text input has the `id="textIn"`:

Click here to view code image

```
<input id="textIn" type="text"/>
<input type="button" onclick="textChange()" value="Update" /> <br>
```

6. Add the following `<span>` and `<p>` elements. For now, they are empty; the dynamic code will update them:

Click here to view code image

```
<span class="heading"></span>
<p id="p1"></p>
<span class="heading"></span>
<p id="p2"></p>
```

7. Create a file called lesson07/js/dom_objects.js.

8. Create a function called `textChange()`, as shown in Listing 7.2. This function will be called when the button is clicked.

9. Add the following variable definition to the `textChange()` function to find the text input by the id `"textIn"`:

Click here to view code image

```
var inElement = document.getElementById("textIn");
```

10. Add the following variable definition to get all the `<p>` elements by tag name:

Click here to view code image

```
var outElements = document.getElementsByTagName("p");
```

11. Add the following variable definition to get all the elements with the `class="heading"`:

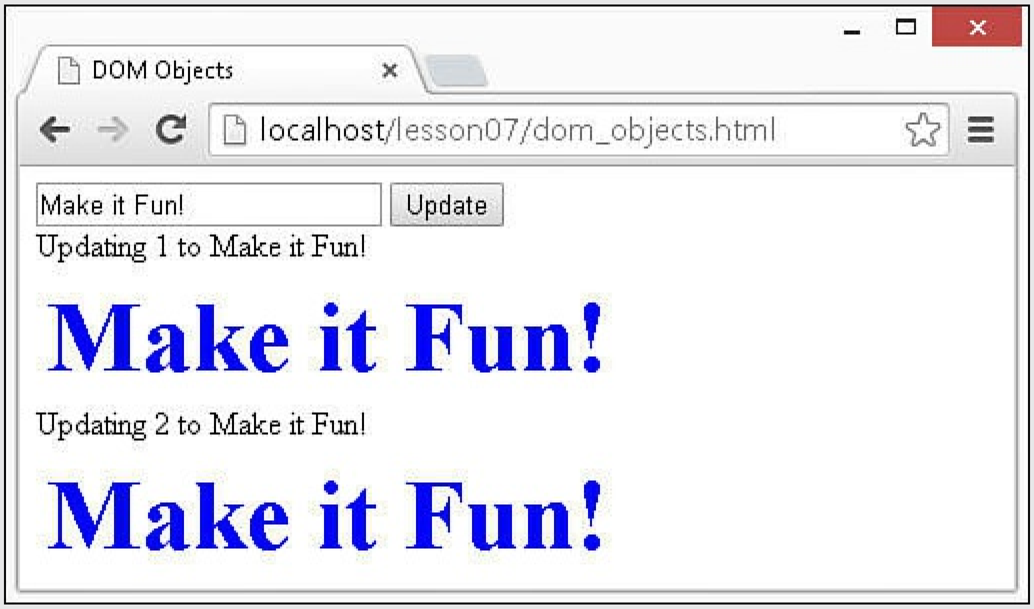
Click here to view code image

```
var headingElements = document.getElementsByClassName("heading");
```

12. Add the `for` loop shown in Listing 7.2 that iterates through the `outElements` that were found in the document. For each element, you set the `innerHTML` text to the `inElement.value`, which is what is typed into the text box.

13. Create a file called lesson07/css/dom_objects.css and add the code shown in Listing 7.3. This stylizes the out elements a bit.

14. Save the three files and open the dom_objects.html file in a web browser to check out the web page shown in Figure 7.1. Try typing in some text and clicking the Update button.



Updating 1 to Make it Fun!

Make it Fun!

Updating 2 to Make it Fun!

Make it Fun!

FIGURE 7.1 A simple JavaScript web page that allows the user to enter text and update components.

LISTING 7.1 dom_objects.html HTML File That Loads JavaScript and Attaches an Event Handler to a Button Element to Update the Page

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Objects</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="js/dom_objects.js"></script>
    <link rel="stylesheet" type="text/css" href="css/dom_objects.css">
  </head>
  <body>
    <input id="textIn" type="text"/>
    <input type="button" onclick="textChange()" value="Update" /><br>
    <span class="heading"></span>
    <p id="p1"></p>
    <span class="heading"></span>
    <p id="p2"></p>
  </body>
</html>
```

LISTING 7.2 dom_objects.js JavaScript File Contains a Function Showing Examples of Accessing Variables by `id`, `tag`, and `class` Attributes

[Click here to view code image]

```
function textChange(){
  var inElement = document.getElementById("textIn");
  var outElements = document.getElementsByTagName("p");
  var headingElements = document.getElementsByClassName("heading");
  for(var i=0; i<outElements.length; i++){
    var outItem = outElements[i];
    headingElements[i].innerHTML = "Updating " + (i+1) +
                                   " to " + inElement.value;
    outItem.innerHTML = inElement.value;
  }
}
```

LISTING 7.3 dom_objects.css CSS That Styles `<p>` Elements

```
p{
  font-weight:bold;
  font-size:50px;
  margin:5px;
  color:blue;
}
```

Using jQuery Selectors

Unlike JavaScript, jQuery enables you to find HTML elements in countless ways using selectors. Yes, just like CSS selectors. In fact, most of the jQuery selectors were taken from CSS, providing a more seamless transition between the two.

As you will find in upcoming sections, jQuery selectors make it very easy to select just about any group of HTML elements. Keep in mind, though, that jQuery selectors return jQuery objects, not DOM objects.

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

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

Caution

Several metacharacters are used in jQuery selector syntax. If you want to use any of the meta characters, such as `!"#$%&'()*+,./:;<=>?`

@[\]^`{|}~), as a part of a class/id/name, you will need to escape the character with \\ two backslashes. For example, if you have an element with id="my.element", you would use the selector $("#my\\.element").

As with CSS selectors, the best way to introduce you to jQuery selectors is to show you some examples. The following sections take you through some examples of different types of jQuery selectors. These sections only scratch the surface. You can go to the following location to review the selector documentation when you get a chance:

http://api.jquery.com/category/selectors/

Applying Basic Selectors

The most commonly used selectors are the basic ones. The basic selectors focus on the id attribute, class attribute, and tag name of HTML elements. Table 7.3 list some examples to show you how to define some of the basic selectors.

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

TABLE 7.3 Examples of Using Basic jQuery Selectors

Applying Attribute Selectors

Another way to use jQuery selectors is to select HTML elements by their attribute values. It can be a default attribute or one that you have added to the HTML elements. Attribute values are denoted in the selector syntax by being enclosed in `[]` brackets. Table 7.4 shows some of the ways that attribute selectors can be applied.

Selector Syntax/Example	**Description**

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

`$([attr*=value])`
`$("p[class*=Content")`
	Attribute `attr` contains `value`.
	Example: Selects all HTML elements with `"Content"` in the class name. For example, all of the following `<p>` elements would be selected:
	`<p class="leftContent">...</p>`
	`<p class="cenerContent">...</p>`
	`<p class="rightContent">...</p>`

`$("[attr^=value]")`
`$("img[src^='icons\\/']")`
	Attribute `attr` begins with value.
	Example: Selects all `<img>` elements whose `src` attribute starts with `"icons/"`. Notice that because the expression was not simple text, quotes were required around the value. Also notice that the / character had to be escaped with \\.

`$("[attr!=value]")`
`$("input[value!=`
`'default text']")`
	Attribute `attr` does not equal value.
	Example: Selects all the `<input>` elements where the value does not equal `"default text"` or that do not have a value attribute.

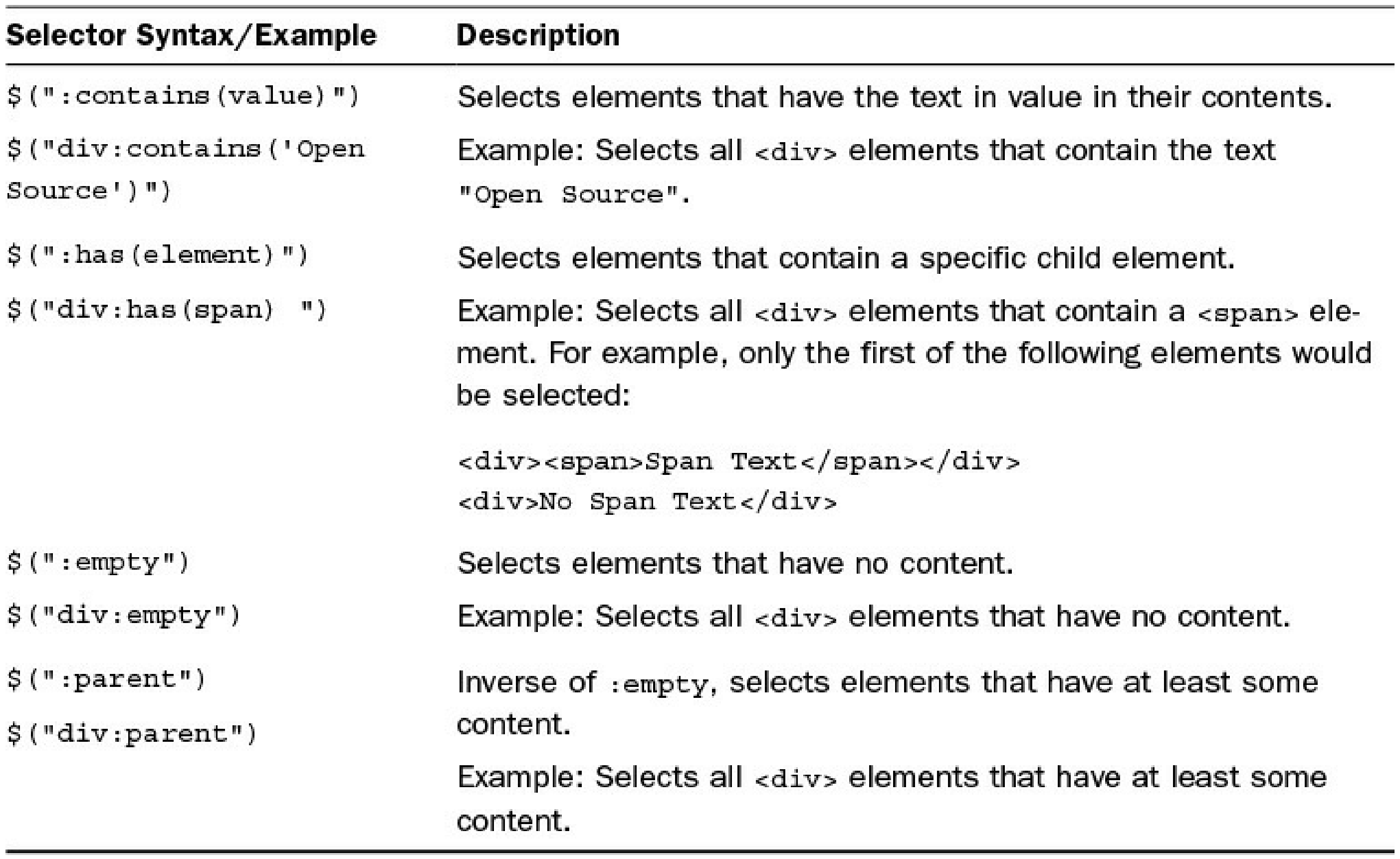
`$("[attr]")`
`$("p[id]")`
	Selects elements that have attribute `attr`.
	Example: Selects all `<p>` elements that have an `id` attribute.

`$("[attr][attr2$=value]")`
`$("p[id][class$=Menu]")`
	Selects elements that have attribute `attr` and have attribute `attr2` equal to `value`.
	Example: Selects all `<p>` elements that have an `id` attribute and have a `class` attribute that equals `"Menu"`. For example, only the top one of the following HTML elements would be selected:
	`<p id="Menu" class=" Menu ">...</p>`
	`<p id="boxA" class="Menu">...</p>`
	`<p id="Menu" class="boxA">...</p>`
	`<p class="Menu">...</p>`

TABLE 7.4 Examples of Using Attribute jQuery Selectors

Applying Content Selectors

Another set of useful jQuery selectors are the content filter selectors. These selectors allow you to select HTML elements based on the content inside the HTML element. Table 7.5 shows examples of using content selectors.



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

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

`$(":empty")`	Selects elements that have no content.
`$("div:empty")`	Example: Selects all `<div>` elements that have no content.

`$(":parent")`
`$("div:parent")`
	Inverse of `:empty`, selects elements that have at least some content.
	Example: Selects all `<div>` elements that have at least some content.

TABLE 7.5 Examples of Using Content jQuery Selectors

Applying Hierarchy Selectors

An important set of jQuery selectors are the hierarchy selectors. These selectors allow you to select HTML elements based on the DOM hierarchy. This enables you to write dynamic code that is more content aware by only selecting elements based on parents, children, or other elements around them in the DOM tree. Table 7.6 shows some examples of hierarchy selectors.

Selector Syntax/Example	Description

`$(" ancestor element")`
`$("div span")`
	Selects elements of a type that have an ancestor of type `ancestor` and match `element`.
	Example: Selects all `<span>` elements that have an ancestor that is a `<div>`. The `<div>` element does not need to be the immediate ancestor. For example, the following `<span>` element would still be selected:
	`<div><p>Some <span>Span Text</span></p></div>`

`$("parent > child")`
`$("div.menu > span ")`
	Selects elements with a specific parent type.
	Example: Selects all `<span>` elements that have an immediate parent element that is a `<div>` with `class="menu"`.

`$("prev + next")`
`$("label + input.textItem")`
	Selects elements immediately followed by a specific type of element.
	Example: Selects all `<label>` elements that are immediately followed by an `<input>` element that has `class="textItem"`.

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

TABLE 7.6 Examples of Using Hierarchy jQuery Selectors

Note

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

Applying Form Selectors

An extremely useful set of selectors when working with dynamic HTML forms are the form jQuery selectors. These selectors enable you to select elements in the form based on the state of the form element. Table 7.7 shows some examples of form selectors.

Selector Syntax/Example	Description

`$(":checked")`
`$("input:checked")`
	Selects elements with `checked` attribute true.
	Example: Selects all `<input>` elements that are currently in a `checked` state.

`$(":selected")`
`$("option:selected")`
	Selects elements with the `selected` attribute true.
	Example: Selects all `<option>` elements that are currently `selected`.

`$(":focus")` `$(":focus")`	Selects elements that are in focus in the form. Example: Selects all HTML elements that are currently in focus.

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

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

TABLE 7.7 Examples of Using Attribute jQuery Selectors

Applying Visibility Selectors

If you are using visibility to control the flow and interactions of your web page components, using the visibility jQuery selectors makes it simple to select the HTML elements that are hidden or visible. Table 7.8 shows some examples of visibility selectors.

Selector Syntax/Example	Description

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

`$(":hidden")`
`$("div:hidden ")`
	Selects hidden elements.
	Example: Selects all `<div>` elements that currently have the CSS property of `visibility:hidden` or `opacity:0`.

TABLE 7.8 Examples of Using Attribute jQuery Selectors

Applying Filtered Selectors

Often you will need to refine your jQuery selectors down to a more specific subset. One way to accomplish that is to use filtered selectors. Filtered selectors append a filter on the end of the selector statement that limits the results returned by the selector. Table 7.9 shows some examples of adding filters to selectors.

Selector Syntax/Example	Description

`$(":even")`
`$("tr:even")`
	Filters out all the odd indexed elements.
	Example: Selects all `<tr>` elements and then filters the results down to only the even-numbered items; for example, second, fourth, sixth, and so on.

`$(":odd")`
`$("li:odd")`
	Filters out all the even indexed elements.
	Example: Selects all `<li>` elements and then filters the results down to only the odd-numbered items; for example, first, third, fifth, and so on.

`$(":first")`
`$("div:first")`
	Filters out everything but the first element.
	Example: Selects only the first `<div>` element encountered.

`$(":last")`
`$("div:last")`
	Filters out all but the last element.
	Example: Selects only the last `<div>` element encountered.

`$(":header")` `$(":header")`	Selects elements that are header types, such as `<h1>`, `<h2>`, `<h3 >`, and so on. Example: Selects all header elements.

`$(":eq(index)")`
`$("div:eq(5)")`
	Selects the element at a specific zero-based index.
	Example: Selects the sixth `<div>` element encountered. The reason that the sixth element and not the fifth is selected is that the index is zero based, so 0 would be the first.

`$(":gt(index)")`
`$("li:gt(1)")`
	Greater than; filters the list to include only elements after a specific zero-based index.
	Example: Selects every `<li>` element after the second one encountered. Again, this index is zero based.

`$(":lt(index)")`
`$("li:lt(2)")`
	Less than; filters the list to include only elements before a specific zero-based index.
	Example: Selects only the first and second `<li>` elements encountered. Again, this index is zero based.

`$(":animated")`	Selects elements that are currently being animated. Example: Selects all elements that are currently being animated.

TABLE 7.9 Examples of Using Filtered jQuery Selectors

Try it Yourself: Using jQuery to Access DOM Objects

In this example, you use jQuery to find different elements and read values and write values to them. This is another basic example designed to help solidify the concepts of how jQuery selectors enable you to find and access DOM elements. You see some more advanced examples later in this lesson. The full scripts used in the example are shown in Listings 7.4, 7.5, and 7.6:

1. In the lesson07 folder, create the source files named jquery_selectors.html, js/jquery_selectors.js, and css/jquery_selectors.css.
2. Open jquery_selectors.html and add the usual basic elements (html, head, body).
3. Inside the `<head>` element, add the following `<script>` and `<link>` elements, shown in Listing 7.4, that will be used to load the jQuery, JavaScript, and CSS:

Click here to view code image

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
<script type="text/javascript" src="js/jquery_selectors.js"></script>
<link rel="stylesheet" type="text/css" href="css/jquery_selectors.css">
```

4. Now add the following `<span>` elements that will be styled as buttons. Each

<span> element contains a different onclick handler that will allow you to run your dynamic script when it is clicked:

Click here to view code image

```
11     <span onclick="setEven()">Even</span>
12     <span onclick="setOdd()">Odd</span>
13     <span onclick="setFirst4()">First 4</span>
```

5. Add the <p> and <ul> elements shown in Listing 7.4 to add a list of ancient gods to the page.

6. Open the file jquery_selectors.css and add the contents of Listing 7.6 that style the <span> elements as buttons and the <p> element as a list header.

7. Open the jquery_selectors.js files so that you can add the event handlers for the <span> elements.

8. Add a JavaScript function named setEven() to act as an event handler.

9. Add the following line that uses jQuery to select all <li> and <span> elements and clears the CSS font-weight value:

Click here to view code image

```
02   $("li, span").css("font-weight","");
```

10. Add the following lines that define a variable named $evenItems to the results of the selector that selects all <li> elements and then filters the list to only those with an even index. Line 3 uses the $evenItems variable to set the CSS font-weight property of those items to bold:

Click here to view code image

```
03   var $evenItems = $("li:even");
04   $evenItems.css("font-weight","bold");
```

11. Add the following line that uses jQuery to select all <span> elements that contain "Even" in their contents and sets the CSS font-weight property to bold:

Click here to view code image

```
05   $("span:contains(Even)").css("font-weight","bold");
```

12. Add the following line that uses jQuery to select the elements with class="label" to select the <p> element. The .html("Even") part of the statement changes the innerHTML property of the selection to "Even", thus changing the heading:

Click here to view code image

```
06   $(".label").html("Even");
```

13. Add the other two event handlers, `setOdd()` and `setFirst4()`, shown in Listing 7.5, which basically do the same thing, but for a different set of list items.

14. Save the three files and open the jquery_selectors.html file in a web browser to check out the web page shown in Figure 7.2. Try clicking the different buttons and watch the button, heading, and list items change.

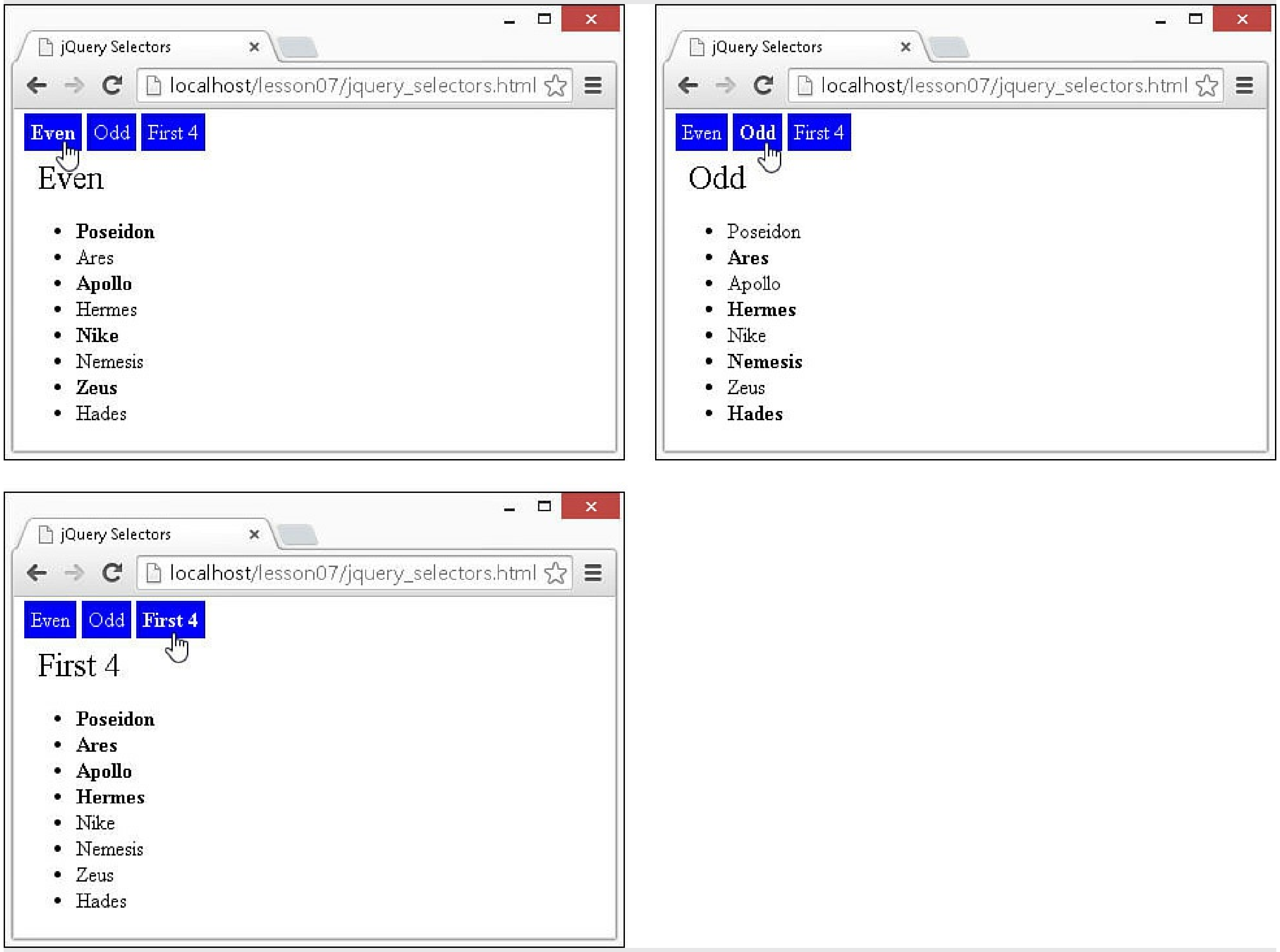


FIGURE 7.2 A simple JavaScript web page that uses jQuery to dynamically change page elements based on user interaction.

LISTING 7.4 jquery_selectors.html HTML File That Loads jQuery and JavaScript and Attaches Event Handlers Elements to Provide User Interaction

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery Selectors</title>
    <meta charset="utf-8" />
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/jquery_selectors.js">
</script>
    <link rel="stylesheet" type="text/css"
href="css/jquery_selectors.css">
  </head>
  <body>
    <span onclick="setEven()">Even</span>
    <span onclick="setOdd()">Odd</span>
    <span onclick="setFirst4()">First 4</span>
    <p class="label">Planets</p>
    <ul>
      <li>Poseidon</li>
      <li>Ares</li>
      <li>Apollo</li>
      <li>Hermes</li>
      <li>Nike</li>
      <li>Nemesis</li>
      <li>Zeus</li>
      <li>Hades</li>
    </ul>
  </body>
</html>
```

LISTING 7.5 jquery_selectors.js JavaScript File Containing Event Handler Functions That Use jQuery in Various Ways to Select and Alter Page Elements

Click here to view code image

```
function setEven(){
  $("li, span").css("font-weight","");
  var $evenItems = $("li:even");
  $evenItems.css("font-weight","bold");
  $("span:contains(Even)").css("font-weight","bold");
  $(".label").html("Even");
}
function setOdd(){
  $("li, span").css("font-weight","");
  var $oddItems = $("li:odd");
  $oddItems.css("font-weight","bold");
  $("span:contains(Odd)").css("font-weight","bold");
  $(".label").html("Odd");
}
function setFirst4(){
  $("li, span").css("font-weight","");
  var $first4 = $("li:lt(4)");
  $first4.css("font-weight","bold");
  $("span:contains('First 4')").css("font-weight","bold");
  $(".label").html("First 4");
}
```

LISTING 7.6 jquery_selectors.css CSS That Styles <span> Elements and Elements with class="label"

```
span{
  padding:2px;
  border:3px ridge blue;
  color:white;
  background:blue;
  cursor:pointer;
}
.label{
  font-size:25px;
  margin:10px;
}
```

Summary

In this lesson, you learned about using jQuery and JavaScript objects to find and access DOM elements. This is the most critical piece of dynamic programming because you must be able to provide efficient access to the DOM elements to be able to manipulate them dynamically. You learned the basic syntax and structure of JavaScript DOM objects, as well as a few of the methods and attributes attached to those objects.

Q&A

Q. When should I use the getObjectById() rather than $("#id")?

A. It's likely that you will rarely use `getObjectById()`. The only time you might use it is when you don't want to take the time to link the jQuery library to your HTML docs. jQuery is so much more extensible.

Workshop

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

Quiz

1. How to would you convert a DOM object named `myDiv` into a jQuery object?
2. True or false: A jQuery selector returns a list of DOM objects.
3. How can you tell if an object is a jQuery object?

4. What jQuery selector would you use if you wanted all elements with `class="heading"`?
5. What jQuery selector would you use to get all `<p>` elements that are children of `<div>` elements?
6. How can you get a JavaScript array of the DOM elements represented in a jQuery object?

Quiz Answers

1. Use `$(myDiv)` to create a new jQuery object with the DOM element as the only item in the set.
2. False. A jQuery selector returns a jQuery object that contains a set of DOM elements.
3. Test to see whether it has the `jquery` property set.
4. `$(".heading")`
5. `$(div p)`
6. Use the `get()` method on the jQuery object.

Exercise

1. Add an additional button to the example in jquery_selectors.html that will select only the first and last items in the list. You will need to add the HTML and JavaScript code necessary to do so.

Lesson 8. Navigating and Manipulating jQuery Objects and DOM Elements with jQuery

What You'll Learn in This Lesson:

- Chaining jQuery operations together for efficiency
- Ways to filter the DOM elements in a jQuery object
- Methods to use jQuery objects to traverse the DOM
- Iterating through each element in the jQuery object set

jQuery selectors return a jQuery object that represents zero or more elements that match the selector definition. Simple selectors are great for a lot of things. However, as your web pages become more complex, you will find that the selectors do not do everything that you need.

jQuery objects provide additional functionality and enhance the selector results by allowing you to easily refine the list of DOM elements represented, navigate the DOM tree to find other elements, and manipulate the values of the HTML elements. The following sections cover how to chain jQuery operations together to efficiently find, filter, and navigate around the DOM elements in the web page.

Chaining jQuery Object Operations

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

Think of the results of the chained jQuery operations as a stack of jQuery objects, with each object representing a set of DOM elements. Each operation in the chain will place a jQuery object onto the stack, but the current operation will be applied only to the top jQuery.

To help illustrate this, consider the following statements. The code first finds the `<div>` element with `id="content"` and then finds the first `<p>` element inside and changes the `font-weight` to `bold`. Then it finds the `<span>` elements inside the `<p>` and sets the `color` to `red`:

Click here to view code image

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

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

Click here to view code image

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

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

Filtering the jQuery Object Results

jQuery objects provide a good set of methods that allow you to alter the DOM objects represented in the query. Reducing the results is helpful when you are trying to pinpoint a specific set of elements within a jQuery selector result. Table 8.1 provides some examples of chaining jQuery selectors.

Method/Example	Description

`.eq(index)`
`$("div").eq(1);`
	Selects the element at the zero-based index in the set.
	Example: Returns the second `<div>` element.

`.filter(selector or element or function(index) or index or object)`
`$("option").filter(`
` function (index) {`
`  return (this.value>5);`
`});`
	Reduces the set to match the filter. The filter can be a selector, a specific DOM element, a jQuery object, or a function that tests each element and returns true if it should be included.
	Example: Executes the function on each item in the current set and includes only those items where the `value > 5`. The `this` keyword refers to the current DOM object during each loop through the set.

`.first()`
`$("li").first()`
	Selects the first element in the set.
	Example: Reduces to just the first `<li>` element.

`.last()`
`$("p").last()`
	Selects the last element in the set.
	Example: Reduces to just the last `<p>` element.

`.has(selector or element)`
`$("div").has("p")`
	Reduces the set to those elements that have descendent elements that match the selector or contain the specified DOM element.
	Example: Reduces to only `<div>` elements that have `<p>` children elements.

`.not(selector or elements or function(index) or object)`
`$(".menu").not("span")`
	Reduces the set to match the filter. The filter can be a selector, one or more specific DOM elements, a jQuery object, or a function that tests each element and returns true if it should be included.
	Example: Selects elements where `class="menu"` but are not `<span>` elements.

`.slice(start, [end])`
`$("tr").slice(2,5)`
	Remove elements before index start and after index end. The indexes are zero based.
	Example: Selects the `<tr>` elements with index `2`, `3`, and `4`.

TABLE 8.1 jQuery Object Methods with Examples That Filter the DOM Elements Represented

Tip

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

Traversing the DOM Using jQuery Objects

Another important set of methods attached to the jQuery object are the DOM traversing methods. DOM traversal enables you to select elements based on their relationship to other elements.

The DOM is sometimes referred to as the DOM tree because it is organized in a tree structure, with the document as the root and nodes that can have both parents, siblings, and children. To visualize this better, check out the following HTML code:

```
<body>
  <div>
    <p>Paragraph 1</p>
    <p>Paragraph 2</p>
    <ul>
       <li>Item 1</li>
        <li>Item 2</li>
      </ul>
 </div>
</body>
```

The `<p>` elements and `<ul>` element are siblings to each other, and they are all children of the `<div>` element. The `<div>` element is the parent of the `<p>` and `<ul>` elements, and the `<ul>` element is the parent of the `<li>` elements, and so forth.

jQuery DOM traversing methods enable you to move from one layer in the DOM to another to select elements—for example, if you want to access all `<p>` elements that are children of `<div>` elements or if you want to find a `<label>` element that is a sibling of an `<input>` element.

jQuery objects provide an incredible set of methods that allow you to traverse the DOM in almost innumerable ways by allowing you to use the current selection of DOM elements in the jQuery object to select other sets of DOM elements in the tree. Table 8.2 lists the methods that you can use to traverse the DOM.

Method/Example	Description

`.children([selector])`
`$("div").children("p")`
	Returns a jQuery object representing the children of the elements represented by the current object. You can specify an optional selector that limits the results to include only children that match the selector.
	Example: Selects the `<p>` elements that are direct `children` of all `<div>` elements.

`.closest(selector,[context] or object or element)`
`$("p.menu").closest("div")`
	Returns a jQuery object representing the first element that matches the argument that is passed in. The argument can be a selector, a selector with context of where to begin searching, a jQuery object, or a DOM object. The search begins at the current set of elements and then searches ancestors.
	Example: Selects the closest `<div>` ancestor for `<p>` elements that have `class="menu"`.

`.contents()`
`$("div").contents()`
	Returns a jQuery object representing the immediate children of the current set of elements.
	Example: Selects all the immediate child elements in `<div>` elements.

`.find(selector or object or element)`
`$("table").find("span")`
	Returns a jQuery object representing descendants of the current set that match the argument supplied. The argument can be a selector, a jQuery object to match elements against, and an element tag name.
	Example: Selects all `<span>` elements contained somewhere in `<table>` elements.

`.next([selector])`
`$("p#title").next("p")`
	Returns a jQuery object representing the next sibling of each element in the current set. If the optional selector is provided, the next sibling is added only if it matches the selector.

	Example: Finds the `<p>` element with `id="title"` and selects the very next `<p>` element that is a sibling.
`.nextAll([selector])` `$("p:first").nextAll()`	Returns a jQuery object representing all the following sibling objects of each element in the current set. Also accepts an optional selector. Example: Selects the first `<p>` element that it finds and then selects all the `<p>` siblings to that element.
`.nextUntil([selector] or [element] [,filter])` `$("p:first").nextUntil("ul")`	Returns a jQuery object representing all the sibling objects after each element in the current set, up until an object matching the element or selector argument is encountered. The first argument can be a DOM object or a selector. The second optional argument is a filter selector to limit the items returned in the results. Example: Selects the first `<p>` element that it finds and then selects all the siblings until it finds a `<ul>` element.
`.offsetParent()` `$("#notify").offsetParent()`	Returns a jQuery object representing the closest ancestor element that has a CSS position attribute of relative, absolute, or fixed. This allows you to get the element used for positioning. Example: Finds the element with `id="notify"` and then selects the ancestor that is used to position that element.

`.parent([selector])` `$("div#menu").parent()`	Returns a jQuery object representing the immediate parent objects of each of the elements represented in the current set. An optional selector argument allows you to limit the results to those parents matching the selector. Example: Selects the `<div>` element with `id="menu"` and then finds its immediate parent.
`.parents([selector])` `$("#data").parents("div")`	Returns a jQuery object representing the ancestors of each of the elements represented in the current set. An optional selector argument allows you to limit the results to those parents matching the selector. Example: Finds the element with `id="data"` and then returns a set with all `<div>` ancestors.
`.parentsUntil([selector] or [element] [, filter])`	Returns a jQuery object representing the ancestors of each of the elements represented in the current set, up until an object matching the element or selector argument is encountered. The first argument can be an element tag name or a selector. The second optional argument is a filter selector to limit the items returned in the results.
`.prev()` `$("p#footer").prev("p")`	Returns a jQuery object representing the previous sibling of each element in the current set. If the optional selector is provided, the previous sibling is added only if it matches the selector. Example: Finds the `<p>` element with `id="footer"` and selects the previous `<p>` element that is a sibling.
`.prevAll()` `$("div#footer").prevAll("div")`	Returns a jQuery object representing all the previous sibling objects of each element in the current set. Also accepts an optional selector.

	Example: Selects the `<div>` element with `id="footer"` and then selects all the `<div>` siblings prior to that element.
`. prevUntil([selector] or [element] [,filter])`	Returns a jQuery object representing all the sibling objects that come before each element in the current set, up until an object matching the element or selector argument is encountered. The first argument can be a DOM object or a selector. The second optional argument is a filter selector to limit the items returned in the results.
`.siblings([selector])` `$(".menu").siblings("span")`	Returns a jQuery object representing all the sibling objects for each element in the current set. An optional selector argument allows you to limit the results to those siblings matching the selector. Example: Selects all `<span>` elements that are siblings to elements with `class="menu"`.

TABLE 8.2 jQuery Object Methods with Examples That Allow You to Traverse the DOM to Select Elements

Looking at Some Additional jQuery Object Methods

When filtering the jQuery object or using it to traverse the DOM, you should be aware of some additional methods. Table 8.3 lists a set of methods that you can use in conjunction with filtering and traversing elements, to iterate through the DOM elements in the jQuery object, add additional elements to the set, end filtering, and test items. You will find that you'll use these methods more and more frequently as your jQuery code becomes more natural to you.

Method/Example	Description
`.add(selector,[context] or elements or html or object)` `$("div.left").add("div.right")`	Returns a new jQuery with additional elements. The arguments to `.add()` can be another jQuery selector with optional context to start from, another set of elements, an HTML fragment, or a jQuery object. Example: Selects all `<div>` elements with `class="left"` and then adds all `<div>` elements with `class="right"`.
`.andSelf()` `$("#title").nextAll().andSelf()`	Adds the previous set of elements on the stack to the current set so that you can apply operations on both. Selects the element with `id="title"`, gets all the siblings after it, and finally returns an object that includes all the siblings and the element itself.
`.each(function(index, Element))` `$("span").each(function(i){ $(this).css('width','300px'); })`	Iterates through the current set of elements and executes the specified function by passing in the index and DOM object. `.each()` returns the same jQuery object that it operated on. Example: Sets the `width` of all `<span>` elements to 300 pixels.
`.end()` `$("p").first().css("font-width", "bold").end().eq(1)`	Ends the current filtering operations in chained jQuery statements and returns to the jQuery object in the stack from the previous state. Example: Selects all `<p>` elements; then selects the first one and sets the font to bold. Then the `end()` method return to the jQuery object with all `<p>` elements, and you are able to select the second one in that set.

`.is(selector or index or object or element)`
`$(".menu").each(function(i){`
` $v = $(this);`
` if($v.is("span")){`
`    $v.css('color','red');`
`  }`
`})`
	Check the current set of objects against a selector, function, jQuery object, or DOM object. If the elements match, true is returned; otherwise, false is returned.
	Example: Selects all elements with `class="menu"`, then iterates through the elements, and if it is a `<span>` element, it changes the `color` CSS property to `red`.

`.map(function(index, element))`
`$(":selected").map(function(i) {`
`    return $(this).val();`
` }).get();`
	Iterates through the current set of elements and executes a function on each item. The `.map()` function returns a jQuery object that represents a set of the results from each time the map function was executed.
	Example: Gets all the selected elements, then uses `map` to iterate through them and returns a JavaScript array of the `value` of each of the selected elements using `.get()`.

TABLE 8.3 A Few Additional Methods and Examples That Allow You to Work with jQuery Objects

Note

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

Using .each()

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

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

The following snippet of code illustrates using `.each()`. It iterates through all paragraph elements and sets the content, including the index number of the element:

Click here to view code image

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

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

Using .map()

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

The following snippet of code illustrates using `.map()`. It will iterate through all `<li>` elements and return a comma-separated string of the elements' text:

Click here to view code image

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

Notice that for each iteration, the function returns the HTML content in the `<li>` element. You call `.get()` to return a JavaScript array version of the new jQuery object returned by `.map()` and then call `.join(",")` on that array to build the comma-separated string.

Try it Yourself: Using the jQuery .map() and .each() Methods to Navigate, Access, and Manipulate the DOM Elements

In this example, you learn the process of using a jQuery selector to find all paragraph elements. You then use the `.map()` function to read the content and use it to create a new `<span>` box. You also use `.each()` to iterate through the `<p>` elements and restyle them. The full sample files can be found in Listings 8.1, 8.2, and 8.3. Use the following steps to implement the example:

1. In Eclipse, create a source folder named lesson08. Inside the lesson08 folder, create a js and css folder.
2. In the lesson08 folder, create the source files named dom_manipulation.html, js/dom_manipulation.js, and css/dom_manipulation.css.
3. Open dom_manipulation.html and add the usual basic elements (html, head, body).
4. Inside the `<head>` element, add the following `<script>` and `<link>` elements, shown in Listing 8.1, that will be used to load the jQuery,

JavaScript, and CSS:

Click here to view code image

```
<script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
<script type="text/javascript" src="js/dom_manipulation.js"></script>
<link rel="stylesheet" type="text/css" href="css/dom_manipulation.css">
```

5. Add the code shown in lines 11–19 of Listing 8.1 to the css/dom_manipulation.css file. This code defines two input buttons, a bunch of `<p>` elements, and a `<div>` that will be used to place content.

6. Add the styling code from Listing 8.3 to the dom_manipulation.css file. The styling code will style the `<span>` elements so that they will display as inline-block so that you can set their size.

7. Open the js/dom_manipulation.js file and add the following lines that will create a `.ready()` function that will be executed when the page is loaded:

Click here to view code image

```
$(document).ready(function (){
. . .
});
```

8. Add the following code that will add a `click` event handler to the first button. The function uses a simple `.each()` function to iterate through the `<p>` elements; it gets the string, splits it into a color and size, and then sets the `font-size` and `color` CSS properties:

Click here to view code image

```
  $("input:eq(0)").click(function (){
    $("p").each(function(){
      var parts = $(this).html().split(" ");
      $(this).css({"font-size":parts[1]+"px", color:parts[0]});
    });
  });
```

9. Add the following code that will add a `click` event handler to the second button. The function uses a simple `.map()` function to iterate through the `<p>` elements; it gets the string and splits it into a color and size. This time, however, the function returns a JavaScript object with a `color` and `size` attribute. The `.get()` at the end converts the results of the `.map()` to a JavaScript array named `items`:

Click here to view code image

```
  $("input:eq(1)").click(function (){
```

```
var items = $("p").map(function(){
  var parts = $(this).html().split(" ");
  return {color:parts[0], size:parts[1]};
 }).get();
```

10. Add the following `for` loop that iterates through items and creates a new `<span>` element with the color and size based on the values read from the `<p>` elements:

Click here to view code image

```
for (var idx in items){
 var item = items[idx];
 var span = $("<span>" + item.color + "</span>");
 var size = item.size*5;
 span.css({"background-color":item.color,  "font-size":
item.size+"px", width:size, height:size});
 $("div").append(span);
}
```

11. Save the three files and open the HTML document in a web browser. When you click the `.each()` button, the text of the `<p>` elements should change, as shown in Figure 8.1. When you click the `.map()` button, new boxes should be displayed in the `<div>`.

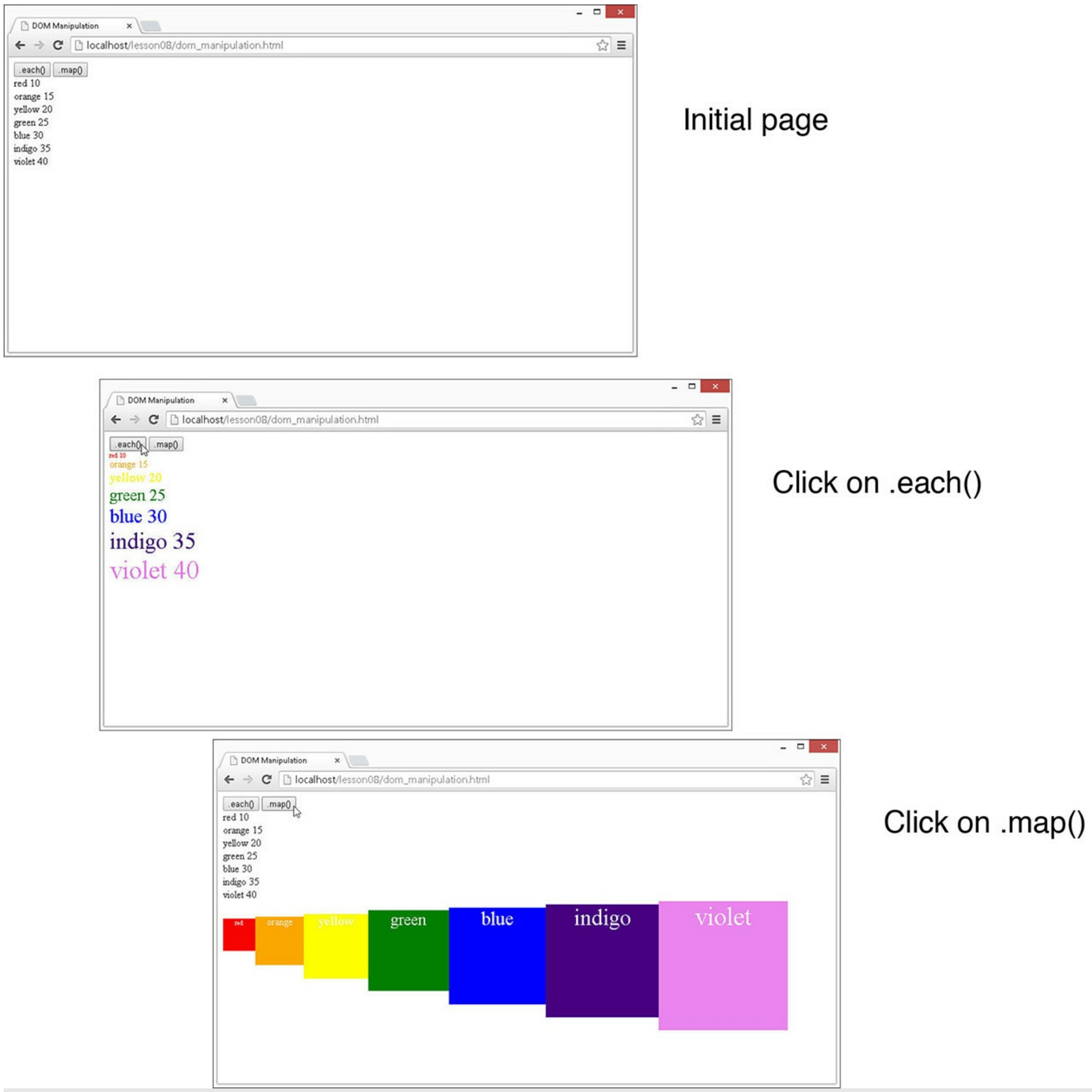


FIGURE 8.1 A simple JavaScript web page that uses `.map()` and `.each()` to read the text in `<p>` elements and make dynamic changes to the web page for each of those elements.

LISTING 8.1 dom_manipulation.css HTML File That Loads jQuery and JavaScript

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/dom_manipulation.js">
</script>
    <link rel="stylesheet" type="text/css"
href="css/dom_manipulation.css">
  </head>
  <body>
    <input type="button" value=".each()">
    <input type="button" value=".map()">
    <p>red 10</p>
    <p>orange 15</p>
    <p>yellow 20</p>
    <p>green 25</p>
    <p>blue 30</p>
    <p>indigo 35</p>
    <p>violet 40</p>
    <div></div>
  </body>
</html>
```

LISTING 8.2 dom_manipulation.js jQuery and JavaScript Code Gets the `<p>` Elements and Iterates Through Them Using `.map()` and `.each()` to Apply Different Changes for Each Element

Click here to view code image

```
$(document).ready(function (){
  $("input:eq(0)").click(function (){
    $("p").each(function(){
      var parts = $(this).html().split(" ");
      $(this).css({"font-size":parts[1]+"px", color:parts[0]});
    });
  });
  $("input:eq(1)").click(function (){
    var items = $("p").map(function(){
        var parts = $(this).html().split(" ");
        return {color:parts[0], size:parts[1]};
      }).get();
    for (var idx in items){
      var item = items[idx];
      var span = $("<span>" + item.color + "</span>");
      var size = item.size*5;
      span.css({"background-color":item.color,  "font-size":
item.size+"px",
                  width:size, height:size});
      $("div").append(span);
    }
  });
});
```

LISTING 8.3 dom_manipulation.css CSS Code That Styles the `<span>` and `<p>` Elements

Click here to view code image

```
p{margin:0px; padding:0px;}
span{
  display:inline-block;
  color: white;
  text-align:center;
}
```

Try it Yourself: Using jQuery Objects to Traverse the DOM

In this example, you use jQuery objects to dynamically access DOM elements relative to their position from each other. The purpose of this example is to illustrate how easy it is use jQuery objects to navigate and find other related DOM elements. The results will be a simple web page that allows users to input ratings and provides a graphical indicator of their value. The full scripts used in the example are shown in Listings 8.4, 8.5, and 8.6:

1. In the lesson08 folder, create the source files named traverse_dom.html, js/traverse_dom.js, and css/traverse_dom.css.
2. Open traverse_dom.html and add the usual basic elements (html, head, body).
3. Inside the `<head>` element, add the following `<script>` and `<link>` elements, shown in Listing 8.4, that will be used to load the jQuery, JavaScript, and CSS:

Click here to view code image

```
06     <script type="text/javascript"
src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
07     <script type="text/javascript" src="js/traverse_dom.js">
</script>
08     <link rel="stylesheet" type="text/css"
href="css/traverse_dom.css">
```

4. Add the code shown in lines 11–25 of Listing 8.4. This code defines a set of `<div>` elements containing a label, a text input, and a set of five `<span>` elements. The `onkeyup="update()"` attribute of the `<input>` elements provides the dynamic interaction when the user types into the text box.
5. Add the styling code from Listing 8.6 to the traverse_dom.css file. The styling code styles the `<span>` elements with a certain `height` and `width` and

adds the `margin` and `padding` necessary.

6. Open the traverse_dom.js file in Eclipse and create a JavaScript function named `update()` to be called by the `onkeyup` event handler.
7. Add the following line to set the `background-color` of all `<span>` elements to `lightgrey`:

Click here to view code image

```
$("span").css("background-color","lightgrey");
```

8. Add the following jQuery code to select all `<div>` elements, and then use `.each()` to iterate on each of them and apply a function:

Click here to view code image

```
$("div").each(function(i){
...
})
```

9. Inside the function, add the following line that gets the first `<input>` child of the `<div>` element:

Click here to view code image

```
  var $input = $(this).children("input:first");
```

10. Add the following lines that get the value of the `<input>` element and create a jQuery selector string to filter on only the first *n* number of `<span>` elements, where `n` is the value of `<input>`:

Click here to view code image

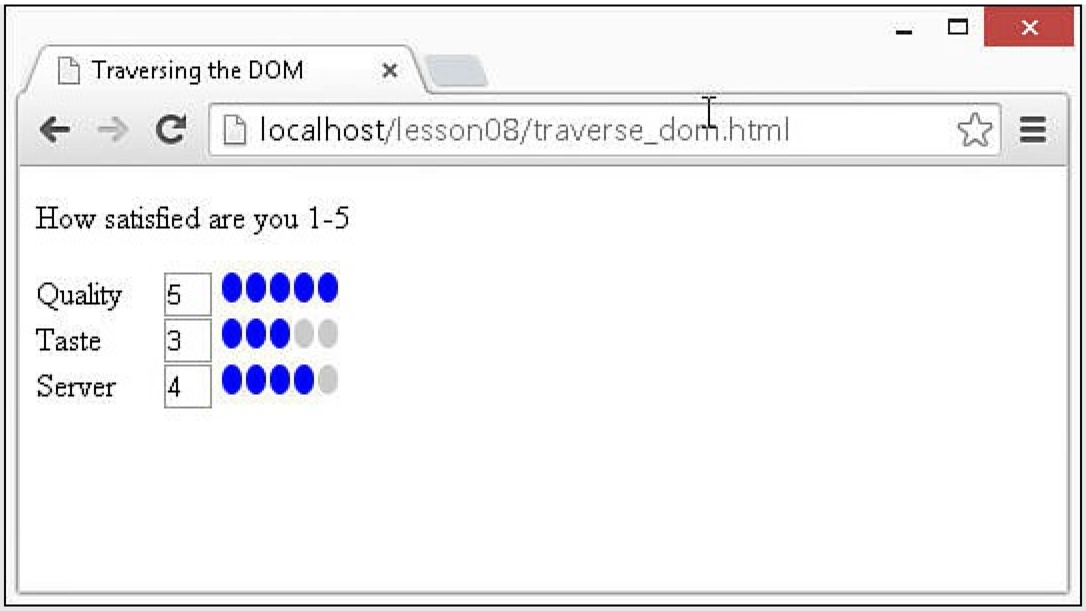
```
  var $value = $input.val();
  var filter = "span:lt(" + $value + ")";
```

11. Add the following line that uses the `$input` jQuery object to search for siblings based on the `filter` defined in step 10. The `css()` method changes the background color to blue. Notice the simplicity of the function. You can add as many `<div>` sections as you want. As long as they follow the same structure, `<div><input><span>...</div>`, the jQuery code will work on all of them:

Click here to view code image

```
  $input.siblings(filter).css("background-color","blue");
```

12. Save the files and open traverse_dom.html in a web browser. Notice that as you type into the text boxes, the `<span>` elements are changed to blue to match the value of the input.



How satisfied are you 1-5

Quality 5 ●●●●●
Taste 3 ●●●●●
Server 4 ●●●●●

FIGURE 8.2 A simple JavaScript web page that allows users to input a rating and dynamically changes based on the value typed in.

LISTING 8.4 traverse_dom.html HTML File That Loads jQuery and JavaScript and Attaches Event Handler Elements to Provide User Interaction

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Traversing the DOM</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/traverse_dom.js"></script>
    <link rel="stylesheet" type="text/css" href="css/traverse_dom.css">
  </head>
  <body>
    <p>How satisfied are you 1-5</p>
    <div>
      <label>Quality</label>
      <input type="text" onkeyup="update()"></input>
      <span></span><span></span><span></span><span></span><span></span>
    </div>
    <div><label>Taste</label>
      <input type="text" onkeyup="update()"></input>
      <span></span><span></span><span></span><span></span><span></span>
    </div>
    <div>
      <label>Server</label>
      <input type="text" onkeyup="update()"></input>
      <span></span><span></span><span></span><span></span><span></span>
    </div>
  </body>
</html>
```

LISTING 8.5 traverse_dom.js JavaScript Code That Handles the Key Up Event and Uses jQuery to Manipulate the Color of the `<span>` Elements Based on the Input Value

Click here to view code image

```
function update(){
  $("span").css("background-color","lightgrey");
  $("div").each(function(i){
    var $input = $(this).children("input:first");
    var $value = $input.val();
    var filter = "span:lt(" + $value + ")";
    $input.siblings(filter).css("background-color","blue");
  })
}
```

LISTING 8.6 traverse_dom.css CSS Code That Styles the `<span>`, `<input>`, and `<label>` Elements

Click here to view code image

```
span{
  display:inline-block;
  height:15px;
  width:10px;
  background-color:lightgrey;
  margin:1px;
  border-radius:50%;
}
input {
  width:20px;
}
label {
  display:inline-block;
  width:60px;
}
```

Summary

In this lesson, you learned about using jQuery and JavaScript objects to find and navigate through the DOM elements. This adds a critical piece in implementing dynamic code because often you will want to act, not on the elements that you search for, but for elements related to them.

You learned how to chain jQuery requests together to apply multiple operations to the same set of DOM elements. This reduces the number of statements required in your scripts.

You also learned how to iterate through the DOM element set associated with the jQuery objects returned by the selector. This allows you to apply a different set of operations to each individual element in a set without the need to look up each one individually.

Q&A

Q. What is the difference between `$("div:eq(n)")` and `$("div").eq(n)`?

A. The biggest difference is that `.eq()` enables you to specify a negative number and count backward from the end. For example, `.eq(-1)` is the last element in the list. `:eq()` does not allow negative indexes.

Q. How many jQuery options should be chained together?

A. Good question. It depends on the circumstances. It is really a question of performance versus readability/reusability. Keep in mind that there is a bit of code behind each jQuery lookup, so the more you can use the same jQuery stack, the better off you are. At some point, it can get very confusing about what is really in the current stack; however, at that point, you should save yourself future headaches and move on. A better approach is to break the operations into separate groups that can be reused and then define variable names for those.

Workshop

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

Quiz

1. What jQuery method would you use to filter out elements that have a `<p>` element as a child?
2. What jQuery method would you use to get all sibling `<p>` elements before an element with `id="middleEarth"`?
3. What is the difference between `.map()` and `.each()`?

Quiz Answers

1. `.has("p");`

2. `$("#middleEarth").prevAll("p");`
3. `.map()` is used to iterate through the set of elements and return a new object containing the results of the specified function of each iteration. `.each()` will just iterate through the elements applying the function specified.

Exercise

1. Extend the HTML code in dom_manipulation.html to include more `<div>` sections like the others. Notice that the same functionality is automatically applied to each.

Lesson 9. Applying JavaScript and jQuery Events for Richly Interactive Web Pages

What You'll Learn in This Lesson:

- Ways that events can be triggered by the user and browser
- What properties are accessible on event objects
- How to add handlers that are able to dynamically respond to events
- Ways to pass extra data to event handlers
- Methods to manually trigger events
- Ways to create your own custom events
- How to implement and fire off callback functions in your code

One of the major goals of AngularJS, jQuery, and JavaScript is to provide developers with the capability to create incredibly sophisticated and richly active web pages. At the heart of interactive web pages are events. An event is basically anytime anything happens in the browser environment, from a page loading, to a mouse movement or click, to keyboard input, to resizing the window.

Understanding events, the objects that represent them, and how to apply them in your web pages will enable to you to create some spectacular user interaction. In this lesson, you learn the concepts required to understand events and how to utilize them in building rich, interactive web pages. You will be using the concepts in this lesson throughout the rest of the book.

AngularJS utilizes the underlying JavaScript and jQuery event functionality discussed in this lesson, but also delivers additional functionality, such as data watching, for more robust event handling. Those concepts are covered in the AngularJS section of lessons.

Understanding Events

You have already been using events in several of the examples in this book. However, to get the most out of them, you need to understand what is happening in the entire process from the physical interaction to the AngularJS, jQuery, and JavaScript interactions that will modify screen elements and data. The following sections describe the event process, the components involved, and how to implement events in web pages.

Understanding the Event Process

The event handling concept is pretty easy to catch at a high level; you click a button on the web page and things happen. This section delves a little bit deeper so you can understand what is going on. The following list describes the important things that happen when a user interacts with the web page or browser window.

1. **A physical event happens**—A physical event occurs; for example, a user clicks or moves the mouse or presses a key.
2. **Events are triggered in the browser**—The user interaction results in events being triggered by the web browser—often, multiple events at the same time. For example, when a user types a key on the keyboard, three events are triggered: the `keypressed`, `keydown`, and `keyup` events.
3. **The browser creates an object for the event**—The web browser creates a separate object for each event that is triggered. The objects contain information about the event that can be used by handlers.
4. **User event handlers are called**—User-defined event handlers are called. You have the capability to create handlers in JavaScript that will interact with the event objects and/or page elements to provide interactivity with HTML elements. There are three phases that the event handlers can be acting in. The following describes the three phases shown in Figure 9.1:
   - **Capturing**—The capturing phase is on the way down to the target HTML element from the document directly through each of the parent elements. By default, behavior for event handlers for the capturing phase is disabled.
   - **Target**—The target phase occurs when the event is in the HTML element where it was initially triggered.
   - **Bubbling**—The bubbling phase is on the way up through each of the parents of the target HTML element, all the way back to the document. By default, the bubbling phase is enabled for events.

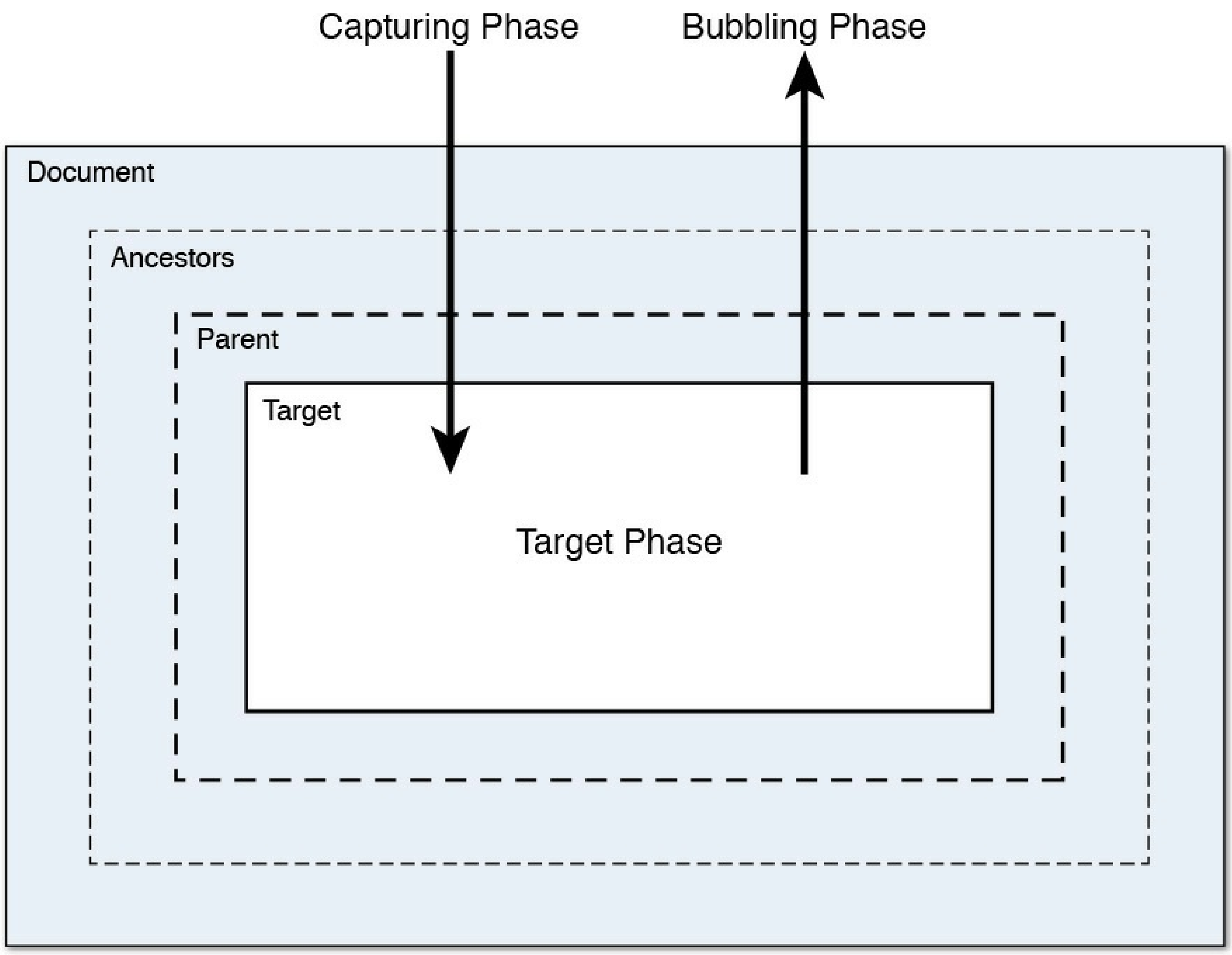


FIGURE 9.1 Events are handled first in the capturing phase from the document down to the target, then in the target phase, and finally, the bubbling phase from the target up through the document.

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

Looking at Event Objects

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

The event object provides you with additional information about the event, such as what type the event was—for instance, a `click` or `keypress`—which key(s) were pressed, the position of the mouse, what HTML element the event occurred on, and so on. Table 9.1 describes the most commonly used event attributes that you will be working with.

Property	Description
`altKey`	True if the Alt key was pressed during the event; otherwise, false.
`button`	Returns the number of the mouse button that was pressed: 0 for left, 1 for middle, and 2 for right.
`cancelable`	True if the default action for the event can be stopped; otherwise, false.
`charCode`	Ordinal value of the character that is pressed if it is a keyboard event.
`clientX`	Horizontal coordinate of the mouse pointer relative to the current window.
`clientY`	Vertical coordinate of the mouse pointer relative to the current window.
`ctrlKey`	True if the Ctrl key was pressed during the event; otherwise, false.
`currentTarget`	The DOM object for the HTML element that the event handler currently being executed is attached to.
`data`	User data that is defined and attached to the event when the event is triggered.

`delegateTarget`	This is the DOM object of the HTML element that was used in the jQuery `.delegate()` or `.on()` method to attach the event handler. (`.on()` and `.delegate()` are discussed later in this lesson.) Available only on jQuery event objects.
`eventPhase`	The current phase that the event handler is operating in, where 1 is at the target, 2 is bubbling, and 3 is capturing.
`metaKey`	True if a “meta” key was pressed during the event; otherwise, false. For example, true if Ctrl, Alt, Cmd, or another meta key is pressed.
`relatedTarget`	Identifies a secondary target relative to the UI event. For example, when using the `mouseover` and `mouseexit` events, it indicates the target being exited or entered.
`results`	Last value returned by an event handler that was triggered by this event. Available only on jQuery event objects.
`screenX`	Horizontal coordinate based on the actual display coordinate system.
`screenY`	Vertical coordinate based on the actual display coordinate system.
`shiftKey`	True if the Shift key was pressed during the event; otherwise, false.
`target`	The DOM object for the HTML element where the event originated.
`type`	Text describing the event, such as “click” or “keydown”.
`timeStamp`	Time in ms since January 1, 1970 and when the event was triggered.
`which`	Actual numerical value.

TABLE 9.1 JavaScript and jQuery Event Object Attributes

Events also provide a few methods that allow you some control into the behavior of the event—for example, stopping propagation and default web behavior. Table 9.2 describes the important methods on event objects.

Method	Description
`isDefaultPrevented()`	True if the default action has been disabled; otherwise, false.
`isImmediatePropagationStopped()`	True if immediate propagation has been disabled; otherwise, false.
`isPropagationStopped()`	Returns true if propagation has been stopped; otherwise, false.
`preventDefault()`	If an event can be canceled, this method will disable the default action that would be applied by the browser. For example, you could disable a link from navigating away from the web page.
`stopImmediatePropagation()`	Stops the event from propagating to any other event handlers that are attached to the current target, as well as from propagating through the DOM tree.
`stopPropagation()`	Stops the event from propagating to other HTML elements in the DOM tree. Other events on the current target will still complete.

TABLE 9.2 JavaScript and jQuery Event Object Attributes

Reviewing Event Types

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

JavaScript provides several event types that correspond to the different events that occur during dynamic page interaction. jQuery event handling supports all the JavaScript and also adds a few of its own events. Table 9.3 lists the different event types supported by JavaScript and jQuery.

Event Type	Description
`abort`	Triggered when an image load is stopped before completing.
`blur`	Triggered when a form element loses focus.
`change`	Triggered when the content, selection, or check state of a form element changes. Applies to `<input>`, `<select>`, and `<textarea>` elements.
`click`	Triggered when the user clicks an HTML element.
`dblclick`	Triggered when the user double-clicks an HTML element.
`error`	Triggered when an image does not load correctly.
`focus`	Triggered when a form element gets the focus.
`focusin`	Triggered when a form element or any descendant element inside of it gets the focus. Different from the focus event type in that it supports bubbling up to parents. This is a jQuery-only event.
`focusout`	Triggered when a form element or any element inside of it loses the focus. Different from blur in that it supports bubbling up to parents. This is a jQuery-only event.
`keydown`	Triggered when a user is pressing a key.
`keypress`	Triggered when a user presses a key.
`keyup`	Triggered when a user releases a key.
`load`	Triggered when a document, frameset, or DOM element is loaded.
`mousedown`	Triggered when a user presses a mouse button. The target element is the one the mouse is over when the button is pressed.
`mousemove`	Triggered when the mouse cursor is moving over an element.
`mouseover`	Triggered when the mouse cursor moves onto an element or any of its descendants.
`mouseout`	Triggered when the mouse cursor moves out of an element or any of its descendants.

`mouseup`	Triggered when a user releases a mouse button. The target element is the one that the mouse is over when the button is released.
`mouseenter`	Triggered when the mouse cursor enters an element. Different from `mouseover` in that it is triggered only by the element and not its descendants. This is a jQuery-only event.
`mouseleave`	Triggered when the mouse cursor leaves an element. Different from `mouseout` in that it is triggered only by the element and not its descendants. This is a jQuery-only event.
`reset`	Triggered when a form is reset.
`resize`	Triggered when the document view is resized by resizing the browser window or frame.
`scroll`	Triggered when a document view is scrolled.
`select`	Triggered when a user selects text in an `<input>` or `<textarea>`.
`submit`	Triggered when a form is submitted.
`unload`	Triggered when a page is unloaded from the `<body>` or `<frameset>` element.

TABLE 9.3 JavaScript and jQuery Event Types

Using the Page Load Events for Initialization

When the HTML document loads, the JavaScript code specified in the `<script>` tags will be loaded and executed. Typically, the JavaScript and jQuery logic will be inside functions that will be executed later. However, there will be some code that does initialization work, such as attaching event handlers to page elements or even adding additional elements to existing ones.

The problem is that the HTML element objects may not have been built by the browser at the point when the JavaScript code is loaded. An exception will be thrown if you try to reference the HTML object before it is created, so you need to wait until the page has fully loaded.

That is where the load event is extremely handy. Placing your initialization code inside of an event handler that gets triggered when the page is loaded allows you to be sure all HTML objects have been created and the DOM is fully ready.

Using the JavaScript onload Event

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

[Click here to view code image]

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

Now you have two options to cause the `onloadHandler()` to trigger when the page is fully loaded. The first is to attach your initialization code to the `onload` attribute of the `<body>` element in the HTML. For example:

Click here to view code image

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

The second method is to assign it directly to the `window` object in your JavaScript code. For example:

Click here to view code image

```
window.onload = onloadHandler;
```

Adding Initialization Code in jQuery

In jQuery, initialization code can be triggered and executed at two different times: when the DOM is ready and when the document and its resources have fully loaded. Which of these options you use will depend on what needs to happen in your initialization code.

Using the `.ready()` jQuery method will trigger the initialization code to run when the DOM is fully ready. All of the DOM objects will have been created and the page will be displayed to users. All page resources, such as images, may not have fully downloaded at this point, however. This is the option that enables you to add interactions and functionality as soon as possible. The following shows an example of using `.ready()` to attach a simple initialization function:

Click here to view code image

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

Using the `.load()` jQuery method will trigger the initialization code to run after all page resources have loaded and are rendered to the user. On occasion, you might use this option if you need resources such as images available as part of the initialization code. The following shows an example of using `load()` to attach a simple initialization function:

Click here to view code image

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

Caution

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

Adding and Removing Event Handlers to DOM Elements

To provide interaction to events that occur in a web page, an event handler must be added for each specific event type you want to interact with. An event handler is a JavaScript function that adds, removes, or alters DOM elements, or whatever else is necessary to complete the interaction. For example, the following code is an example of an event handler that changes the text displayed in the `<p>` elements to `"clicked"`:

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

Several methods enable you to add event handlers to DOM elements. The following section describes each of these methods along with their advantages and disadvantages.

Assigning Event Handers in HTML

The most basic methods of adding an event handler to a DOM element is directly in the HTML code. The advantage of this method is that it is simple and easy to see what event handler gets assigned to a particular DOM object.

Event handlers are added to DOM objects in HTML by setting the value of the handler attribute in the tag statement. For each event that the element supports, there is an attribute that starts with "on" followed by the name of the event. For example, the `click` event attribute is `onclick` and the `load` event attribute is `onload`. To see a list of the events, see Table 9.1.

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

Click here to view code image

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

This would call the following function when the `<div>` element is clicked:

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

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

Click here to view code image

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

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

Click here to view code image

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

You can also define that specific arguments are passed to an event, allowing you to distinguish specific functionality. The parameters can be numbers or strings surrounded by single quotes or even the id value of DOM objects, in which case the DOM element will be passed. For example, the following code adds event handlers that pass the event object, as well as the DOM object for the `<h1>` element with `id="heading"` along with a number and string:

Click here to view code image

```
<h1 id="heading"></h1>
<div onclick="clickHandler(event,heading,1,"Yes")">Click Here</div>
<div onclick="clickHandler(event,heading,2,"No")">Or Here</div>
```

When clicked, the elements would call the following function and use the DOM objects' `innerHTML` property to change the text to display the number, message, and x coordinate of the mouse cursor:

Click here to view code image

```
function clickHandler(e,obj,num,msg){
  obj.innerHTML = "DIV " + num + " says " + msg +" at X postion: " +
e.screenX;
}
```

There are a couple of major disadvantages to this method, though. The first is that the DOM element has to be defined in the HTML and never removed and read dynamically. The second is that if you want the same event handler assigned to several DOM elements, you have to add code to the tag to every one of them in the HTML.

Typically, you use the HTML method of assigning event handlers only for basic examples with very little JavaScript.

Adding Event Handlers in JavaScript

You can also add and remove event handlers dynamically to DOM objects inside of JavaScript code. This method provides the advantage that you can dynamically add event handlers at any point in time, not just when the page is loaded. Thus, it enables you to add event listeners to elements created after the page load.

To add an event handler in JavaScript, call `addEventListener()` on the DOM object. The `addEventListener()` function takes three parameters: the first is the event `type` (event types are defined in Table 9.1), the second is the function to call, and the third is a Boolean that specifies true if the handler should be called during the capturing phase and the bubbling phase, or false if the handler should be called only during the bubbling phase.

The trick with using `addEventListener()` is how to pass custom data into the handler call. One method is to wrap the actual function handler inside of a simple wrapper function that passes the arguments to the event handler. The following code illustrates this:

Click here to view code image

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

So here is what is going on in the `addEventListener()` calls in the preceding code. The event type `'click'` is specified, and then a new anonymous `function` with no name is defined that accepts the event object as the only parameter represented by `e`. Inside the wrapper function, the actual event handler is called with the custom parameters, including the `e` variable to pass the event object along. In the example, false is used to indicate that the handler should be called only during the bubbling phase.

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

Click here to view code image

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

Tip

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

Try it Yourself: Adding Event Handlers to DOM Objects via JavaScript

In this example, you attach event handlers to DOM objects dynamically using the JavaScript method. Use the following steps to create the files shown in Listings 9.1, 9.2, and 9.3:

1. In Eclipse, create the lesson09, lesson09/js, and lesson09/css folders.
2. Create the lesson09/broken_event.html, lesson09/js/broken_event.js, and lesson09/css/broken_event.css files.
3. Add the code shown in Listing 9.1 to the HTML file. You should recognize almost everything in this file. Notice the `onload` event handler for the `<body>` tag is set to `onloadHandler()`.
4. Add the css from Listing 9.3 to the CSS file. You should also recognize the CSS statements by now.
5. Open the broken_event.js file and the following function that will be used to handle click events. Notice that it takes several arguments. The first argument is the DOM event object, which is used to get the x position via `e.screenX`. It also accepts an object id that is used to get the object that should be written to. The rest of the arguments are used to write out the message the `innerHTML` of the object specified by `objId`:

Click here to view code image

```
function clickHandler(e,objId,num,msg){
  var obj = document.getElementById(objId);
  obj.innerHTML = "DIV " + num + " says " + msg +" at X postion: "
+ e.screenX;
}
```

6. Create the following wrapper functions that will be used to pass arguments to the actual event handler. The wrapper functions only accept the event object as argument `e`. Then they call the `clickHandler()` function with the appropriate values. In this example, the event handler is then removed from the target object by calling `removeEventListener()`. It is not typical to remove event handlers; this is just to provide an example:

Click here to view code image

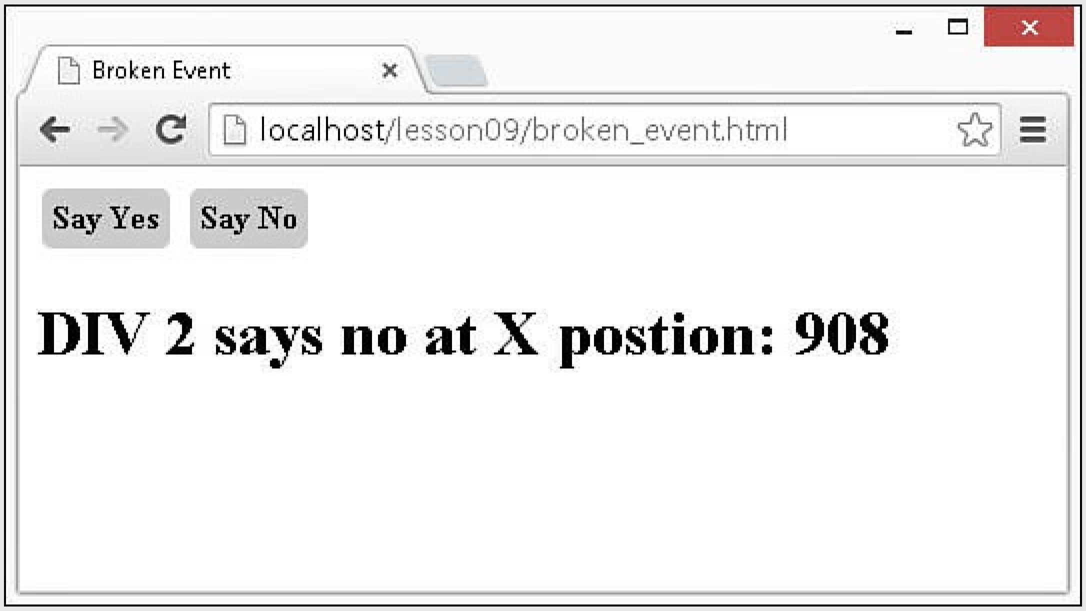
```
function yesWrapper(e){
  clickHandler(e, "heading", 1, "yes");
  e.target.removeEventListener('click', yesWrapper);
}
function noWrapper(e){
  clickHandler(e, "heading", 1, "no");
  e.target.removeEventListener('click', noWrapper);
}
```

7. Add the following `onloadHander()` function that calls `addEventListener()` to add a `click` event handler to each of the `<div>` elements:

Click here to view code image

```
function onloadHandler(){
  document.getElementById("div1").addEventListener('click', yesWrapper, false);
  document.getElementById("div2").addEventListener('click', noWrapper, false);
}
```

8. Save all three files and then open the HTML document in a web browser and play around with the Say Yes and Say No buttons shown in Figure 9.2. Notice that the buttons work only once because the event handler was removed. If we had not removed the event handler, the buttons would continue to handle the event and change the `<h1>` element text.



DIV 2 says no at X postion: 908

FIGURE 9.2 A simple JavaScript web page that dynamically adds event handlers to `<div>` elements and responds to mouse clicks.

LISTING 9.1 broken_event.html HTML File That Loads jQuery and JavaScript, Attaches Event Handlers Elements to Provide User Interaction, and Then Defines the `<div>` and `<h1>` Elements Used in the Example

Click here to view code image

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

```
  <head>
    <title>Broken Event</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/broken_event.js"></script>
    <link rel="stylesheet" type="text/css" href="css/broken_event.css">
  </head>
  <body onload="onloadHandler()">
    <div id="div1")">Say Yes</div>
    <div id="div2")">Say No</div>
    <h1 id="heading"></h1>
  </body>
</html>
```

LISTING 9.2 broken_event.js JavaScript Code That Defines an Event Handler and Dynamically Attaches It to the `<div>` Elements

Click here to view code image

```
function clickHandler(e,objId,num,msg){
  var obj = document.getElementById(objId);
  obj.innerHTML = "DIV " + num + " says " + msg +" at X postion: " +
e.screenX;
}
function yesWrapper(e){
  clickHandler(e, "heading", 1, "yes");
  e.target.removeEventListener('click', yesWrapper);
}
function noWrapper(e){
  clickHandler(e, "heading", 2, "no");
  e.target.removeEventListener('click', noWrapper);
}
function onloadHandler(){
  document.getElementById("div1").addEventListener('click', yesWrapper,
false);
  document.getElementById("div2").addEventListener('click', noWrapper,
false);
}
```

LISTING 9.3 broken_event.css CSS Code That Styles the `<div>` Elements

Click here to view code image

```
div{
  border-radius:5px;
  margin:3px;
  padding:5px;
  background-color:lightgrey;
  font-weight:bold;
  display:inline-block;
  cursor:pointer;
}
```

Applying Event Handlers in jQuery

The best method of attaching event handlers to DOM elements is using jQuery. Using jQuery objects makes it simple to select different sets of objects and apply the same event handler to all of them at the same time.

In the past, jQuery has had a couple of ways to add and remove event handlers, including `bind()/unbind()` and `delegate()/undelegate()`. The `bind()` method mimicked the behavior of `addEventListener()`, and `delegate()` made it possible to add an event to a jQuery object and then, as additional child elements were added to the event handler, would automatically be delegated to those as well.

As of jQuery 1.7, these methods have all been replaced by a simple pair, `on()` and `off()`. Event handlers are attached to jQuery objects using the `on()` method. The `on()` method can be called in one of the two following formats:

Click here to view code image

```
on(events [, selector] [, data], handler(eventObject))
on(events-map [, selector][, data])
```

The following list describes the purpose of each of the arguments that can be added to the `on()` method when adding event handlers to jQuery objects:

- **events**—One or more space-separated event types and optional namespaces denoted by dot syntax; for example, `"click"`, `"mouseenter mouseleave"`, or `"keydown.myPlugin"`. The list of event types can be found in Table 9.1.
- **events-map**—A mapping object in which the string keys specify one or more space-separated event types, and then the values specify handler functions that will be called when the event is triggered; for example, `{'click':myhandler}` or `{'click':function myHandler(e) {return true;}}`.
- **selector**—Optional. Selector string that specifies which descendants should also call the event handler when the event is triggered. This replaces the functionality of the `delegate()` method. If you add the event handler to a set of parent objects that will always exist in the page, any of their descendants that match the selector will also call the handler when the event is triggered, even if they get added after the event is attached.

- **data**—Optional. This can be a number, a string, or an object that will get passed to the handler as the data part of the `event` as `event.data` when an event is triggered.
- **handler(eventObject)**—If you are not using an `events-map`, you will need to specify the handler function that will be executed when the event is triggered.

To remove an event handler from elements using jQuery, call the `off()` method on the jQuery object. The syntax for the off method is one of the following:

Click here to view code image

```
off(events [, selector] [, handler(eventObject]))
off(events-map [, selector])
```

If no handler is specified, the `off()` function removes all event handlers for the events specified. The following example shows a basic example of adding an event handler to all `<div>` elements using `on()` and then removing it using `off()`:

Click here to view code image

```
$("div").on("click",clickHandler);
$("div").off("click",clickHandler);
```

Note

The `on()` and `off()` methods work for all jQuery event types, including your own custom events. jQuery also provides some simple helper functions that are discussed later in the lesson to make it easier to attach handlers for certain events.

Tip

Often you will want an event handler to run only once, the first time the event occurs. jQuery provides the very helpful `one()` method that will add an event, then automatically remove it after it triggers for the first time. The `one()` method uses the same syntax as the `on()` method.

Try it Yourself: Adding Event Handlers Using jQuery

In this example, you attach event handlers to DOM objects dynamically using the jQuery method. Use the following steps to create the files shown in Listings 9.4, 9.5, and 9.6:

1. In Eclipse, create the lesson09/working_event.html,

lesson09/js/working_event.js, and lesson09/css/working_event.css files.

2. Add the code shown in Listing 9.4 to the HTML file. You should recognize almost everything in this file.
3. Add the css from Listing 9.6 to the CSS file.
4. Open the working_event.js file and the following function that will be used to handle click events. The event takes only the event object as argument `e`. It relies on the data portion of the event object to provide the `objId` of the element that text should be added to. It also uses the `e.target.id`, `e.screenX`, and `e.data.answer` values as part of the message written to the `<h1>` element:

Click here to view code image

```
function clickHandler(e){
  $("#"+e.data.objId).html(e.target.id + " says " + e.data.answer +
                               " at X postion: " + e.screenX);
}
```

5. Create the following `ready()` function that will automatically be called when the document has been loaded and is ready:

Click here to view code image

```
$(document).ready(function(){
...
});
```

6. Add the following jQuery code that finds the `#div1` element and then uses `on()` to add an event handler. In this example, an `events-map` object is passed specifying `clickHandler()` for the click event type. In addition, an object is passed in that it will add the heading and answer attributes to the event object data. This is where the values came from that we used in step 4:

Click here to view code image

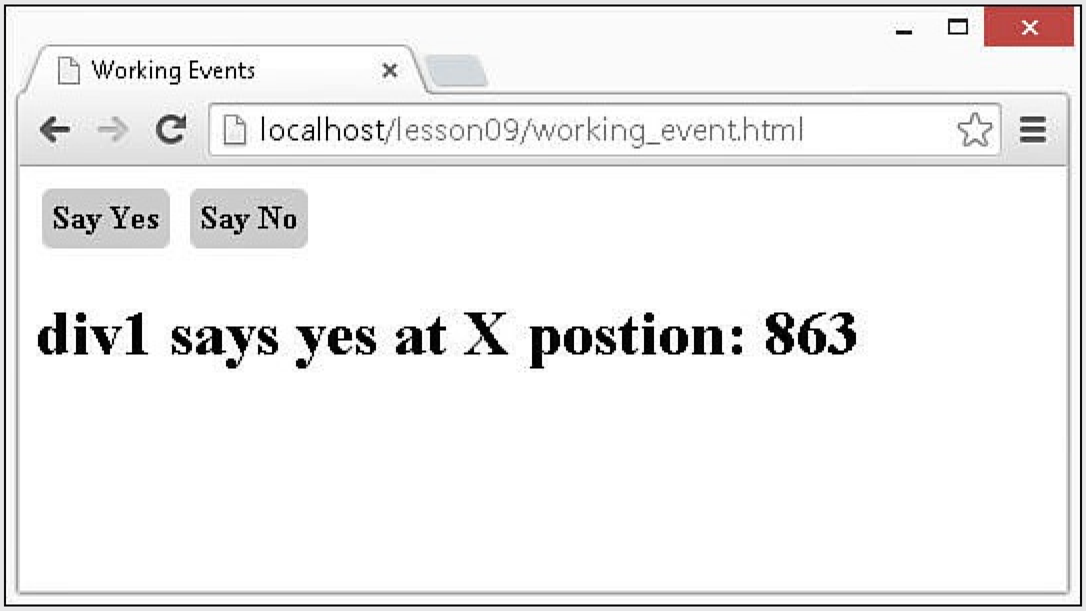
```
  $("#div1").on({"click":clickHandler},
                  {"objId":"heading", "answer":"yes"});
```

7. Add the following jQuery code that gets the `document` object and then adds the event handler to the `#div2` element. The difference between this method and the one in step 6 is that we use a selector. By using a selector, you can delete and re-add the `#div2` element, and it will still call the event handler. Also notice that, rather than an `event-map`, you use the events method and add the hander as the final argument:

Click here to view code image

```
  $(document).on("click",
                   "#div2",
                   {"objId":"heading", "answer":"no"},
                   clickHandler);
```

8. Save all three files and then open the HTML document in a web browser and play around with the Say Yes and Say No buttons shown in Figure 9.3.



div1 says yes at X postion: 863

FIGURE 9.3 A simple JavaScript web page that uses jQuery to dynamically add event handlers to `<div>` elements and responds to mouse clicks.

LISTING 9.4 working_event.html HTML File That Loads jQuery and JavaScript, Attaches Event Handlers Elements to Provide User Interaction, and Defines the `<div>` and `<h1>` Elements Used in the Example

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Working Events</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/working_event.js"></script>
    <link rel="stylesheet" type="text/css"
href="css/working_event.css">
  </head>
  <body>
    <div id="div1")">Say Yes</div>
    <div id="div2")">Say No</div>
    <h1 id="heading"></h1>
  </body>
</html>
```

LISTING 9.5 working_event.js jQuery and JavaScript Code That Defines an Event Handler and Dynamically Attaches It to the `<div>` Elements

[Click here to view code image]

```
function clickHandler(e){
  $("#"+e.data.objId).html(e.target.id + " says " + e.data.answer +
                            " at X postion: " + e.screenX);
}
$(document).ready(function(){
  $("#div1").on({"click":clickHandler},
                {"objId":"heading", "answer":"yes"});
  $(document).on("click",
                "#div2",
                {"objId":"heading", "answer":"no"},
                clickHandler);
});
```

LISTING 9.6 working_event.css CSS Code That Styles the `<div>` Elements

[Click here to view code image]

```
div{
  border-radius:5px;
  margin:3px;
  padding:5px;
  background-color:lightgrey;
  font-weight:bold;
  display:inline-block;
  cursor:pointer;
}
```

Using jQuery Event Helper Function to Assign Event Handlers

In addition to using the `addEventListener()` method to add event handlers, jQuery also provides helper functions that enable you to set the event handler. These helper functions are named after the event, so it helps your code look a bit cleaner.

The helper functions use the following syntax similar to `addEventListener()` but without the need to specify the event type:

[Click here to view code image]

```
.<event type>( [eventData], handler(eventObject))
```

The `eventData` argument is optional and, as with `addEventListener()`, will set the `data` value for the event object. The handler argument is required and specifies the function to call. Replace `<event type>` with the event name listed in Table 9.3. For example, the following two jQuery statements are equivalent ways to assign a handler to the click event:

Click here to view code image

```
obj.click( dataObj, function myHandler(e){...});
obj.addEventListener("click", dataObj, function myHandler(e){...});
```

In addition to the helper functions based on the event type name, jQuery also provides one additional helper function that is useful for mouse hovering events. The `hover()` helper function allows you to set the handlers for the `mouseenter` and `mouseleave` events at the same time. For example:

Click here to view code image

```
obj.hover(function enterHandler(e){...}, function leaveHandler(e){...});
```

Is equivalent to

Click here to view code image

```
obj.addEventListener("mouseenter", function enterHandler(e){...});
obj.addEventListener("mouseleave", function leaveHandler (e){...});
```

You can also set the same handler for both the `mouseenter` and `mouseleave` by specifying only one handler. For example:

Click here to view code image

```
obj.hover(function hoverHandler(e){...});
```

Triggering Events Manually

JavaScript and jQuery allow you to trigger events manually. This provides advantages such as simulating user interactions, tying page element interactions together, or interactions with your own custom events.

The following sections describe how to trigger events in both JavaScript and jQuery.

Triggering Events in JavaScript

The simplest way to trigger an event in JavaScript is to use the event method attached to the DOM object if there is one. Simple events such as `click`, `select`, `blur`, and `focus` have corresponding methods attached to DOM objects that support them. For example, you can trigger the `click` event on most DOM objects using the following syntax:

```
obj.click();
```

The more advanced method of triggering events in JavaScript is to create an event object and then use the `document.dispatchEvent()` call to trigger the event mechanism. This method provides much more control over event information and is the only way to trigger many of the events. The `dispatchEvent()` method involves a three-step process.

The first step is to create an event object using the `document.createEvent()` method. The `createEvent()` method requires that you specify the event `type` for the object that is being created. Table 9.4 lists some of the common event types accepted by `createEvent()`. For example, to create a mouse `click` event object, use the following statement:

Click here to view code image

```
var clickEvent = document.createEvent("MouseEvents");
```

Event Type	Initialization Method
`CustomEvent`	`event.initCustomEvent(type, canBubble, cancelable, detail)`
`HTMLEvents`	`event.initEvent(type, bubbles, cancelable)`
`KeyboardEvent`	`event.initKeyboardEvent(type, bubbles, cancelable, viewArg, ctrlKeyArg, altKeyArg, shiftKeyArg, metaKeyArg, keyCodeArg, charCodeArg)`
`MouseEvents`	`event.initMouseEvent(type, canBubble, cancelable, view, detail, screenX, screenY, clientX, clientY, ctrlKey, altKey, shiftKey, metaKey, button, relatedTarget)`
`MutationEvents`	`event.initMutationEvent(type, canBubble, cancelable, relatedNode, prevValue, newValue, attributeName, attributeChange)`
`UIEvents`	`event.initUIEvent(type, canBubble, cancelable, view, detail)`

TABLE 9.4 Common Event Types Supported by `createEvent()`

The next step is to initialize the event object with values by calling the events' initialization function. Table 9.4 lists the corresponding initialization function for the event types along with the parameters supported.

For example, to initialize the click event object, use the following statement. The type is specified as `"click"`, and `bubbling` and `cancelable` attributes are both true. Here, `window` is used as the `view` element. The coordinates don't really matter, so they remain at 0, and the additional keyboard keys don't matter, so those are set to false. The button argument is set to 0 for the left-click, and no related target is specified:

Click here to view code image

```
clickEvent.initMouseEvent("click", true, true, window, 0, 0, 0, 0, 0, 0,
                               false, false, false, false, 0, null);
```

For the most part, the parameters to the initialization functions should be self-explanatory. However, you will likely need to check on some of the specific arguments. You can find the definitions for the event objects and methods at www.w3.org/TR/DOM-Level-2-Events/events.html.

The final step is to call dispatchEvent() on the HTML object that you want to trigger the event for. For example, the following code looks up an object and then triggers the click event:

Click here to view code image

```
var obj = document.getElementById("someId");
obj.dispatchEvent(clickEvent);
```

Caution

Prior to IE9, IE used a different set of methods to create and trigger an event. Instead of createEvent(), you need to use createEventObject(), and instead of dispatchEvent(), you need to use fireEvent(). If you need to trigger events and support earlier IE browsers, you should review the documentation at MSDN on using fireEvent().

Try it Yourself: Triggering Events Manually in JavaScript

In this example, you step through the process of using an event handler from a <span> element to trigger a click event on a different element. The result will be that you can click the span element to select or deselect a check box. Use the following steps to create the files shown in Listings 9.7, 9.8, and 9.9:

1. In Eclipse, create the lesson09/manual_event.html, lesson09/js/manual_event.js, and lesson09/css/manual_event.css files.

2. Add the code shown in Listing 9.7 to the HTML file. This is just a basic HTML file that loads additional JavaScript and CSS files. There are two check boxes and two <span> elements added to the web page.

3. Add the css from Listing 9.9 to the CSS file. This code stylizes the <span> element to look like a button.

4. Open the manual_event.js file and the following `onloadHandler()` initialization function that attaches an event listener to both of the `<span>` elements:

Click here to view code image

```
function onloadHandler(){
  var employee = document.getElementById("Employee");
  employee.addEventListener('click', simpleClick, false);
  var registered = document.getElementById("Registered");
  registered.addEventListener('click', eventClick, false);
}
```

5. Create the `simpleClick()` handler function added to the first button. The code uses the `target.id` from the event to find the corresponding check box element. Then the `click()` function is called on the check box object, which triggers the `click` event just as if the mouse had clicked it:

Click here to view code image

```
function simpleClick(e){
  var cb = document.getElementById("check"+e.target.id);
  cb.click();
}
```

6. Add the following `eventClick()` handler function to the second button. This code first creates a `"MouseEvents"` object and then initializes it to a `"click"` type with the appropriate parameters. This code also uses the `target.id` from the event to find the corresponding check box element, then calls `dispatchEvent()` on the check box element's object, triggering a mouse `click` event:

Click here to view code image

```
function eventClick(e){
    var event = document.createEvent("MouseEvents");
    event.initMouseEvent("click", true, true, window,
                          0, 0, 0, 0, 0, false, false,
                          false, false, 0, null);
    var cb = document.getElementById("check"+e.target.id);
    cb.dispatchEvent(event);
}
```

7. Save all three files and then open the HTML document in a web browser, as shown in Figure 9.4. Play around with the check boxes and buttons. Notice that the buttons provide the same interaction with the check box as directly clicking the check box.

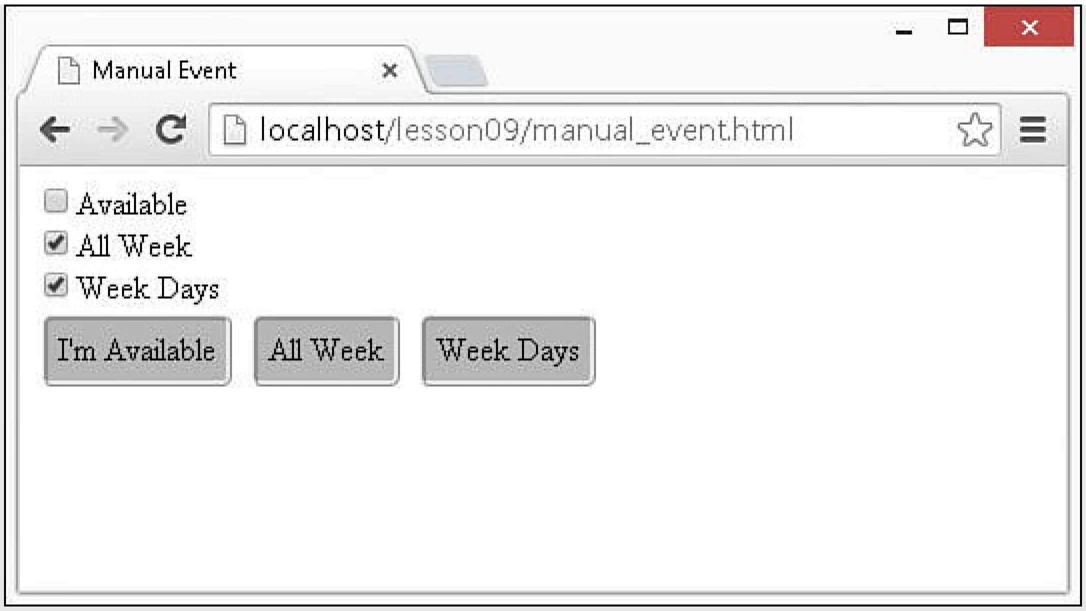


FIGURE 9.4 A simple web page that triggers the click event for check boxes from `<span>` elements.

LISTING 9.7 manual_event.html HTML File That Loads CSS and JavaScript and Defines the Check Boxes and `<span>` Elements

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Manual Event</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="js/manual_event.js"></script>
    <link rel="stylesheet" type="text/css" href="css/manual_event.css">
  </head>
  <body onload="onloadHandler()">
    <input id="checkAvailable" type="checkbox" /><label>Available</label><br>
    <input id="checkAllWeek" type="checkbox" /><label>All Week</label><br>
    <input id="checkWeekDays" type="checkbox" /><label>Week Days</label><br>
    <span id="Available">I'm Available</span>
    <span id="AllWeek">All Week</span>
    <span id="WeekDays">Week Days</span>
  </body>
</html>
```

LISTING 9.8 manual_event.js JavaScript Code Manually Triggers the Click Event for the Check Box Elements from the `<span>` Elements

Click here to view code image

```
function onloadHandler(){
  var available = document.getElementById("Available");
  Available.addEventListener('click', simbleClick, false);
  var allWeek = document.getElementById("AllWeek");
  AllWeek.addEventListener('click', eventClick, false);
  var weekDays = document.getElementById("WeekDays");
  WeekDays.addEventListener('click', eventClick, false);
}
function simbleClick(e){
  var cb = document.getElementById("check"+e.target.id);
  cb.click();
}
function eventClick(e){
    var event = document.createEvent("MouseEvents");
    event.initMouseEvent("click", true, true, window,
                          0, 0, 0, 0, 0, false, false,
                          false, false, 0, null);
    var cb = document.getElementById("check"+e.target.id);
    cb.dispatchEvent(event);
}
```

LISTING 9.9 manual_event.css CSS Code That Styles the `<span>` Elements

```
span{
  border-radius:5px;
  margin:3px;
  padding:5px;
  background-color:#C0C0C0;
  border:3px ridge;
  display:inline-block;
  cursor:pointer;
}
```

Using jQuery to Trigger Events Manually

Now that you've had a chance to trigger events from JavaScript, you are ready to discover how much easier jQuery will make your life if you are able to use it. jQuery also supports two methods to trigger events manually.

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

`<span>` elements:

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

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

Click here to view code image

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

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

Click here to view code image

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

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

Click here to view code image

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

Notice that this is much simpler than the JavaScript method because `trigger()` accepts a simple object with the appropriate attribute names to set whatever values in the event object you might want to pass in. Occasionally, you might end up having to use the JavaScript method, but you should use jQuery as often as possible.

Try it Yourself: Triggering Events Manually in jQuery

In this example, you step through the process of using an event handler from a `<span>` element to trigger a `keypress` event on a text `<input>` element. The result is that the click on the `<span>` will pass through the same event handler as a keystroke in the `<input>` element. Use the following steps to create the files shown in Listings 9.10, 9.11, and 9.12:

1. In Eclipse, create the lesson09/card_suits.html, lesson09/js/ card_suits.js, and lesson09/css/card_suits.css files.
2. Add the code shown in Listing 9.10 to the HTML file. This is a basic HTML file that loads additional jQuery, JavaScript, and CSS files. There is a `<p>` element, an `<input>` element, and a series of `<span>` elements that act as buttons.
3. Add the CSS from Listing 9.12 to the CSS file. This code stylizes the `<span>`

element to look like a button and the <p> element to provide nice formatting.

4. Open the card_suits.js file and add the following `ready()` initialization function that attaches a `keypress` event handler to the `<input>` element and a `click` handler to all the `<span>` elements:

Click here to view code image

```
09 $(document).ready(function(){
10   $("input").keypress(function (e){inputHandler(e)});
11   $("span").click(function (e){spanHandler(e)});
12 });
```

5. Create the `inputHandler()` handler function that gets the `charCode` from the event object and appends the string form to the `<p>` element. At this point as you type in the text `<input>`, it will be displayed in the `<p>` element as well:

Click here to view code image

```
01 function inputHandler(e){
02   var chr = String.fromCharCode(e.charCode);
03   $("p").append(chr);
04 }
```

6. Add the following `spanHandler ()` handler function that gets the `innerHTML` of the `<span>` element and converts the first character to a code. Line 7 then triggers a `keypress` event on the `<input>` element passing in the character code for the button as the `charCode` value. This allows the user to click a button and apply the card symbol using the event handler for the text `<inputs>`:

Click here to view code image

```
05 function spanHandler(e){
06   var chrCode = e.target.innerHTML.charCodeAt(0);
07   $("input").trigger({'type':'keypress', 'charCode':chrCode});
08 }
```

7. Save all three files and then open the HTML document in a web browser, as shown in Figure 9.5. Type some text into the input and also click the `<span>` elements and notice that the result is the same.

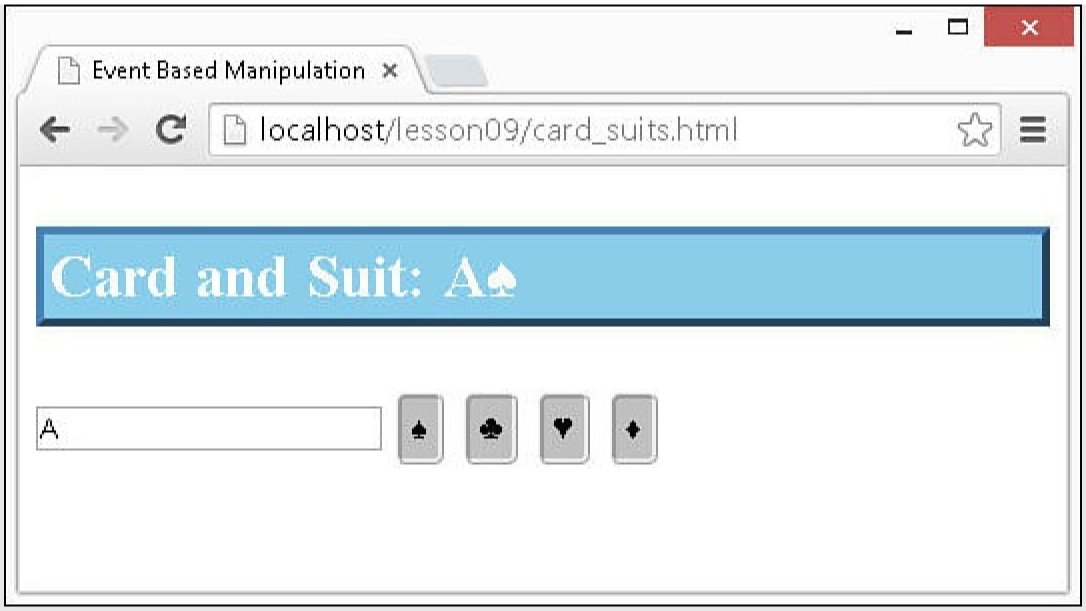


FIGURE 9.5 A simple web page that triggers the `keypress` event for the `<input>` element from the `<span>` elements' event handler.

LISTING 9.10 card_suits.html HTML File That Loads CSS and JavaScript and Defines the `<p>`, text `<input>`, and `<span>` Elements

[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Event Based Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/card_suits.js"></script>
    <link rel="stylesheet" type="text/css" href="css/card_suits.css">
  </head>
  <body>
    <p>Card and Suit: </p>
    <input type="text" />
    <span>&#9824;</span>
    <span>&#9827;</span>
    <span>&#9829;</span>
    <span>&#9830;</span>
  </body>
</html>
```

LISTING 9.11 card_suits.js JavaScript Code That Triggers the `keypress` Event for the `<input>` Element from the `<span>` Elements' Event Handler

Click here to view code image

```
function inputHandler(e){
  var chr = String.fromCharCode(e.charCode);
  $("p").append(chr);
}
function spanHandler(e){
  var chrCode = e.target.innerHTML.charCodeAt(0);
  $("input").trigger({'type':'keypress', 'charCode':chrCode});
}
$(document).ready(function(){
  $("input").keypress(function (e){inputHandler(e)});
  $("span").click(function (e){spanHandler(e)});
});
```

LISTING 9.12 card_suits.css CSS Code That Styles the `<span>` and `<p>` Elements

Click here to view code image

```
span{
  border-radius:5px;
  margin:3px;padding:5px;
  background-color:#C0C0C0;
  border:3px ridge;
  display:inline-block;
  cursor:pointer;
}
p{
  border:4px outset steelblue;
  padding:3px;
  color:white;background-color:skyblue;
  font-size:30px;font-weight:bold;
}
```

Creating Custom Events

An extremely powerful aspect of the JavaScript and jQuery event handling mechanism is the capability to add custom events. This solves the problem that the built-in events are all tied to DOM elements and specific behaviors supported by the browser. What if you want an event for every time the number of items in an array is greater than 100?

Custom events enable you to extend the events that can be added to elements. Custom events do not directly correspond to the physical interaction with the user, but rather to events that happen in your code based on the interactions that the user is performing.

The advantage of creating custom events is that you can use the already in-place event-handling system that allows you to add/remove events, bubble them up, and trigger them.

The following sections describe the different methods you can use in JavaScript and jQuery to add your own custom events.

Adding Custom Events Using JavaScript

Adding a custom event in JavaScript is similar to triggering an event in that you create a custom event object and then initialize it. The following code shows an example of creating a basic custom event:

Click here to view code image

```
var myEvent = document.createEvent("CustomEvent");
myEvent .initCustomEvent(
  "worldEnds",
  true,
  false,
  {
      'fire': false,
      'ice': false,
      'time': new Date()
  }
);
```

The first parameter defines the `type` name, the second sets the `bubbles` property to true, the third sets `cancelable` to false, and the fourth defines the `details` of the event. The values of `fire`, `ice`, and `time` will be available via the `details` attribute of the event object.

After the event has been created, you can trigger the event by calling the `dispatchEvent()` method on a DOM object. For example:

Click here to view code image

```
var obj = document.getElementById("#notify");
obj.dispatchEvent(myEvent);
```

Event handlers for the event can be added to DOM elements in the normal way using the `addEventListener()` method. For example:

Click here to view code image

```
document.addEventListener("worldEnds ", endOfWorldHandler, false);
```

Adding Custom Events Using jQuery

Adding custom events in jQuery is a simple process. It is a matter of calling `$.event.trigger()` and passing in the new event object. You must specify a type attribute for the event object, but after that, you can define whatever default values for the event object that you would like. For example, the following code defines a new event:

Click here to view code image

```
var custEvent = $.Event("worldEnds", {
  fire: false,
  ice: false,
  time: new Date()
});
```

The next line of code triggers the event in the event system:

```
$.event.trigger(custEvent);
```

After the event has been added to the system, you can attach event handlers for it. For example:

Click here to view code image

```
$(document).on("worldEnds", endOfWorldHandler);
```

Implementing Callbacks

Although they are not really events, callbacks have some similarities. A callback is a function that can be registered in jQuery and then fired off at specific times from your code. When a callback list is fired, each of the functions are executed in order.

An example where callbacks are useful is if you want users to have the option to specify that they want a notification when a background process, such as downloading data via AJAX, completes. If a user selects the option to enable the notification, a callback can be added. Then, anytime a download completes, the callback function will execute and display a notification to the user.

Understanding the Callback Mechanism

The callback mechanism is used by calling `$.Callbacks(flags)` to create a `callbacks` object. The purpose of the `flags` attribute is to provide the capability to specify the behavior that should occur when different callback functions are executed. The possible values for flags are as follows:

- **once**—Functions added are fired only once.
- **memory**—As new functions are added, they are fired right away with the same values as the last time callbacks were fired.
- **unique**—Allows the callback functions to be added only once.
- **stopOnFalse**—Stops firing other callback functions if one of the functions fired returns a false.

The `callbacks` object supports adding functions using the `add(functionName)` method and removing functions using the `remove(functionName)` method. To fire the callback list, call the `fire()` method. You can also disable the list using the `disable()` method.

The simplest way to understand implementing callbacks is to look at some examples. The following code defines a list of callbacks that must have unique function names and that also stop if one of the functions returns false:

Click here to view code image

```
function functionA(){return true;}
function functionB(){return true;}
function functionC(){return true;}
var callbacks = $Callbacks("unique stopOnFalse");
callbacks.add(functionA);
callbacks.add(functionB);
callbacks.fire()
callbacks.remove(functionB);
callbacks.add(FunctionC);
callbacks.fire();
callbacks.disable();
callbacks.fire();
```

The first time that `callbacks` is fired, `functionA` and `functionB` are executed. The second time, `functionA` and `functionC` are executed because `functionB` was removed. The third time, no functions are executed because the list has been disabled.

Using Deferred Objects

jQuery provides an intriguing option to implement callbacks in a different way, the deferred object. A deferred object is an object that contains a set of functions that can be run at a later time. The actual arguments passed to the function are not added until the deferred object is resolved and the functions are executed.

This allows you to apply a set of functions to a single object that can be applied at any time.

The following code illustrates creating a simple deferred object with multiple functions and then executing those functions:

Click here to view code image

```
var resultString = "";
function function1(n1, n2, p3) { resultString += "Problem: "; }
function function2(n1, n2) { resultString += n1 + " + "; }
function function3(n1, n2) { resultString += n2 + " = "; }
var deferredObj = $.Deferred();
deferredObj.done( [function1, function2], function3).done(function(n1, n2)
{
    resultString += n1 + n2;
    $("div").append(resultString);
  });
deferredObj.resolve(5,6);
```

This is what is happening in the preceding code: The three defined functions all add to the `resultString` variable. A new deferred object named `deferredObj` is created. The `.done(functions [, functions])` method is called to add the functions to the deferred object. All three ways were used to add functions. The first argument is a list of functions to execute, the second is a single function, and the third is an anonymous function.

The functions are not executed until `.resolve(args)` is called. Notice that the arguments of `.resolve()` match those required by the functions. Each function is executed in order. The result is that the following string is appended to the `<div>` element:

```
Problem: 5 + 6 = 11
```

Summary

This lesson focused on the different ways to implement event handling in web pages. Event handling is at the heart of interactive web pages. JavaScript and jQuery differ in how you create and add events to DOM elements. You learned the methods used to add event handlers in both.

You also learned how to access the event object and pass your own custom objects into event handlers, and you learned how to create and trigger your own custom events.

Q&A

Q. When is it better to use JavaScript to create and trigger custom functions rather than jQuery?

A. Never. Okay, not necessarily. jQuery has a lot more cross-browser support right now for events. Also, it is much easier to create the objects necessary. For those reasons, you should almost always use jQuery. If jQuery isn't an option for your environment because of backward compatibility or the like, then use JavaScript.

Q. Can't you do everything with custom events that you can do with callbacks?

A. Not very easily. Remember that you can create as many callback lists as you want with different flags and only fire off the functions tied to a specific list. Trying to manage all of that in event handlers would get out of hand pretty quickly

Workshop

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

Quiz

1. What HTML attribute of the `<body>` tag can be used to call JavaScript code when the page has finished loading?
2. True or false: An event will trigger the hander function only for the target element in which it occurs.
3. When would you use the `.ready()` method rather than the `.load()` method?
4. True or false: Mouse events can be triggered only by mouse movement or mouse clicks.

Quiz Answers

1. `onload`.
2. False. Most events will bubble up through ancestor objects all the way through the document object.
3. `.ready()` is used when you need only the DOM to be loaded. `.load()` is used if you also require other resources, such as images, to be finished loading as well.
4. False. Mouse events can be manually triggered in both jQuery and JavaScript.

Exercises

1. Extend the example in Listings 9.10, 9.11, and 9.12 to add additional buttons for other nonstandard characters. You will need to get the Unicode character codes from the Web.
2. Extend the example in Listings 9.7, 9.8, and 9.9 to include another button that will trigger the events for all check boxes.

Lesson 10. Dynamically Accessing and Manipulating Web Pages with JavaScript and jQuery

What You'll Learn in This Lesson:

- Tracking the mouse position and other information
- Manipulating the size and other attributes of page elements
- How to show and hide elements dynamically
- Layering page elements
- Dynamically creating DOM elements and adding them to web pages
- Adding and removing classes in jQuery
- Modifying the layout of the web page

This lesson is a big one. So far you have been given most of the tools that you will need to implement highly dynamic and interactive web pages. In this lesson, you take the next step and learn how to use those tools and some new ones to access information about web page elements. Then you use that information to manipulate and arrange them.

The ability to access and modify the properties and values of the different page elements will allow you to change basic web pages into true web applications.

You will learn in this lesson that the JavaScript and jQuery methods for web page manipulation are wide open. You can manipulate any page element from anywhere in your JavaScript code. This has the advantage of allowing you to easily transform pages with only a little bit of JavaScript/jQuery code. The disadvantage is that you need to be disciplined to keep your code easily maintainable so that you know where everything is being manipulated from. AngularJS has a much different approach that you will learn in the AngularJS section of lessons. For now, enjoy the ride of fast and easy dynamic web page manipulation.

Accessing Browser and Page Element Values

One of the core fundamentals of both jQuery and JavaScript is access to the DOM objects that represent the elements on the web page. This section guides you through accessing property values, position, size, and other attributes of the elements, as well as showing you how to modify many of them.

Getting Mouse Position

Mouse movement is one of the most common web interactions that you will be dealing with. Often, you will need to make adjustments to elements on the page based on the position of the mouse.

The mouse position is accessible from the event object when a mouse event is triggered. Keep in mind that three mouse coordinates are specified by the event. Table 10.1 describes each of these.

Attribute	Description
`screenX`	Specifies the x coordinate based on the mouse position relative to the left edge of the screen, regardless of the position of the browser window.
`screenY`	Specifies the y coordinate based on the mouse position relative to the top of the screen, regardless of the position of the browser window.
`pageX`	Specifies the x coordinate based on the mouse position relative to the left edge of the web page, regardless of the horizontal scrolling position.
`pageY`	Specifies the y coordinate based on the mouse position relative to the top of the web page, regardless of the vertical scrolling position.
`clientX`	Specifies the x coordinate based on the mouse position relative to the left edge browser window view area.
`clientY`	Specifies the y coordinate based on the mouse position relative to the top edge browser window view area.

TABLE 10.1 Event Object Attributes That Specify the Mouse Position

The following code shows an example of getting the mouse coordinates from an event object named `e` and applying them to a string variable:

Click here to view code image

```
var screenPosition = e.screenX + ", " + e.screenY;
var pagePosition = e.pageX + ", " + e.pageY;
var browserPosition = e.clientX + ", " + e.clientY;
```

Getting and Setting Values

Several HTML elements, especially input elements, have values that are associated with them. It is a simple matter in both jQuery and JavaScript to access the values of elements.

In JavaScript, the value can be accessed directly by accessing the `value` attribute of the object. For example:

Click here to view code image

```
domObject.value = 5;
var value = domObject.value;
```

jQuery, on the other hand, provides the `.val()` method to retrieve and set the value. For example, the following code sets and then gets the value of a jQuery object representing the HTML `<input>` element with `id="textInput"`:

Click here to view code image

```
$("#textInput").val(5);
var value = $("#textInput ").val();
```

Getting and Setting Attributes and Properties in jQuery

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

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

Click here to view code image

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

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

Click here to view code image

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

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

Click here to view code image

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

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

Click here to view code image

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

Note

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

Getting and Setting CSS Properties

Another important aspect of dynamic web programming is the dynamic manipulation of CSS.

JavaScript enables you to access the CSS properties via the style attribute of the DOM object. For example, you can get the `color` and `background-color` CSS properties of an element from a DOM object using the following code. Notice that you need to use `style["background-color"]` syntax because `style.background-color` is not valid in JavaScript:

Click here to view code image

```
var domObj = document.getElementById("banner");
var color = domObj.style.color;
var color = domObj.style.["background-color"];
```

Another example is that you can set `position`, `top`, and `left` CSS properties for an element using the following code:

Click here to view code image

```
domObj.style.position = "absolute";
domObj.style.top = "100px";
domObj.style.left = "100px";
```

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

Click here to view code image

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

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

Click here to view code image

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

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

Click here to view code image

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

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

Getting and Setting Element Size

Another important aspect when dynamically working with HTML elements is the capability to get the element's size. jQuery makes this very simple. Table 10.2 shows the methods provided in the jQuery object that enable you to get the height and width of an element.

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

TABLE 10.2 jQuery Object Methods to Get and Set the Element Size

Caution

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

Getting and Setting Element Position

In addition to the size of HTML elements, you often need to get their position. When you're working with HTML elements, there are two types of positions. The first is the position relative to the full document. The second is the position relative to the HTML element that acts as an offset parent. Which element is the offset parent depends on the position settings in CSS.

jQuery provides the `.position([position])` method to get the position relative to the offset parent. The `.offset([position])` method provides the position relative to the document. Both of these methods can be called with no argument. They return a simple object with a `left` and `right` attribute that represent the number of pixels from the left and top of document when using `.position()` or offset parent when using `.offset()`. They can also be called with a simple `position` attribute, which is an object that has `left` and `right` attributes. When called with a `position` parameter, they will set the position of the element.

For example, the following code will get the number of pixels from the top and the left of the document and the offset parent to an element with `id="myElement"`. Notice that to get the statements on one line, you reference the `top` and `left` attributes directly after the call:

Click here to view code image

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

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

Click here to view code image

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

Accessing the Class

The simplest way to get the classes associated with an HTML element is to look at the `className` attribute of the DOM object. The `className` attribute is a string containing the names of the class(es) attached to the element. Keep in mind that an element may have more than one class name attached to it. In those cases, the names will be separated by a space.

The following example shows how to get the `className` attribute of an element with `id="mainList"` from jQuery:

Click here to view code image

```
var class = $("#mainList").get().className;
```

Getting Browser and Screen Size and Color Information

An important part of dynamically manipulating web pages is the capability to access information about the screen and browser. This section focuses on the size and color capability.

Getting the screen or browser window size allows you to adjust the size of HTML elements based on the available area on the screen or browser window.

You can access the screen size using the `Screen` object in JavaScript. The `Screen` object provides the `width` and `height` attributes that are the screen resolution dimensions. For example, the following code stores the screen `width` and `height` in variable:

Click here to view code image

```
var sWidth = screen.width;
var sHeight = screen.height;
```

To get the supported colors by the screen and thus the web browser, use the `colorDepth` attribute of the `Screen` object. This returns a value of the number of bits supported—for example, 8, 16, 24. Following is an example of getting the color depth supported by the screen:

Click here to view code image

```
var colorBits = screen.colorDepth;
```

Even if the screen is large, the browser window may not be taking up the full screen amount. Although the browser window provides scrollbars for web pages that exceed the size of the window, it often looks better to resize elements on the page or adjust the layout as the browser size changes.

The browser dimensions can be accessed from the `innerWidth`, `outerWidth`, `innerHeight`, and `outerHeight` attributes of the window object available in JavaScript. The `innerHeight` and `innerWidth` values refer to the actual area available for the web page. The `outerWidth` and `outerHeight` refer to the total area, including the browser menu, borders, and bars. The following is an example of getting the web page display area dimensions:

Click here to view code image

```
var pageWidth = window.innerWidth;
var pageHeight = window.innerHeight;
```

Try it Yourself: Getting Screen, Browser, Mouse, and Element Info Using jQuery and JavaScript

Now it's time to put everything together. In this example, you build a basic web page with some different HTML elements. One of the elements will be a `<div>` that displays screen, browser, mouse, and element information and is updated with each movement of the mouse. The code is shown in Listings 10.1, 10.2, and 10.3. It may look like a lot of code at first, but it really is pretty basic and will help you see how to get information from jQuery and JavaScript about the current state of things.

Use the following steps to implement the example:

1. In Eclipse, create the lesson10, lesson10/js, and lesson10/css folders.

2. Create the lesson10/web_page_manipulation.html, lesson10/js/web_page_manipulation.js, and lesson10/css/web_page_manipulation.css files.

3. Add the code shown in Listing 10.1 to the HTML file. You should recognize the HTML components. There are two main sections: the first part defines several types of elements for you to interact with, and the second part, the really long `<div>`, provides a place to display information about the mouse, screen, browser, and elements.

4. Add the CSS from Listing 10.3 to the CSS file. You should also recognize the CSS statements by now. Some of the lines in the CSS are combined to make it fit better in the book.

5. Open the web_page_manipulation.js file and the following `load()` function that adds event handlers for the `mousemove`, `mouseover`, `change`, `keypress`, and `resize` events. You will create these event handlers in later steps. The `mousemove` event handler is attached to the document to catch the mouse movements at the document level. The resize handler is attached to the window so that you can display info when the browser is resized:

Click here to view code image

```
$(window).load(function(){
  $(document).mousemove(mousePosition);
  $("*").mouseover(elementInfo);
  $("*").change(elementInfo);
  $("*").keypress(elementInfo);
  $(window).resize(windowResize);
});
```

6. Add the following handler that will be called on mouse movement. Notice that propagation is stopped so that you try to display the information only once. The `Screen` object is used to get and display the screen dimensions and color bits. The event object `e` is used to get and display the various mouse position coordinates:

Click here to view code image

```
function mousePosition(e){
  e.stopPropagation();
  $("#screenSize").html(screen.width + "x" + screen.height);
  $("#colors").html(screen.colorDepth+"bit");
12   $("#browserSize").html(window.innerWidth + "x" +
window.innerHeight);
13   $("#mousePosition").html("X:" + e.screenX + "  Y:" + e.screenY);
14   $("#pagePosition").html("X:" + e.pageX + "  Y:" + e.pageY);
15   $("#scrollPosition").html("X:" + e.clientX + "  Y:" + e.clientY);
16 }
```

7. Create the following functions that will display all the element information. Propagation is stopped. The contents of the `.infoContainer <span>` elements are removed by line 19. Using the event object, the `domObj` variable is assigned the `target` DOM object that the event occurred on, and `jObj` is set to the jQuery version of the `target` object:

Click here to view code image

```
17 function elementInfo(e){
18   e.stopPropagation();
19   $(".infoContainer span").html("");
20   var domObj = e.target;
21   var jObj = $(domObj);
...
33 }
```

8. Add the following statements that use the DOM and jQuery objects to display the `id`, `tag`, and `class` attributes:

Click here to view code image

```
22   $("#elementId").html(domObj.id);
23   $("#elementType").html(jObj.prop('tagName'));
24   $("#elementClass").html(domObj.className);
```

9. Add the following statements that display the element's size and position:

Click here to view code image

```
25   $("#elementSize").html(jObj.width() + "x" + jObj.height());
26   $("#elementPosition").html(jObj.offset().top + ", "
+  jObj.offset().left);
```

10. Add the following miscellaneous statements. Line 27 uses the `.css()` method to get the CSS color attribute, line 28 uses `.val()` to get the value of the element if there is one, line 30 gets the checked state of the element if available, and line 32 gets the `src` attribute of `<img>` elements:

Click here to view code image

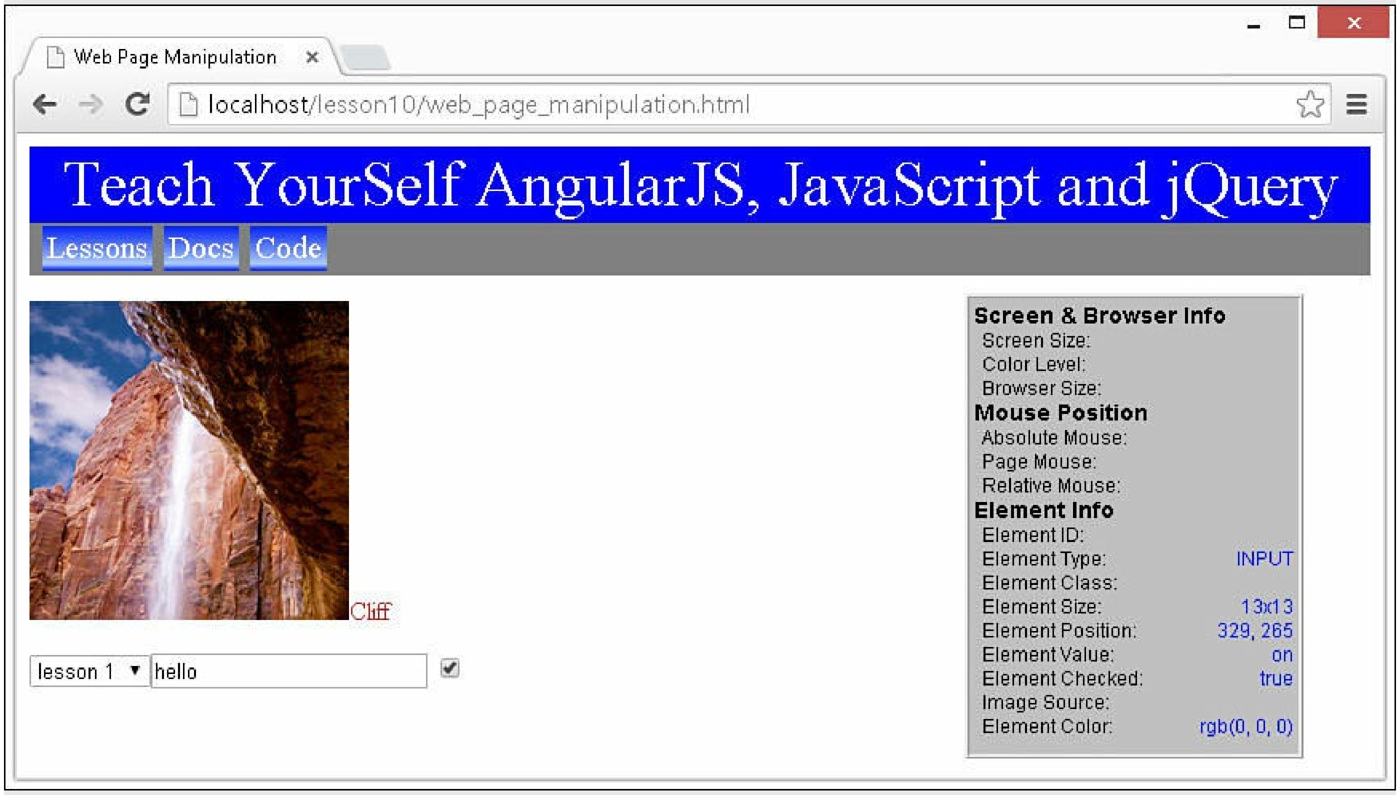
```
27   $("#elementColor").html(jObj.css("color"));
28   $("#elementValue").html(jObj.val());
29   try{
30     $("#elementChecked").html(jObj.prop('checked').toString());
31   } catch (e) {}
32   $("#elementSource").html(jObj.attr('src'));
```

11. Add the following handler that sets the browser window size each time you resize the browser:

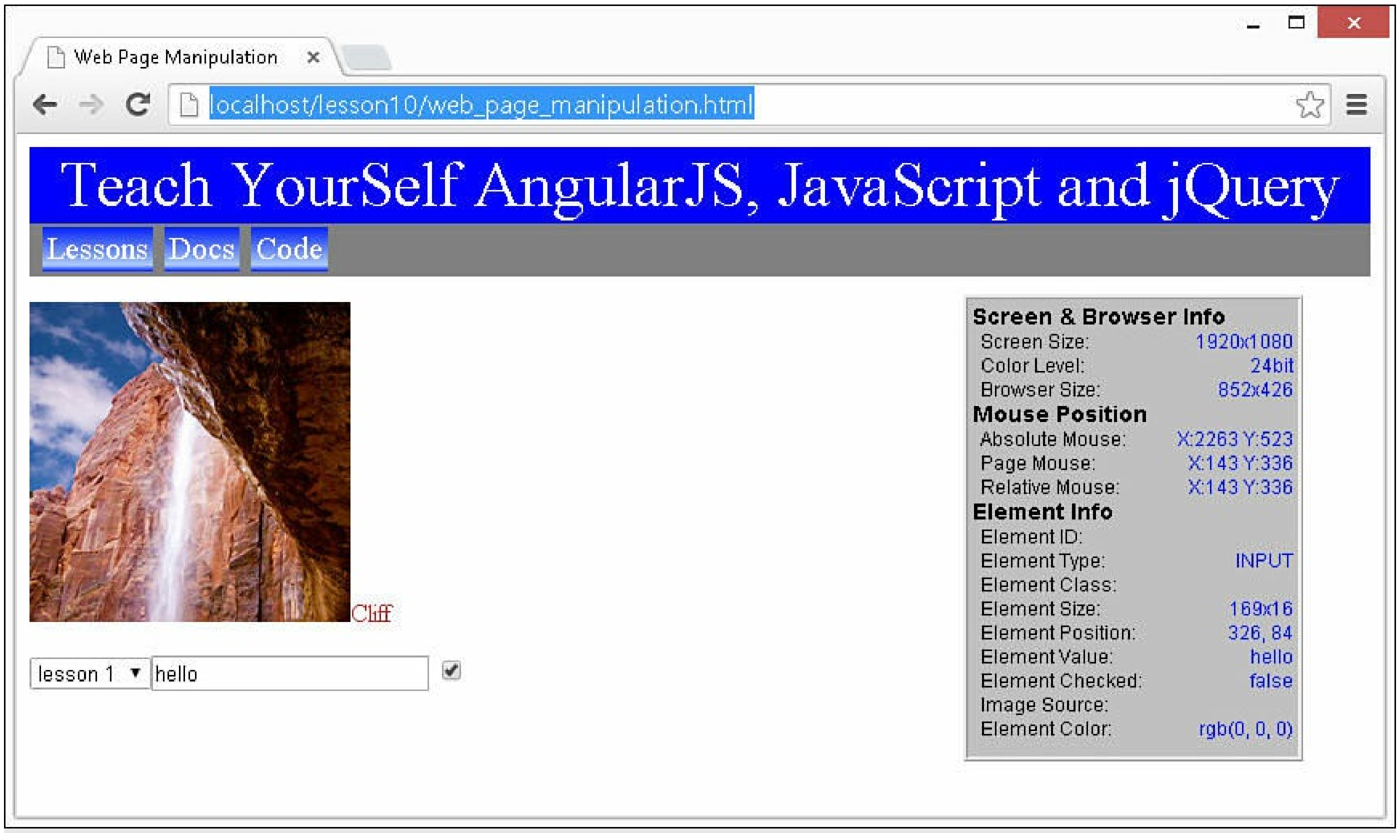
Click here to view code image

```
function windowResize(e){
  $("#browserSize").html(window.innerWidth + "x" +
window.innerHeight);
}
```

12. Save all three files and then open the HTML document in a web browser. Play around with the HTML elements and see the information change, as shown in Figure 10.1. Mouse coordinates, screen, and browser size, as well as element info, are displayed as available. Don’t forget to try resizing the browser and scrolling.


Web Page Manipulation
localhost/lesson10/web_page_manipulation.html
Teach YourSelf AngularJS, JavaScript and jQuery
Lessons
Docs
Code
Cliff
lesson 1
hello

```
Screen & Browser Info
 Screen Size:
 Color Level:
 Browser Size:
Mouse Position
 Absolute Mouse:
 Page Mouse:
 Relative Mouse:
Element Info
 Element ID:
 Element Type:              INPUT
 Element Class:
 Element Size:              13x13
 Element Position:       329, 265
 Element Value:                on
 Element Checked:            true
 Image Source:
 Element Color:      rgb(0, 0, 0)
```


Web Page Manipulation

localhost/lesson10/web_page_manipulation.html

Teach YourSelf AngularJS, JavaScript and jQuery

Lessons Docs Code

Cliff

lesson 1 hello

```
Screen & Browser Info
 Screen Size:            1920x1080
 Color Level:                24bit
 Browser Size:             852x426
Mouse Position
 Absolute Mouse:      X:2263 Y:523
 Page Mouse:           X:143 Y:336
 Relative Mouse:       X:143 Y:336
Element Info
 Element ID:
 Element Type:               INPUT
 Element Class:
 Element Size:              169x16
 Element Position:          326, 84
 Element Value:              hello
 Element Checked:            false
 Image Source:
 Element Color:       rgb(0, 0, 0)
```

FIGURE 10.1 A web page that displays the screen, browser, mouse, and element info as you play around with the web page.

LISTING 10.1 web_page_manipulation.html HTML File That Provides Several Elements to Play with, as Well as an Independent `<div>` That Displays Info

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Web Page Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/web_page_manipulation.js"></script>
     <link rel="stylesheet" type="text/css" href="css/web_page_manipulation.css">
  </head>
  <body>
    <div>
      <div id="banner" class="header">
        Teach YourSelf AngularJS, JavaScript and jQuery</div>
      <div id="menu" class="menu">
        <span class="menuItem">Lessons</span>
        <span class="menuItem">Docs</span>
        <span class="menuItem">Code</span>
      </div>
      <div id="content">
        <p class="imageArea">
          <img id="Cliff" src="/images/cliff.jpg" width="200px"/>
          Cliff</p>
        <select>
          <option value=1>lesson 1</option>
          <option value=2>lesson 2</option>
          <option value=3>lesson 3</option>
          <input type=text />
        </select>
        <input type="checkbox" />
      </div>
    </div>
    <div class=infoContainer>
      <p>Screen & Browser Info</p>
      <div>Screen Size: <span id="screenSize"></span></div>
      <div>Color Level: <span id="colors"></span></div>
      <div>Browser Size: <span id="browserSize"></span></div>
      <p>Mouse Position</p>
      <div>Absolute Mouse: <span id="mousePosition"></span></div>
      <div>Page Mouse: <span id="pagePosition"></span></div>
      <div>Relative Mouse: <span id="scrollPosition"></span></div>
      <p>Element Info</p>
      <div>Element ID: <span id="elementId"></span></div>
      <div>Element Type: <span id="elementType"></span></div>
      <div>Element Class: <span id="elementClass"></span></div>
      <div>Element Size: <span id="elementSize"></span></div>
      <div>Element Position: <span id="elementPosition"></span></div>
      <div>Element Value: <span id="elementValue"></span></div>
      <div>Element Checked: <span id="elementChecked"></span></div>
      <div>Image Source: <span id="elementSource"></span></div>
      <div>Element Color: <span id="elementColor"></span></div>
    </div>
  </body>
</html>
```

LISTING 10.2 web_page_manipulation.js jQuery and JavaScript Code That Retrieves and Displays Information About the Screen, Browser, Mouse, and HTML Elements

Click here to view code image

```
$(window).load(function(){
  $(document).mousemove(mousePosition);
  $("*").mouseover(elementInfo);
  $("*").change(elementInfo);
  $("*").keyup(elementInfo);
  $(window).resize(windowResize);
});
```

```
function mousePosition(e){
  e.stopPropagation();
  $("#screenSize").html(screen.width + "x" + screen.height);
  $("#colors").html(screen.colorDepth+"bit");
  $("#browserSize").html(window.innerWidth + "x" + window.innerHeight);
  $("#mousePosition").html("X:" + e.screenX + "  Y:" + e.screenY);
  $("#pagePosition").html("X:" + e.pageX + "  Y:" + e.pageY);
  $("#scrollPosition").html("X:" + e.clientX + "  Y:" + e.clientY);
}
function elementInfo(e){
  e.stopPropagation();
  $(".infoContainer span").html("");
  var domObj = e.target;
  var jObj = $(domObj);
  $("#elementId").html(domObj.id);
  $("#elementType").html(jObj.prop('tagName'));
  $("#elementClass").html(domObj.className);
  $("#elementSize").html(jObj.width() + "x" + jObj.height());
  $("#elementPosition").html(jObj.offset().top + ", "
+ jObj.offset().left);
  $("#elementColor").html(jObj.css("color"));
  $("#elementValue").html(jObj.val());
  try{
    $("#elementChecked").html(jObj.prop('checked').toString());
  } catch (e) {}
  $("#elementSource").html(jObj.attr('src'));
}
function windowResize(e){
  $("#browserSize").html(window.innerWidth + "x" + window.innerHeight);
}
```

LISTING 10.3 web_page_manipulation.css CSS Code That Styles the `<div>` and Other Elements

Click here to view code image

```
.infoContainer{
  border:3px ridge;
  padding:3px;
  position:fixed; display:inline-block;
  top:100px; right:50px;
  background-color:#C0C0C0;
  width:200px; height:280px;
  font:12px arial;
}
.infoContainer div{ margin-left:5px; }
.infoContainer div span{
  color:blue;
  float:right; }
.infoContainer p{
  margin:0px; padding:0px;
  font-size:14px; font-weight:bold;
 }
#banner{
  height:100;
  color:white; background-color:blue;
  font-size:40px; text-align:center;
}
#menu{ background-color:gray; padding:5px; }
.menuItem{
  padding:3px; margin-left:3px;
  background-image: -moz-linear-gradient(top, #0022ff 0%, #AACCFF 85%,
#0022ff 100%);
  background-image: -webkit-linear-gradient(top, #0022ff 0%, #AACCFF
85%, #0022ff 100%);
  background-image: -ms-linear-gradient(top, #0022ff 0%, #AACCFF 85%,
#0022ff 100%);
  font:20px bold;
  color:white;
}
.imageArea{color:firebrick;}
```

Dynamically Manipulating Page Elements

In the previous section, you not only learned how to access the components, but also how to change values, attributes, properties, and CSS settings. This section extends those concepts by discussing some additional ways to dynamically manipulate the web page by adding and removing elements, changing classes, and toggling visibility.

Adding Page Elements Dynamically

Often, you will not know all the elements that belong on a web page until a user logs in, or you receive addition information from a web service, or some other interaction. In those cases, you must be able to add elements on-the-fly in your jQuery and JavaScript code.

Adding Page Elements in JavaScript

There are several ways to add HTML elements dynamically. The most basic is to set the `innerHTML` attribute of a container object to an HTML string. For example, the following code sets the contents of an existing object to a new `<p>` element:

Click here to view code image

```
domObj.innerHTML = "<p>Paragraph 1 goes here</p>";
domObj.innerHTML += "<p>Paragraph 2 goes here</p>";
```

The problem with this method is that it lends itself to really ugly string statements that will likely become a major headache later. So JavaScript provides a way to create the

DOM objects and then append them to the parent object.

The `document.createElement(tag)` method allows you to create an element object, and the `document.createTextNode(text)` allows you to create the text node that is part of the element. Then the `appendChild(object)` method can be called on the DOM objects to append the newly created elements and nodes.

To illustrate this, check out the following code that adds a couple of paragraphs to an existing object named `domObj`:

Click here to view code image

```
var newP = createElement("p");
var newT = createTextNode("Paragraph 1 goes here");
newP.appendChild(newT);
domObj.appendChild(newP);
```

It takes a bit more code, but the flow is much safer. Plus, the upside of creating objects is that you have an actual DOM object that you can add additional values to the element. For example, the following code allows you to add an `<img>` element and set the `src` and `height` attributes before appending it to the existing object:

Click here to view code image

```
var newImg = createElement("img");
newImg.src = "images/sunset.jpg";
newImg.height = 200;
domObj.appendChild(newImg);
```

Adding Page Elements in jQuery

Now you can look at how to add new elements in jQuery. You can apply the same `innerHTML` shortcut in jQuery as in JavaScript by using the `.html()` method, which will get or set the `innerHTML` string. For example:

Click here to view code image

```
$("#myDiv").html("<p>Paragraph 1 goes here</p>");
```

This method, however, has the same limitations and should be used sparingly. The better method is to create a new jQuery object and append it. The following code takes you through that process. The first step is to create the object by passing the tag name or HTML string to the jQuery object `$`. For example, the following statement creates a new jQuery object with one `<p>` element in the set:

```
var newP = $("<p></p>");
```

You can then add text to the paragraph using the following:

Click here to view code image

```
newP.html("Paragraph 1 goes here");
```

Then you can append an element to one or more existing elements using `.append(jQueryObject)`. For example, the following code adds the paragraph to all `<div>` elements:

```
$("div").append(newP);
```

To illustrate this, the following code creates a new jQuery object with an `<img>` element in the set and adds it to all `<li>` elements:

Click here to view code image

```
var newImg = $("<img></img>");
newImg.attr("src", "images/sunset.jpg");
newImg.height(30);
$("li").append(newImg);
```

jQuery also provides the `.appendTo(target)` method, which allows you to append an object to another object. This works the same way as `.append()` but in reverse. The method is called from the child object and not the new parent. For example, the `newImg` object from the preceding example could be appended by the following statement instead:

```
newImg.appendTo("li");
```

Removing Page Elements

Page elements can be removed in a couple of ways. The most basic way is to get the parent object and then set `domObj.innerHTML = ""` for DOM objects or call `jObj.html("")` for jQuery objects. This erases all content inside the parent element and thus removes any child elements.

Try to design your html components so that you can remove elements using the `.html("")` method because it keeps the clean-up code so simple. Another advantage is that adding new elements to the container is easier because you don't have to deal with existing elements.

In JavaScript, you can also call the `.removeElement(child)` on the parent element. For example, the following code gets an element with `id="container"` and then removes a child with `id="paragraphA"`:

Click here to view code image

```
var parent=document.getElementById("container");
var child=document.getElementById("paragraphA");
child.parentNode.removeChild(child);
```

jQuery provides two methods to remove elements. The first is `.empty()`, which is equivalent to `.html("")`. With `.empty()`, all child elements and text will be removed. The second method is `.remove([selector])`, which removes elements based only on the original query and an optional `selector`.

If no selector is specified, the elements from the original query are removed. For example, to remove all `<div>` elements, use the following statement:

```
$("div").remove()
```

If a selector is specified, child elements from the original query that match the selector will be removed. For example, to remove the `<p>` elements inside `<div>` elements, use the following:

```
$("div").remove("p");
```

jQuery provides one additional method that is useful when removing elements: the `.detach([selector])` method. The detach method works the same way as the `.remove([selector])` method, with one important difference. The actual element DOM data is not deleted, even though the elements are removed from the parents. You still have the elements in the existing jQuery object and can insert them back into the DOM at another location.

For example, the following code detaches all paragraphs from one element and appends them to another:

Click here to view code image

```
var ps = $("#div1").detach("p");
ps.appendTo("#div2");
```

Replacing Elements in jQuery

As you have seen in the previous sections, you can easily add and remove elements via jQuery. You also need to be aware of the capability to replace existing sets of elements with other sets of elements. This is required in jQuery because after you remove the elements, you will no longer be able to perform the same queries because the elements will be gone.

You can use three methods to replace elements in jQuery. The simplest is to use `.html()`. The `.html()` method in jQuery is extremely useful for replacing the contents of an existing element with completely different content. The `.html()` method accepts a string or an object and replaces the content of the set of elements in the jQuery with the object or string. For example, the following statement replaces the contents of all `<div>` elements with a new paragraph:

Click here to view code image

```
$("div").html($("<p>New Paragraph</p>"));
```

Another method of replacing a set of elements in the document with new content is the `.replaceAll(target)` method. This method replaces the set of elements that match the `target` selector with those of the current set. For example, to replace all `<div>` elements in a `parentB` with `<span>` elements from `parentA`, you could use the following:

Click here to view code image

```
$("#parentA span").replaceAll("#parentB div");
```

The final method is to use `.replaceWith(newContent)`, which does the opposite of `.replaceAll()`. The `.replaceWith()` function replaces the elements in the current set with the content specified. For example, to replace all `<div>` elements with a single new blank `<div>`, you could use the following:

Click here to view code image

```
$("div").replaceWith($("<div></div>"));
```

Inserting Elements in jQuery

Another important feature of jQuery is the capability to easily insert elements into existing content. You have already seen how to append items to the end; however, what if you want to put content into the middle? That is where the `.before()` and `.after()` methods come in handy.

The `.after(content [,content])` method allows you to specify an element that should be inserted after each element in the current jQuery object's set. For example, to insert a new paragraph after the third `<p>` element in the document, you would use the following:

Click here to view code image

```
$("p:eq(2)").after($("<p>New Fourth Paragraph</p>"));
```

The `.before(content [,content])` method allows you to specify an element that should be inserted before each element in the current jQuery object's set. For example, to insert a new paragraph before the third `<p>` element in the document, you would use the following:

Click here to view code image

```
$("p:eq(2)").before($("<p>New Third Paragraph</p>"));
```

Note

Both the `.after()` and `.before()` methods allow you to pass in multiple objects to insert, separated by a comma. This enables you to prepare sets of objects and then insert them all together in the correct spot.

Changing Classes

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

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

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

Click here to view code image

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

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

Click here to view code image

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

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

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

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

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

Click here to view code image

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

Toggling Visibility

A simple way of changing the look and feel of web pages is to toggle the visibility of elements. You can do this from JavaScript by setting the `style.display` property to `"none"` or to `""`; however, jQuery provides much more elegant and extensible solutions.

To display an element using jQuery, call the `.show()` method on that object. Then to hide the element, use the `.hide()` method. It's as simple as that. For example, to hide an object name `jObj`, use the following statement:

```
jObj.hide();
```

To display it again, use the following:

```
jObj.show();
```

One problem with `.hide()` is that after it is applied, the element will no longer take up any page space. This may be the way that you want it, or in some instances, you may want only the element to be invisible, but still take up space. To make an element invisible, set the `opacity` CSS property to 0. For example:

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

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

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

Tip

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

Try it Yourself: Dynamically Manipulating Web Page Elements

Now it’s time to put everything together again. In this example, you start with a basic web page and then dynamically add, modify, and remove elements based on user interactions, as shown in Figure 10.2. The purpose of this example is to get you going on adding content dynamically, as well as accessing properties of existing elements and manipulating the content, visibility, and classes.

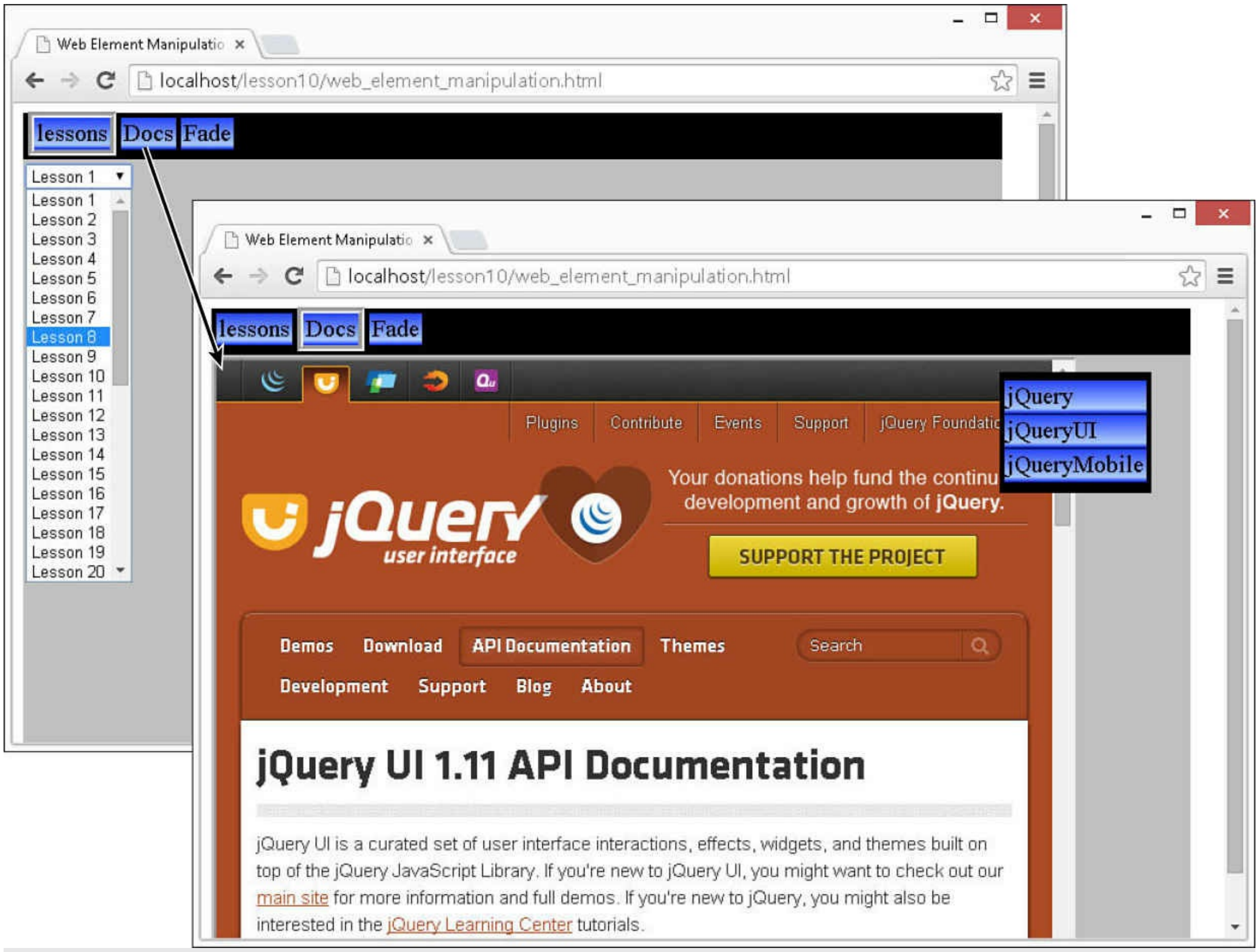

Web Element Manipulatio
localhost/lesson10/web_element_manipulation.html
lessons
Docs
Fade
Lesson 1
Lesson 1
Lesson 2
Lesson 3
Lesson 4
Lesson 5
Lesson 6
Lesson 7
Lesson 8
Lesson 9
Lesson 10
Lesson 11
Lesson 12
Lesson 13
Lesson 14
Lesson 15
Lesson 16
Lesson 17
Lesson 18
Lesson 19
Lesson 20
jQuery
jQueryUI
jQueryMobile
Plugins
Contribute
Events
Support
jQuery Foundatic
Your donations help fund the continu
development and growth of jQuery.
SUPPORT THE PROJECT
jQuery user interface
Demos
Download
API Documentation
Themes
Search
Development
Support
Blog
About
jQuery UI 1.11 API Documentation
jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library. If you're new to jQuery UI, you might want to check out our main site for more information and full demos. If you're new to jQuery, you might also be interested in the jQuery Learning Center tutorials.

FIGURE 10.2 A web page that dynamically builds the elements displayed in the left pane and then changes the content based on user selections.

The code is shown in Listings 10.4, 10.5, and 10.6. After you grasp this example, it should open up your mind to more elaborate and rich implementations. Use the following steps to implement the example:

1. In Eclipse, create the lesson10/web_element_manipulation.html, lesson10/js/web_element_manipulation.js, and lesson10/css/web_element_manipulation.css files.
2. Add the code shown in Listing 10.4 to the HTML file. You should recognize the HTML components. There is a set of `<span>` elements that will be buttons, a content `<div>` where you will dynamically place new elements, and a free-floating menu that provides buttons with links to jQuery docs.
3. Add the CSS from Listing 10.6 to the CSS file. There is quite a bit of CSS code to style the different elements. However, all of it should be familiar to you by now.
4. Open the web_element_manipulation.js file and add the following `load()`

function. Line 2 hides the `docMenu` div, which will be shown only when the user clicks the Docs button. The rest of the lines add click handlers for the buttons:

Click here to view code image

```
01 $(window).load(function(){
02   $("docMenu").hide();
03   $("#lessons").click(setLessonNav);
04   $("#docs").click(setDocNav);
05   $("#fade").click(fade);
06 });
```

5. Add the code shown in lines 7 through 22. This provides two functions. The `setLessonNav()` function is called when the user clicks the Lessons button. This function goes through the process of creating a new `<select>` element and adding 24 `<option>` elements to it. On line 16, a `change` event handler is added to the `<select>` element to catch when the lesson changes. The second function sets the content paragraph element to match the lesson selected:

Click here to view code image

```
24   $("#content").append(select).append("<br><p></p>");
```

6. Add the following function that will be called when a user clicks the Docs button. This function shows the Doc menu, removes the active class from all `<span>` elements, and then adds it to the Docs button to show it is active:

Click here to view code image

```
27 function setDocNav(){
28   $("docMenu").show();
29   $("span").removeClass("active");
30   $("#docs").addClass("active");
31 }
```

7. Add the following handler for when the user clicks one of the options in the Doc menu. This function creates a new `<iframe>` element that points to the selected doc and then replaces the contents of the #content `<div>` with that new element:

Click here to view code image

```
32 function setDoc(doc){
33   var frame = $("<iframe></iframe>");
34   frame.attr("src", doc);
35   $("#content").html(frame);
36 }
```

8. Add the following handler for when the user clicks the Fade button. This checks the current `opacity` CSS property and increases it or decreases it to cause the content to fade in and out:

Click here to view code image

```
function fade(){
  var opacity = $("#content").css("opacity");
  if (opacity < 1){ $("#content").css("opacity", 1);}
  else { $("#content").css("opacity", .5); }
}
```

9. Save all three files and then open the HTML document in a web browser. Play around with the Lessons and Docs buttons to see the elements dynamically displayed. Try the Fade button to fade the content in and out.

LISTING 10.4 web_element_manipulation.html HTML File Basic Web Page Used in the Example

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Web Element Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/web_element_manipulation.js"></script>
    <link rel="stylesheet" type="text/css" href="css/web_element_manipulation.css">
  </head>
  <body>
    <div id="container">
      <div id="menu" class="menu">
        <span id="lessons" class="menuItem">lessons</span>
        <span id="docs" class="menuItem">Docs</span>
        <span id="fade" class="menuItem">Fade</span>
      </div>
      <div id="content"></div>
    </div>
    <div id="docMenu">
      <span onclick="setDoc('http://api.jquery.com/')">jQuery</span>
      <span onclick="setDoc('http://api.jqueryui.com/')">jQueryUI</span>
      <span onclick="setDoc('http://jquerymobile.com/demos/1.4.0/')">jQueryMobile</spar
    </div>
  </body>
</html>
```

LISTING 10.5 web_element_manipulation.js jQuery and JavaScript Code That Dynamically Builds the Left Navigation Items Based on the Button Clicked in the Top Menu

Click here to view code image

```
$(window).load(function(){
  $("#docMenu").hide();
  $("#lessons").click(setHourNav);
  $("#docs").click(setDocNav);
  $("#fade").click(fade);
});
function setHour(e){
  var hour = $("#lessonSelect").val();
  $("#content p").html("Lesson "+ hour);
}
function setHourNav(){
  $("#docMenu").hide();
  $("span").removeClass("active");
  $("#lessons").addClass("active");
  var select = $('<select id="lessonSelect"></select>');
  select.change(setHour);
  for(var x=1; x<41; x++){
    var option = $("<option></option");
    option.val(x);
    option.html("Lesson "+x);
    select.append(option);
  }
  $("#content").html("");
  $("#content").append(select).append("<br><p></p>");
  setLesson(5);
}
function setDocNav(){
  $("#docMenu").show();
  $("span").removeClass("active");
  $("#docs").addClass("active");
}
function setDoc(doc){
  var frame = $("<iframe></iframe>");
  frame.attr("src", doc);
  $("#content").html(frame);
}
function fade(){
  var opacity = $("#content").css("opacity");
  if (opacity < 1){ $("#content").css("opacity", 1);}
  else { $("#content").css("opacity", .5); }
}
```

LISTING 10.6 web_element_manipulation.css CSS Code That Styles the Banner, Buttons, and Other Elements

Click here to view code image

```
#banner{
  height:100px;
  color:white; background-color:blue;
  font-size:40px; text-align:center;
}
#menu, #docMenu{
  background-color:black;
  padding:6px 4px 9px 4px;
}
.menuItem, #docMenu span{
  padding:2px;
  background-image: -moz-linear-gradient(top, #2244ff 0%, #AACCFF 85%,
#0022ff 100%);
  background-image: -webkit-linear-gradient(top, #2244ff 0%, #AACCFF
85%, #0022ff 100%);
  background-image: -ms-linear-gradient(top, #2244ff 0%, #AACCFF 85%,
#0022ff 100%);
  font:20px bold;
  cursor:pointer;
}
.active{ border:5px groove; }
#docMenu span{ display:block; margin-top:1px; }
#content, iframe{
  display:inline-block;
  width:700px; height:500px;
}
#container{ width:800px; background-color:#C0C0C0}
#docMenu{
  position:fixed; right:60px; top:60px;
}
#content{
    padding:2px;
    color:blue;
    font-size:20px;
}
```

Dynamically Rearranging Elements on the Web Page

One of the coolest interactions that you can make with web pages is to rearrange elements based on user interaction. For instance, you can make elements bigger or smaller, or change the position. These were already covered as part of getting and setting element attributes in the first section of this lesson. This section builds further on those concepts by discussing a final way to position page elements using the z-index; then you step through an example that shows the different methods of rearranging the elements.

Adjusting the **z-index**

The `z-index` is a CSS property that specifies the position of an HTML element with respect to other elements, not vertically or horizontally but projected out toward the user, as if it were papers stacked on top of one another on the screen. The element with the highest `z-index` is displayed on top of other elements when the page is rendered by the browser.

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

Click here to view code image

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

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

Click here to view code image

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

Try it Yourself: Dynamically Rearranging Page Elements

In this example, you learn the process of using jQuery and JavaScript to move, resize, and rearrange images. You use images because they are one of most commonly rearranged elements. You can apply these same principles to any HTML element. The code for the example is shown in Listings 10.7, 10.8, and 10.9. Use the following steps to implement the example:

1. In Eclipse, create the lesson10/rearranging_elements.html, lesson10/js/rearranging_elements.js, and lesson10/css/rearranging_elements.css files.
2. Add the code shown in Listing 10.7 to the HTML file. You should recognize the HTML components. The HTML code contains several `<span>` elements that will be styled as buttons and three images that will be rearranged as the user clicks the buttons.
3. Add the CSS from Listing 10.9 to the CSS file. This code styles the buttons and puts a frame around the images.
4. Open the rearranging_elements.js file and the following global definitions, which will be used later to keep track of the starting coordinates when tiling and stacking images, as well as the current image with the top `z-index` and max image index:

```
01 var startX = startY = 60;
02 var topIndex, maxIndex;
```

5. Add the following `load()` function. Lines 3 and 4 set the current top index

and the max index to the number of files minus 1 to make it zero-based. Then lines 6–12 add click handlers for each of the buttons:

Click here to view code image

```
03 $(window).load(function(){
04   topIndex = $(".photo").length-1;
05   maxIndex = topIndex;
06   $("#right").click(function(e){move(e, "right");});
07   $("#left").click(function(e){move(e, "left");});
08   $("#bigger").click(function(e){resize(e, "bigger");});
09   $("#smaller").click(function(e){resize(e, "smaller");});
10   $("#stack").click(stack);
11   $("#tile").click(tile);
12   $("#flip").click(flip);
13   stack();
14 });
```

Note

You will need to use the `$(window).load()` event handler rather than the `$(document).ready()` for this exercise so the images are loaded for certain prior to running the `stack()` function.

6. Add the following resize handler that accepts a direction and uses the value to either enlarge or shrink the image by adjusting the width using the `width()` method. Notice that the `topIndex` variable is used to determine which image to adjust:

Click here to view code image

```
15 function resize(e, direction){
16   var img = $("img:eq(" + topIndex + ")");
17   if (direction == "bigger"){ img.width(img.width()+20); }
18   else { img.width(img.width()-20); }
19 }
```

7. Add the following move handler that adjusts the `offset()` of the image to the left or right based on the direction argument. Also `startX` and `startY` are adjusted to the new position:

Click here to view code image

```
20 function move(e, direction){
21   var img = $("img:eq(" + topIndex + ")");
22   var pos = img.offset();
23   if (direction == "right"){ pos.left += 10;}
24   else {pos.left -= 10;}
25   img.offset(pos);
26   startX = pos.left;
27   startY = pos.top;
}
```

8. Add the following `stack()` function that iterates through all the `.photo` elements and stacks them by adjusting the `top` and `left` offset values during each iteration:

Click here to view code image

```
function stack(){
  var x = startX,  y = startY;
  $(".photo").each(function(indx){
    $(this).offset({ top:y, left:x });
    x += 20;
    y += 20;
  });
}
```

9. Add the following `tile()` function that also iterates through all the `.photo` elements and places them next to each other, rotating to the next line if the current set has surpassed 400 pixels. Notice that I use several variables through each iteration to keep track of the max height and current `x` and `y` coordinates:

Click here to view code image

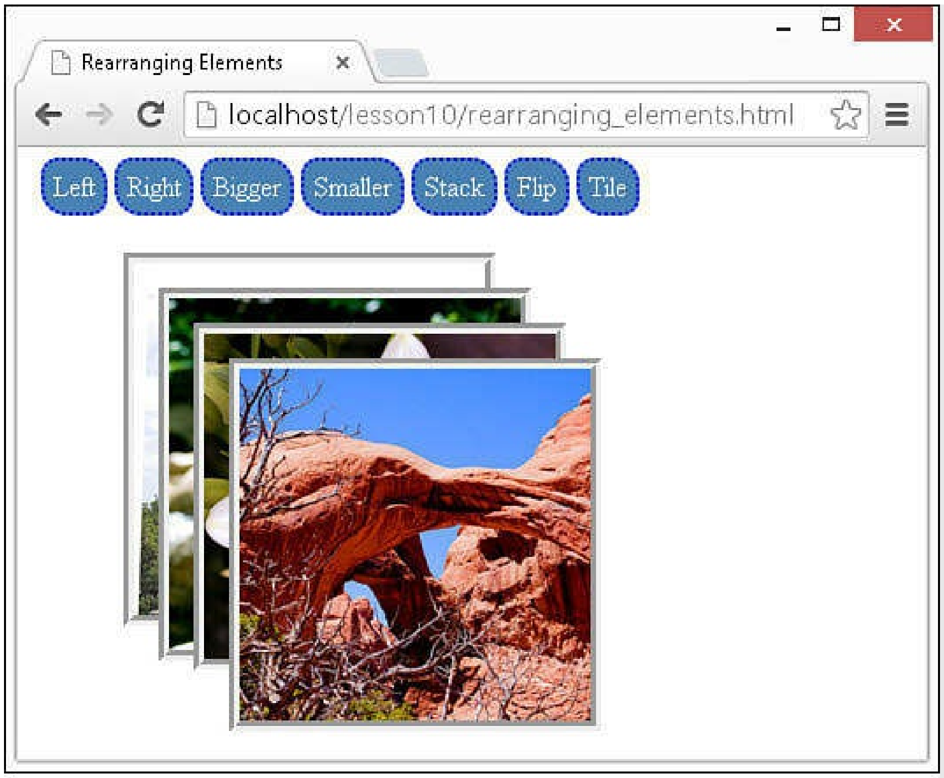
```
function tile(){
  var x = startX, y = currTop = startY;
  var maxH = 0;
  $(".photo").each(function(indx){
    maxH = Math.max(maxH, $(this).outerHeight());
    $(this).offset({ top:y, left:x });
    x += $(this).outerWidth();
    if (x > 400){
      y = currTop + maxH;
      x = startX;
      maxH = 0;
    }
  });
}
```

10. Add the `flip()` function that will adjust the `z-index` so that when images overlap each other, you can change which is on top:

Click here to view code image

```
function flip(){
  topIndex++;
  if (topIndex > $(".photo").length-1){ topIndex=0; }
  $(".photo").each(function(indx){
    if (indx <= topIndex){ z = maxIndex - (topIndex - indx); }
    else { var z = indx - topIndex - 1; }
    $(this).css("z-index", z);
  });
}
```

11. Save all three files and then open the HTML document in a web browser, as shown in Figure 10.3. Play around with the buttons and notice how the interactions of the images matches the rearrangements made by the click handler functions.


Rearranging Elements
localhost/lesson10/rearranging_elements.html
Left
Right
Bigger
Smaller
Stack
Flip
Tile

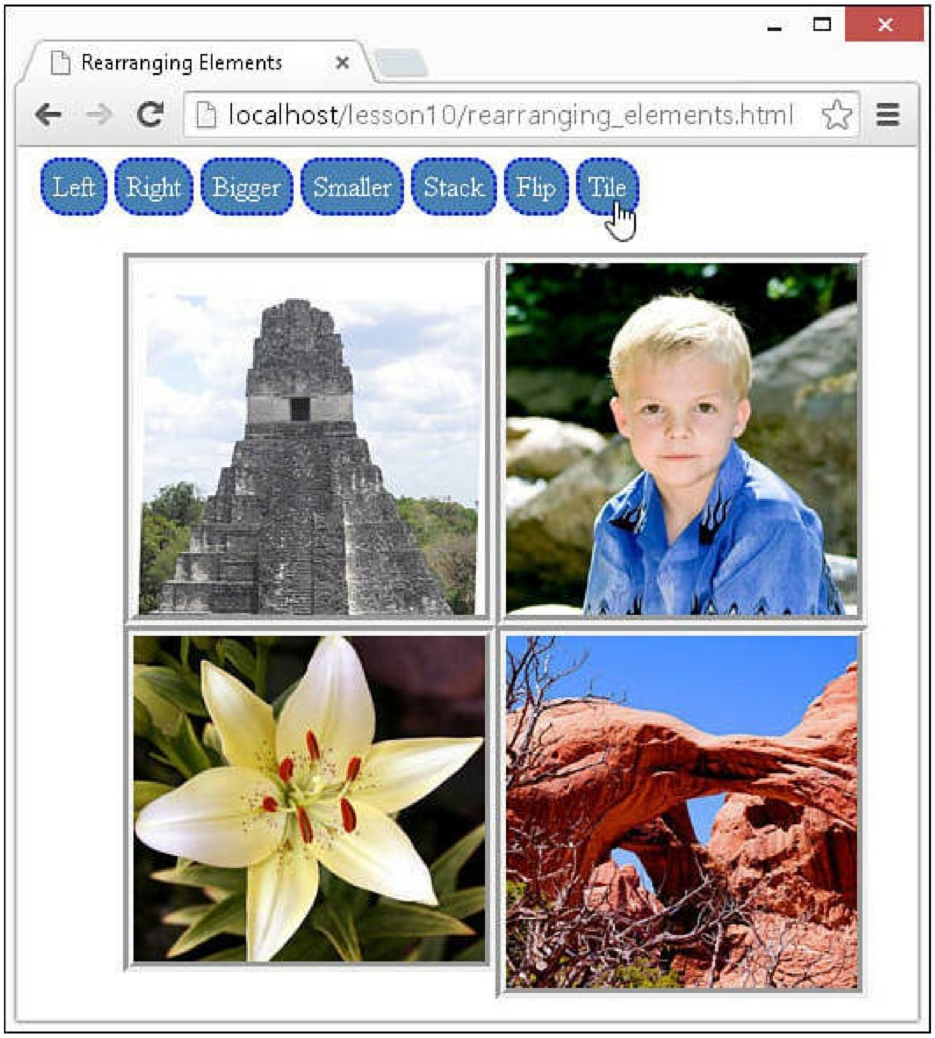

Rearranging Elements
localhost/lesson10/rearranging_elements.html
Left
Right
Bigger
Smaller
Stack
Flip
Tile

FIGURE 10.3 Using the `width()`, `offset()`, and `css("z-index")` methods on image objects; you can easily rearrange them on the web page.

LISTING 10.7 rearranging_elements.html HTML File Basic Web Page Used in the Example That Defines Several `<span>` Elements Used for Buttons and `<img>` Elements

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Rearranging Elements</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/rearranging_elements.js"></script>
    <link rel="stylesheet" type="text/css" href="css/rearranging_elements.css">
  </head>
  <body>
    <div id="container">
      <span id="left">Left</span>
      <span id="right">Right</span>
      <span id="bigger">Bigger</span>
      <span id="smaller">Smaller</span>
      <span id="stack">Stack</span>
      <span id="flip">Flip</span>
      <span id="tile">Tile</span>
      <div id="photos">
        <img class="photo" src="../images/pyramid.jpg" />
        <img class="photo" src="../images/boy2.jpg" />
        <img class="photo" src="../images/flower.jpg" />
        <img class="photo" src="../images/double.jpg" />
      </div>
    </div>
  </body>
</html>
```

LISTING 10.8 rearranging_elements.js jQuery and JavaScript Code That Dynamically Moves, Resizes, and Adjusts the `z-index` of Several `<img>` Elements

Click here to view code image

```
var startX = startY = 60;
var topIndex, maxIndex;
```

```
$(window).load(function(){
  topIndex = $(".photo").length-1;
  maxIndex = topIndex;
  $("#right").click(function(e){move(e, "right");});
  $("#left").click(function(e){move(e, "left");});
  $("#bigger").click(function(e){resize(e, "bigger");});
  $("#smaller").click(function(e){resize(e, "smaller");});
  $("#stack").click(stack);
  $("#tile").click(tile);
  $("#flip").click(flip);
  stack();
});
function resize(e, direction){
  var img = $("img:eq(" + topIndex + ")");
  if (direction == "bigger"){ img.width(img.width()+20); }
  else { img.width(img.width()-20); }
}
function move(e, direction){
  var img = $("img:eq(" + topIndex + ")");
  var pos = img.offset();
  if (direction == "right"){ pos.left += 10;}
  else {pos.left -= 10;}
  img.offset(pos);
  startX = pos.left;
  startY = pos.top;
}
function stack(){
  var x = startX,  y = startY;
  $(".photo").each(function(indx){
    $(this).offset({ top:y, left:x });
    x += 20;
    y += 20;
  });
}
function tile(){
  var x = startX, y = currTop = startY;
  var maxH = 0;
  $(".photo").each(function(indx){
    maxH = Math.max(maxH, $(this).outerHeight());
    $(this).offset({ top:y, left:x });
    x += $(this).outerWidth();
    if (x > 400){
      y = currTop + maxH;
      x = startX;
      maxH = 0;
    }
  });
}
function flip(){
  topIndex++;
  if (topIndex > $(".photo").length-1){ topIndex=0; }
  $(".photo").each(function(indx){
    if (indx <= topIndex){ z = maxIndex - (topIndex - indx); }
    else { var z = indx - topIndex - 1; }
    $(this).css("z-index", z);
  });
}
```

LISTING 10.9 rearranging_elements.css CSS Code That Styles the Buttons and Images

Click here to view code image

```
.photo{
  border:6px groove;
  width:200px;
  position:absolute; top:40px; left:20px;
}
span{
  padding:5px;
  background-color:steelblue; color:white;
  border-radius:10px 15px; border:2px dotted blue;
  cursor:pointer;
}
#container{ padding:5px; }
```

Summary

This lesson has covered a lot of ground. You already had the tools to understand the JavaScript code, CSS styling, and the various objects involved in jQuery and JavaScript. In this lesson, you learned how to access the attributes, properties, methods, and other parts of those objects and then modify them to apply interactivity and dynamics to web pages.

You also learned how to create HTML objects dynamically and add them to web pages, and how to remove and modify existing elements based on user interaction.

Q&A

Q. You showed how to add and remove classes in jQuery; is there a way to do the same in JavaScript?

A. Yes, but you probably shouldn't use it. The `className` attribute of the DOM object contains a space-separated list of classes. You can add a class by appending the new class name to that attribute—for example, `obj.className += " " + newClass;`. Removing the `className` is more difficult. You need to either use a regex statement or split the string, remove the class, and then rebuild it. These methods are a lot more risky than jQuery because you can end up mangling the string and then none of the classes will work.

Q. Is there a way to rotate an image element?

A. Yes, in some browsers. You can use the transform CSS property for Firefox and Chrome and the filter property for Internet Explorer. The following code illustrates rotating an image 90 degrees using jQuery in Firefox, IE, and Chrome:

Click here to view code image

```
$("img").css({
    "-webkit-transform": "rotate(90deg)",
    "-moz-transform": "rotate(90deg)",
    "filter":
"progid:DXImageTransform.Microsoft.BasicImage(rotation=1)"
});
```

Workshop

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

Quiz

1. How do you make an element disappear and yet keep taking up space in the web browser?
2. What CSS property allows you to specify which HTML element is displayed on top when two elements overlap?
3. What is the difference between `screenX`, `pageX`, and `clientX` properties of a mouse event?
4. What jQuery would you use to remove all `<p>` elements from an element with `id="container"`?
5. True or false: An HTML element can have only one class assigned to it at a time.

Quiz Answers

1. By setting the `opacity` CSS property to 0.
2. `z-index`
3. `screenX` is relative to the left edge of the `screen`, `pageX` is relative to the left edge of the document, and `clientX` is relative to the left edge of the browser window.
4. `$("#container p").remove();`
5. False. Elements can have many classes assigned to them.

Exercises

1. Open the code in Listings 10.1, 10.2, and 10.3 and add a link with a `target` and `href` value. Then modify the JavaScript to also display those values in the info portion of the web page so that you can hover over the link and see them.

2. Open the code in Listings 10.7, 10.8, and 10.9 and add four new buttons. Add two buttons to move the image up and down. Then add two buttons that change the opacity of the image up .1 or down .1. You will need to make certain that the opacity stops at 0 and 1.

Lesson 11. Working with Window, Browser, and Other Non-Web Page Elements

What You'll Learn in This Lesson:

- Adding timers to web pages
- Getting and setting cookies
- Creating pop-ups

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

The first part of this lesson describes the `screen`, `window`, `location`, and `history` objects that provide JavaScript with an interface to access information beyond the web page. The second part covers utilizing those objects to implement cookies, pop-up windows, and timers.

Understanding the Screen Object

You have already seen the screen object in use in previous lessons. You have used the screen object to get the color depth as well as the height and width of the screen. Getting the screen dimensions has become more important with the shift toward mobile devices and tablets.

A wide range of screen sizes is available; therefore, you must be able to design your web pages and dynamic interactions to take the screen size into account.

Table 11.1 describes the full set of properties available on the screen object.

Property	Description
`availHeight`	Height of the physical screen minus the Windows taskbar
`availWidth`	Width of the physical screen minus the Windows taskbar
`colorDepth`	Number of bits in the color palette available to display images
`height`	Full height of the physical screen
`pixelDepth`	Color resolution of the screen in bits per pixel
`width`	Full width of the physical screen

TABLE 11.1 Screen Object Properties

Using the Window Object

The window object is by far the most robust of the external object set. The window object provides access to the browser window, allowing you to get information such as the dimensions and position of the browser window.

Using the window object, you can also create and control new browser windows when you want to display additional web content but do not want to navigate away from the current page.

The following sections describe some of the methods and properties attached to the window object.

Accessing the Window Object Properties

The window object provides you with important information about browser windows. You can access the size and position of the current window or even its parent window. For example, the following code gets the pixels down and left from the top left of the screen to the top left of the browser:

Click here to view code image

```
var fromTop = window.self.screenY;
var fromLeft = window.self.screenX;
```

Table 11.2 shows a list of some of the more important window object properties and what they are used for.

Property	Description
`closed`	True if window has been closed; false if it is still open.
`innerHeight`	Settable. Inner height of a window's content area.
`innerWidth`	Settable. Inner width of a window's content area.
`name`	Settable. Name of a window.
`opener`	Window object for the parent window that created this one.
`outerHeight`	Settable. Outer height of a window, including additional bars.
`outerWidth`	Settable. Outer width of a window, including additional bars.
`pageXOffset`	Number of pixels the current page has been scrolled down.
`pageYOffset`	Number of pixels the current page has been scrolled up.
`parent`	Parent window object of the current window.
`screenX`	X coordinate of the window relative to the screen.
`screenY`	Y coordinate of the window relative to the screen.
`self`	Current window object.
`top`	Topmost browser window object.

TABLE 11.2 Window Object Properties

Using the Window Object Methods

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

For example, the following code opens a new browser window and loads the URL specified:

Click here to view code image

```
var tempWindow = window.open("http://jquery.com");
```

Later, from the JavaScript code in your original web page, you can close the new window using the following statement:

```
tempWindow.close();
```

Table 11.3 shows a list of some of the more important window object properties and what they are used for.

Method	Description
`alert()`	Launches a dialog box with a message and an OK button
`blur()`	Removes focus from the current window
`clearInterval()`	Clears a timer set with `setInterval()`
`clearTimeout()`	Clears a timer set with `setTimeout()`
`close()`	Closes the current window
`confirm()`	Launches a dialog box with a message and an OK and Cancel button
`createPopup()`	Launches a dialog box
`focus()`	Sets focus to the current window
`moveBy()`	Moves a window relative to its current position
`moveTo()`	Moves a window to the specified position
`open()`	Opens a new browser window
`print()`	Prints the content of the current window
`prompt()`	Displays a dialog box that prompts the visitor for input
`resizeBy()`	Resizes the window by the specified pixels
`resizeTo()`	Resizes the window to the specified width and height
`scrollBy()`	Scrolls the content by the specified number of pixels
`scrollTo()`	Scrolls the content to the specified coordinates
`setInterval()`	Calls a function or evaluates expression at *n* ms intervals
`setTimeout()`	Calls a function or evaluates expression after *n* ms

TABLE 11.3 Window Object Methods

Using the Browser Location Object

The browser location object gives you access to the current location in the browser. This allows you to access all the URL information as well as reload the current page or load a new one in the current window.

For example, the following statement gets the following URL from the current page and then loads a new page at a different URL:

Click here to view code image

```
var oldURL = location.href;
location.assign("http://jquery.com");
```

Also, if the URL was linked to a specific anchor on the web page, you can get that portion of the URL using `location.hash`. You can use the anchor points that have existed in static web pages as a way to provide backward compatibility with other web pages that link to specific locations. You read the anchor hash and then adjust the dynamic content to match what was located at that portion of the original web page:

```
var anchor = location.hash;
```

Table 11.4 shows a list of the location object properties and methods.

Property	Description
`hash`	Anchor portion of a URL if specified
`host`	Hostname and port of a URL
`hostname`	Hostname of a URL
`href`	Entire URL
`pathname`	Path portion of a URL
`port`	Port number of the server
`protocol`	Protocol of a URL (http/https/and so on)
`search`	Query portion of a URL (?k=v&k2=v2...)
`assign(url)`	Loads a new document
`reload()`	Reloads the current document
`replace(url)`	Replaces the current document with a new one

TABLE 11.4 Location Object Properties and Methods

Using the Browser History Object

The history object provides access to the browser navigation history, allowing you to move forward and backward dynamically without the user needing to click the browser Forward and Back buttons.

Navigating Forward in the Browser History

To move forward, you can use `history.forward()` to move to the next URL in the history, or you can use `history.go(n)`, where `n` is a positive number that represents the number of steps to move forward. For example, the following statement moves three URLs forward:

```
history.go(3);
```

Navigating Backward in the Browser History

To move backward, you can use `history.back()` to move to the previous URL in the history, or you can use `history.go(n)`, where `n` is a negative number that represents the number of steps to move backward. For example, the following statement moves two URLs back:

```
history.go(-2);
```

Controlling External Links

An important part of dynamic web programming also involves controlling the linkage outside of the web page. The following sections describe some of the ways that you can control the behavior of external links by preventing them from happening or forcing them to open new browser windows.

Stopping External Links on a Web Page

A useful task that you can perform with a simple jQuery script is stopping external links from happening. This allows you to lock linking away from the web page using one of the `<a>` elements within it.

To lock down external links from a web page, you need to first add a click event handler to the `<a>` tags that link externally and then call `preventDefault()` on the click event object. For example, the following code uses the jQuery `^=` selector syntax to find `<a>` tags where the `href` begins with `http://` and then adds a click handler function that prevents the default browser action:

Click here to view code image

```
$('a[href^="http://"').click(function (e){
    any of your own handler code . . .
    e.preventDefault();
  });
```

Forcing Links to Open in New Browser Windows

Another useful task that you can perform with a simple jQuery script is forcing external links to open in new windows. This allows the current window to remain available.

To force external links to open in a new window, set the `target` attribute to "_blank" for `<a>` tags that link externally. For example, the following code finds `<a>` tags where the `href` begins with `http://` and then sets the `target` attribute to "_blank", forcing the links to open in a new browser window when clicked:

Click here to view code image

```
$('a[href^="http://"').attr("target", "_blank");
```

Try it Yourself: Getting and Setting Cookies

Cookies that allow you to store bits of information statically in the client's browser are an important part of the web paradigm. Often, cookies are read by server-side scripts; however, it can also be helpful if you can get and set cookies from JavaScript without the need for additional server communication.

You can get and set cookies by reading or writing to the `document.cookie` attribute. The `document.cookie` is in a string format that includes the `name`, `value`, `expiration`, and `path` in the following format:

```
name=value;expiration;path
```

The simplest way to help you understand how to implement cookies is to show you by example. The following example takes you through the process of creating simple JavaScript to get and set cookies from a web page. The code for the example is shown in Listings 11.1, 11.2, and 11.3. Use the following steps to implement the example:

1. In Eclipse, create the lesson11/cookies.html, lesson11/js/cookies.js, and lesson11/css/cookies.css files.
2. Add the code shown in Listings 11.1, 11.2, and 11.3 to the HTML file. You should recognize the HTML components. The HTML code contains several `<span>` elements that are styled as buttons, text inputs to input cookie names and values, and then a list of cookies.
3. Add the CSS from Listing 11.3 to the CSS file. This code styles the buttons.
4. Open the cookies.js file and add the following `ready()` function that adds click handlers for the get, set, and delete cookie buttons. Notice that the handlers get the cookie names from the name `<input>` field:

Click here to view code image

```
$(document).ready(function(){
  $("#set").click(function(e){setCookie($("#cookieName").val(),
                                          $("#cookieValue").val(),
1);});
  $("#get").click(function(e){getCookie($("#cookieName").val());});
  $("#delete").click(function(e){setCookie($("#cookieName").val(),
"", -1);});
  displayCookies();
});
```

5. Add the following `setCookie()` hander function that gets the date and uses it to create an expires time string. The code then sets `document.cookie` using the `name`, `value`, `expires`, and a root path as the string value:

Click here to view code image

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

6. The following `getCookie()` handler gets the `document.cookie` string, splits it by `;`, and then iterates through the cookie array until it finds a cookie where the name matches the one passed in. The function then sets the value `<input>` field to the value of the cookie:

Click here to view code image

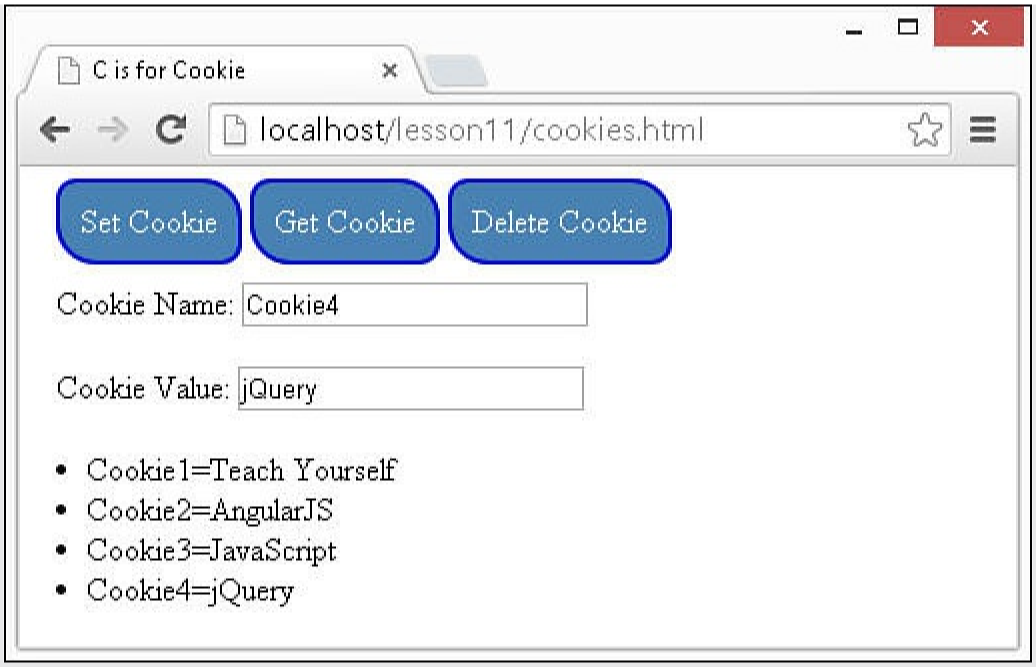
```
function getCookie(name) {
  var cookieStr = $("#cookieName").val() + "=";
  var cArr = document.cookie.split(';');
  for(var i=0;i < cArr.length;i++) {
    var cookie = cArr[i];
    while (cookie.charAt(0)==' '){
      cookie = cookie.substring(1, cookie.length);
    }
    if (cookie.indexOf(cookieStr) == 0){
      $("#cookieValue").val(cookie.substring(cookieStr.length,
cookie.length));
      break;
    }
  }
}
```

7. Add the following `displayCookies()` function that renders the list of currently set cookies:

Click here to view code image

```
function displayCookies(){
  $("#cookieList").html("");
  var cArr = document.cookie.split(';');
  for(var i=0;i < cArr.length;i++) {
    var cookie = cArr[i];
    $("#cookieList").append($("<li></li>").html(cookie));
  }
}
```

8. Save all three files and then open the HTML document in a web browser, as shown in Figure 11.1. Play around with setting, getting, and deleting cookies. Also navigate away from the page and then back, and the cookies should still be set.



Cookie Name: Cookie4

Cookie Value: jQuery

- Cookie1=Teach Yourself
- Cookie2=AngularJS
- Cookie3=JavaScript
- Cookie4=jQuery

FIGURE 11.1 Getting, setting, and deleting cookies using JavaScript.

LISTING 11.1 cookies.html HTML File Basic Web Page Used in the Example That Defines Several `<span>` Elements Used for Buttons and `<input>` Elements to Input Cookie Names and Values

[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>C is for Cookie</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/cookies.js"></script>
    <link rel="stylesheet" type="text/css"
href="css/cookies.css">  </head>
  <body>
    <div>
      <span id="set">Set Cookie</span>
      <span id="get">Get Cookie</span>
      <span id="delete">Delete Cookie</span>
    </div>
    <div>
      <label>Cookie Name: </label><input id="cookieName" type="text" />
    </div>
    <div>
      <label>Cookie Value: </label><input id="cookieValue" type="text"
/>
    </div>
    <div id="cookieList"></div>
  </body>
</html>
```

LISTING 11.2 cookies.js jQuery and JavaScript Code That Gets, Sets, and Deletes Cookies

Click here to view code image

```
$(document).ready(function(){
  $("#set").click(function(e){setCookie($("#cookieName").val(),
                                        $("#cookieValue").val(), 1);});
  $("#get").click(function(e){getCookie($("#cookieName").val());});
  $("#delete").click(function(e){setCookie($("#cookieName").val(), "", -1);});
  displayCookies();
});
function setCookie(name, value, days) {
  var date = new Date();
  date.setTime(date.getTime()+(days*24*60*60*1000));
  var expires = "; expires="+date.toGMTString();
  document.cookie = name + "=" + value + expires + "; path=/";
  displayCookies();
}
function getCookie(name) {
  var cookieStr = $("#cookieName").val() + "=";
  var cArr = document.cookie.split(';');
  for(var i=0;i < cArr.length;i++) {
    var cookie = cArr[i];
    while (cookie.charAt(0)==' '){
      cookie = cookie.substring(1, cookie.length);
    }
    if (cookie.indexOf(cookieStr) == 0){
      $("#cookieValue").val(cookie.substring(cookieStr.length, cookie.length));
      break;
    }
  }
}
function displayCookies(){
  $("#cookieList").html("");
  var cArr = document.cookie.split(';');
  for(var i=0;i < cArr.length;i++) {
    var cookie = cArr[i];
    $("#cookieList").append($("<li></li>").html(cookie));
  }
}
```

LISTING 11.3 cookies.css CSS Code That Styles the Buttons and Images

Click here to view code image

```
span{
  padding:10px;
  background-color:steelblue; color:white;
  border-radius:10px 20px; border:2px ridge blue;
  cursor:pointer;
}
div{ padding:10px; }
```

Adding Pop-up Boxes

The window provides several methods that allow you to launch pop-up windows that you can interact with for alerts, prompts, and notifications. The pop-up windows are displayed, and the user needs to interact with the pop-up before continuing to access the web page.

Note

It is often much better to create a fixed position `<div>` element with an overlay rather than using these pop-up boxes because you have much more control over them. You learn how to do that a little later in the book.

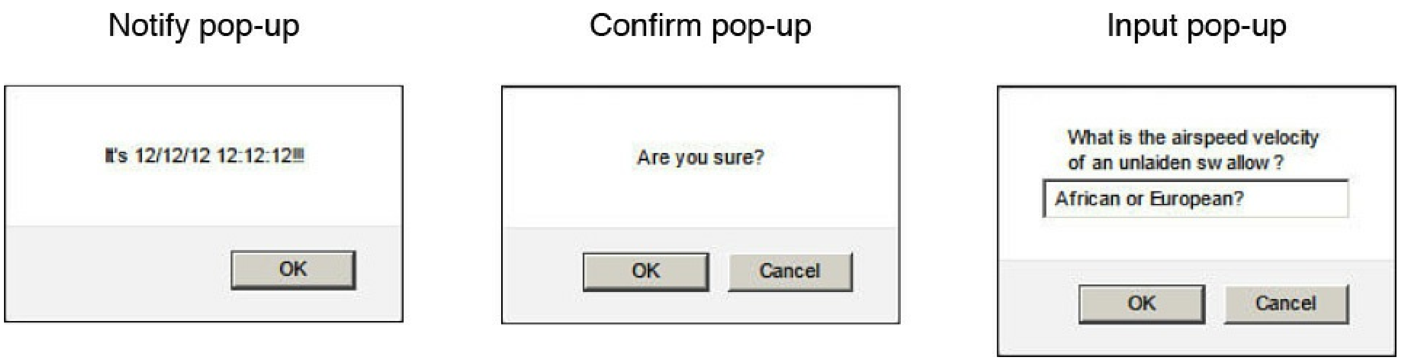
Notifying the User

The most common type of pop-up is an alert pop-up designed to notify the user that something has happened. The user will see the message; however, the only option given is to close the pop-up message.

To create a simple alert message, use the `window.alert()` method as shown next and displayed in Figure 11.2:

Click here to view code image

```
window.alert("It's 12/12/12 12:12:12!!!");
```



It's 12/12/12 12:12:12!!!

Are you sure?

What is the airspeed velocity of an unlaiden sw allow ?

African or European?

FIGURE 11.2 JavaScript pop-up boxes.

Asking the User to Confirm

The next most common type of pop-up is a confirmation pop-up designed to notify the user that something is about to happen. The user will see the message and then be given the option to click OK to allow the action to occur or click Cancel to reject the action.

To create a confirmation dialog box that allows the user to respond with yes or no, use the `window.confirm()` method, as shown next and displayed in Figure 11.2:

Click here to view code image

```
var response = window.confirm("Are you sure?");
if (response == true) { do something; }
else { don't do something; }
```

Prompting the User for Input

Another type of pop-up is the prompt. The prompt displays a text box that allows the user to type a text string into the pop-up box. That string is returned to the JavaScript code and can be used in various ways.

To create a prompt dialog that allows the user to input a single text string as input, use the `window.prompt()` method, as shown next and displayed in Figure 11.2:

Click here to view code image

```
var response = window.prompt("What is the airspeed velocity of an unlaiden
swallow?");
if (response == "African or European?"){ pass }
else { no pass }
```

Setting Timers

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

Using timers allows you to delay the execution of code so that it does not need to happen at the exact moment an event is triggered or the page is loaded.

Adding a Delay Timer

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

Click here to view code image

```
var timerId = setTimeout(myTimer, 10000);
```

At any point before the time runs out and the code is executed, you can clear the timer by calling `clearTimeout(id)` method using the id returned from `setTimeout()`.

For example:

```
clearTimeout(timerId);
```

Adding a Reoccurring Timer

You can also start a timer that will trigger on a regular interval using the `setInterval(code, ms)` method. This method also accepts a `code` statement or a function name and milliseconds as arguments. For example, the following code creates a timer that triggers every minute and calls a function `checkStatus()`:

Click here to view code image

```
var timerId = setInterval(checkStatus, 60000);
```

You can also turn off an interval timer using the `clearInterval()` method, as shown next:

```
clearInterval(timerId);
```

Try it Yourself: Creating Simple Timers and Dialogs

In this example, you create a simple web page that displays a clock and updates it on a per-second basis. The web page also pops up a notification message every few seconds until you tell it to stop. This exercise should solidify timers and alerts in your mind. The code for the example is shown in Listings 11.4, 11.5, and 11.6. Use the following steps to implement the example:

1. In Eclipse, create the lesson11/clock.html, lesson11/js/clock.js, and lesson11/css/clock.css files.
2. Add the code shown in Listing 11.4 and Listing 11.6 to the HTML and CSS files. Just basic stuff.
3. Open the clock.js file and add the following `ready()` function that adds a timeout and interval timer to the web page:

Click here to view code image

```
$(document).ready(function(){
  setTimeout(continueNotify, 3000);
  setInterval(displayTime, 1000);
});
```

4. Add the following `continueNotify()` function that will execute after 3 seconds. The function prompts users to answer whether they want to continue to receive notifications; if they do, the timeout is reset so the function will run again in another 3 seconds:

Click here to view code image

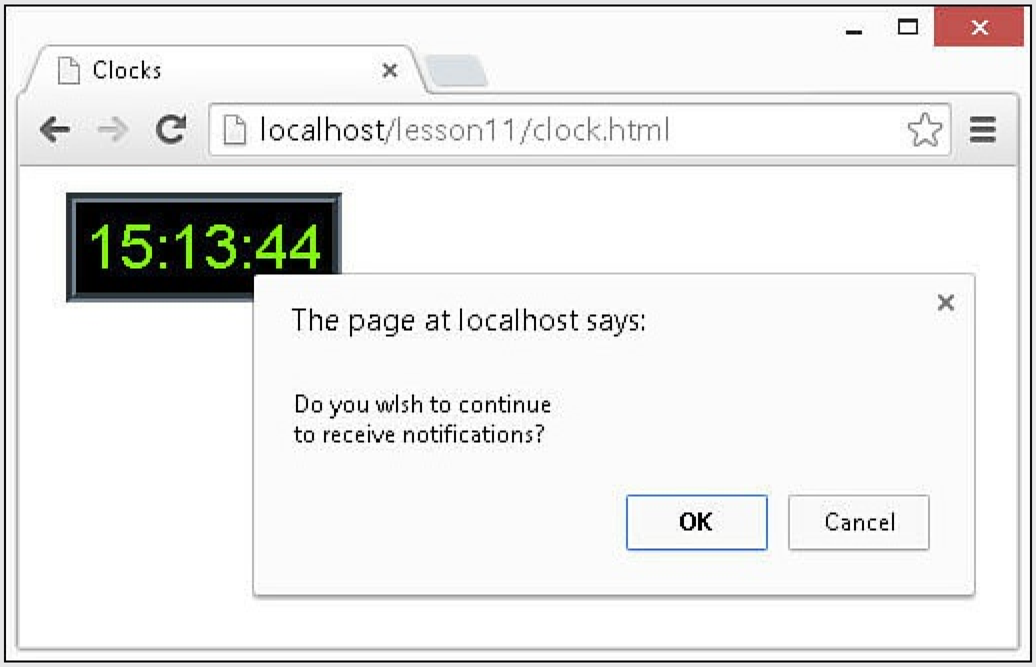
```
function continueNotify(){
  var result = confirm("Do you wIsh to continue\nto receive notifications?");
  if (result==true) { setTimeout(continueNotify, 3000); }
}
```

5. Add the following `displayTime()` function that will run every second when the interval timer is triggered. This function updates the `#clock` element with the current time string. The `padNumber()` function is used to add a leading zero for numbers less than 10:

Click here to view code image

```
function displayTime(){
  var date = new Date();
  $("#clock").html(padNumber(date.getHours()) +":"+
                   padNumber(date.getMinutes()) +":"+
                   padNumber(date.getSeconds()));
}
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 11.3. The time should automatically begin to update. When the prompt appears, try canceling it first and then accepting it the second time. You shouldn't see it again.



The page at localhost says:

Do you wIsh to continue
to receive notifications?

FIGURE 11.3 Simple timer and dialog app.

LISTING 11.4 clock.html HTML File Basic Web Used to Display a Time Element

Click here to view code image

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title>Clocks</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/clock.js"></script>
    <link rel="stylesheet" type="text/css" href="css/clock.css">
  </head>
  <body>
    <div><span id="clock"></span></div>
  </body>
</html>
```

LISTING 11.5 clock.js jQuery and JavaScript Code That Implements a Timeout and Interval Timer

Click here to view code image

```
$(document).ready(function(){
  setTimeout(continueNotify, 3000);
  setInterval(displayTime, 1000);
});
function continueNotify(){
  var result = confirm("Do you wish to continue\nto receive notifications?");
  if (result==true) { setTimeout(continueNotify, 3000); }
}
function padNumber(num){
  if (num<10){ return "0"+num; }
  return num;
}
function displayTime(){
  var date = new Date();
  $("#clock").html(padNumber(date.getHours()) +":"+
                   padNumber(date.getMinutes()) +":"+
                   padNumber(date.getSeconds()));
}
```

LISTING 11.6 clock.cs CSS Code That Styles the Clock

```
div {padding:15px;}
span{
  background-color:black;
  color:#00FF00;
  font:30px arial;
  padding:5px;
  border:5px groove;
}
```

Summary

This lesson focused on using JavaScript objects to access data outside the web page. You learned that there are screen, window, browser, location, and history objects that provide a myriad of details about the physical screen, browser, and client history, as well as access to cookies.

You saw how to open and close browser windows. Using JavaScript, you also learned how to create basic pop-ups that allow you to interact with the user.

Q&A

Q. Is there a way to find out what operating system and browser is being used?

A. Yes. The navigator object will show you the browser in the `window.navigator.appCodeName` attribute. You can also get the operating system using `window.navigator.platform`.

Workshop

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

Quiz

1. What are the three types of pop-up boxes supported by JavaScript?
2. How do you use JavaScript to find the full URL that was used to load the web page?
3. True or false: You can navigate backward through the browser history but not forward.

Quiz Answers

1. Alert, confirmation, and prompt.
2. Access the `location.href` attribute.
3. False. You can navigate forward using `history.forward()` or `history.go(n)`.

Exercises

1. Open the code in Listings 11.1, 11.2, and 11.3 and modify the `ready()` function to get a cookie named `buttonColor`. If the `buttonColor` cookie is set,

change the color of the buttons using the following jQuery line:

Click here to view code image

```
$("span").css("color", getCookie("buttonColor"));
```

2. Open the code in Listings 11.4, 11.5, and 11.6 and modify the prompt to ask the user for a number of seconds before the next notification. If the user enters 0, terminate the notification. Otherwise, use the value in the `setTimeout()` call. Remember that you need to multiply the number of seconds by 1000 to get milliseconds.

Part III: Building Richly Interactive Web Pages with jQuery

Lesson 12. Enhancing User Interaction Through jQuery Animation and Other Special Effects

What You’ll Learn in This Lesson:

- Understanding animation fundamentals
- Creating sliding elements
- Creating image galleries using simple resize and transparency animations
- Implementing expandable and collapsible elements
- How to delay animations for better effects
- How to animate element movement for dynamic web apps

One of the most important features of jQuery is the capability to add animations to changes you are making to elements. This provides the user with the feel of a slick, well-developed application rather than a clunky, traditional web page.

This is especially true if you are moving, resizing, or dynamically adding elements to the web page. It is very frustrating as a user to all of a sudden have a bunch of new things appear or disappear. Using animations for transitions gives users a chance to see where things are leaving or coming from and adjust their mindset to accept the changes.

This lesson focuses on helping you understand the fundamentals of animation. Then you will get a chance to apply those new skills in a series of practical examples.

Understanding jQuery Animation

jQuery animation is the process of modifying a property of an HTML element from one value to another in an incremental fashion visible to the user. This section describes that process and how to implement animations on CSS attributes.

Animating CSS Settings

Most animation in jQuery is done via the `.animate()` method. The `.animate()` jQuery method allows you to implement animations on one or more CSS properties. Keep in mind that the `.animate()` method acts on all elements in the jQuery object set at the same time. Often you will want to act on only a single element, so you will need to filter the set down to one.

The .animate() method accepts a CSS property object mapping as the first argument. You can specify more than one property in the object to animate multiple properties at the same time. For example, the following code animates the `height` and `width` properties for `<img>` elements:

Click here to view code image

```
$("img").animate({height:100, width:100});
```

Note

The `.animate()` method can animate only properties that have a numerical value. For example, you will not be able to animate border styles, but you can animate border size.

There are a couple of ways to call the animate method. The following shows the syntax of both:

Click here to view code image

```
.animate(properties [, duration] [, easing] [, complete])
.animate(properties, options)
```

The first method allows you to specify the `duration`, `easing`, and `complete` functions as optional arguments. The second method allows you to specify the options as a single option map object. For example, the following calls `.animate()` with a `duration` and `easing` object map:

Click here to view code image

```
$("img").animate({height:100, width:100}, {duration:1000,
easing:"linear"});
```

Tip

You cannot animate color changes using the color names; however, you can animate color changes using the hex values, such as `#ff0000`.

Table 12.1 describes the different options available for the `.animate()` method. These options are also available on some of the other animation methods that are discussed later in this lesson.

Option	Description
`complete`	Defines a function that will be called when the animation has completed.
`duration`	A string or number specifying how long the animation will run. The optional string values are `"slow"` or `"fast"`. A number specifies the number of milliseconds the animation will run. If no duration is specified, the default is 400ms.
`easing`	A string indicating which easing function to use for the transition. Currently, the values are `"swing"` (default) or `"linear"`, which provide a more constant speed to the animation. Additional easing functions are available in the jQueryUI library that is discussed later.
`queue`	Can be true, meaning the animation will be queued up behind any others for the object; false, meaning that the animation will start immediately; or a string specifying the name of a specific queue.
`step`	Specifies a function that will be executed each step in the animation until the animation completes.
`specialEasing`	You can also map the easing directly in the properties map, allowing you to do different easing for different elements. For example: `$("img").animate({height:[100, "swing"], width:[100,"linear"]}, 1000);`

TABLE 12.1 Animation Options

Understanding Animation Queues

Animations happen asynchronously with code executing, meaning that the code continues to execute while the animation is happening. What happens if you specify another animation for an object before the first one completes? The answer is that jQuery queues the animations and then executes them in order, one after another, until all are competed; that is, unless you specify `queue:false` in the animation options.

You must understand the animation queue because if you allow user interactions to queue too many animations by moving the mouse or clicking, the animations will be sluggish and behind the users' actions.

Caution

You must pay attention to where you trigger your animations from. Remember that events will bubble up. If you execute the same animation from all levels during the bubble-up phase, you could have some seriously undesired results. To prevent this, you can use the `stopPropagation()` and `stopImmediatePropagation()` methods.

Stopping Animation

jQuery enables you to stop the current animations currently executing on elements contained in the jQuery object set. The `.stop([clearQueue] [, jumpToEnd])` method allows you to stop animations in a few ways.

Calling `.stop()` with no parameters pauses the animations that are in the queue. The next animation that starts will begin executing animations in the queue again. For example, the following code pauses all animations:

```
$("*").stop();
```

Calling `.stop(true)`, with the `cleareQueue` option set to `true`, stops animations at the current point and removes all animations from the queue. For example, the following stops all animations on images and removes the animations from the queue:

```
$("img").stop(true);
```

Calling `.stop(true, true)`, with the `jumpToEnd` option set to true, causes the currently executing animation to jump to the end value immediately, clear the queue, and then stop all animations. For example, the following stops all animations on images but finishes the adjustment made by the current animation and then removes the animations from the queue:

```
$("img").stop(true, true);
```

The `.stop()` method returns the jQuery object, so you can chain additional methods onto the end. For example, the following code stops all animations on images and then starts a new one to set the `opacity` to `.5`:

Click here to view code image

```
$("img").stop(true, true).animate({opacity:.5}, 1000);
```

Delaying Animation

A great option when implementing animations is adding a delay before the animation is added to the queue. This can be used to provide animations in a more advanced way because you delay the execution of the animation queue, allowing the user a better visual experience.

The `.delay(duration, [, queueName])` method enables you to specify the delay `duration` in milliseconds, as well as an optional `queueName` that specifies what queue to apply the delay to. For example, the following code adds a size animation to images; then after the size is complete, there will be a delay of 2 seconds and the `opacity` will animate up to `1`:

Click here to view code image

```
$("img").animate({width:500}, 1000).delay(2000).animate({opacity:1} 1000);
```

Note

The `.delay()` method is great for delaying between queued jQuery effects; however, it is not a replacement for the JavaScript `setTimeout()` function, which may be more appropriate for certain use cases—especially those cases that you require to have the capability to cancel the delay.

Applying .promise() to Animations

The `.promise([type], [, target])` method allows you to apply functionality after all actions bound to the jQuery object's set are completed. It does this by returning a new object that will observe the actions and not execute any attached methods until the actions have completed.

The `.promise()` returns an object similar to a deferred object. It has a `.done()` method that enables you to pass in a function that will be run after the `.promise()` functionality has been executed.

For example, the following code waits for animations to complete, then changes the text to "complete":

Click here to view code image

```
$("span").animate({opacity:0}, 30000).promise().done(function(){
    $("p").html("complete");
  });
```

The `.promise()` method accepts two optional arguments. The first is `type`, which specifies the type of action. The default type is `fx`, which applies to animations; so for animations, you do not need to specify the type. The second parameter is `target`, which specifies an optional target jQuery object to return rather than a newly created one.

Animating Show and Hide

You have already seen the `.show()` and `.hide()` methods in action in Lesson 9. It is common practice to animate this functionality, so jQuery has nicely provided animation options for these methods to make your life easier.

Animating hide()

The `.hide( [duration] [, easing] [, callback])` method provides the optional `duration`, `easing`, and `callback` options allowing you to animate the hide effect, making less of a jump when the element disappears.

For example, the following code applies an animation of 1 second with `linear` easing and executes a simple callback function when hiding an element:

Click here to view code image

```
$("#box").hide(1000, "linear", function() { $("#label").html("Hidden!")
});
```

Animating **show()**

The `.show( [duration] [, easing] [, callback])` method provides the optional `duration`, `easing`, and `callback` options allowing you to animate the show effect, making a more easy transition as an element appears.

For example, the following code applies an animation of 1 second with `linear` easing and executes a simple callback function when showing an element:

Click here to view code image

```
$("#box").show(1000, "linear", function() { $("#label").html("Shown!") });
```

Animating **toggle()**

The `.toggle( [duration] [, easing] [, callback])` method provides the optional duration, easing, and callback options allowing you to animate the toggle between the show or hide effect when toggling between them.

For example, the following code applies an animation of 1 second with `linear` easing and executes a simple callback function when toggling an element between hidden or shown:

Click here to view code image

```
$("#switch").toggle(1000, "linear", function() { $("#label").html("Switch
Toggled!") });
```

Try it Yourself: Using Show and Hide Animations to Create an Expand/Collapse Element

In this example, you create a simple web element that provides a title bar with a collapse and expand button on the left, allowing you to expand and collapse an image. The purpose of the exercise is to provide you with a chance to use the show and hide animations. The code for the example is in Listings 12.1, 12.2, and 12.3.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson12, lesson12/js, and lesson12/css folders, and

then add the lesson12/image_hide.html, lesson12/js/image_hide.js, and hou12/css/image_hide.css files.

Note

For the exercises in this lesson, you will need to use your own images or get the images from the book's website.

---

2. Add the code shown in Listing 12.1 and Listing 12.3 to the HTML and CSS files. It's just basic stuff—a `<div>` with `<span>` used for a handle, an `<img>`, and a footer `<div>`.
3. Open the image_hide.js file and add the following `load()` function that adds a click handler to the `#handle` element:

Click here to view code image

```
$(window).load(function(){
  $("#handle").click(toggleImage);
});
```

4. Add the following handler to toggle the visibility of the image. The `if` statement checks the text in `#handle` and then either calls `show()` to display the image or `hide()` to hide it. Notice that on `show()`, there is a complete function that displays the footer, letting the user know that the image is ready. On `hide()`, the footer is hidden:

Click here to view code image

```
function toggleImage(){
  if ($("#handle").html() == '+'){
    $("#photo").show(1000, function(){$("#footer").show();});
    $("#handle").html('-');
  } else {
    $("#footer").hide();
    $("#photo").hide(1000);
    $("#handle").html('+');
  }
}
```

5. Save all three files and then open the HTML document in a web browser, as shown in Figure 12.1. You should be able to expand the image and collapse it and see the footer displayed at the appropriate time.

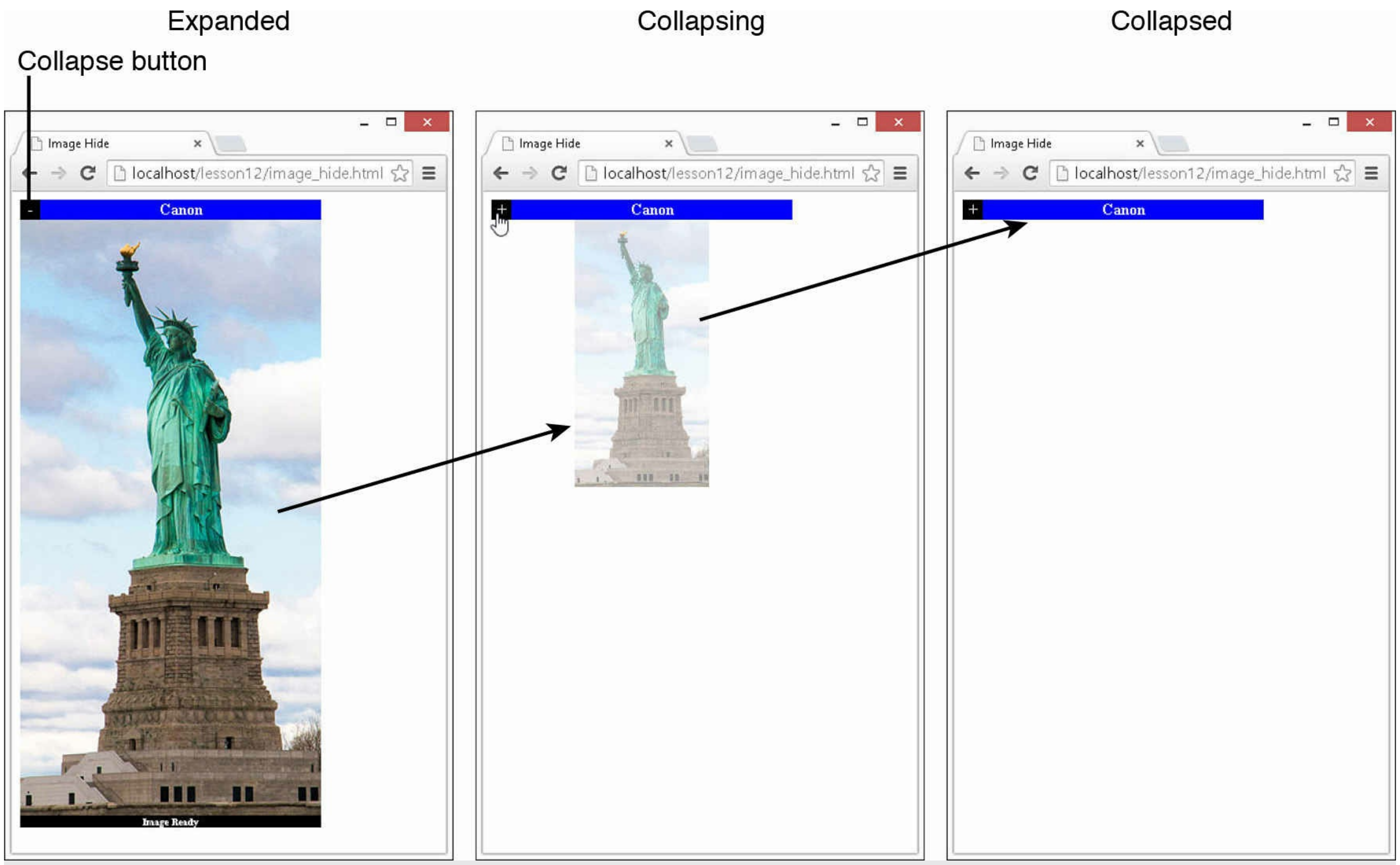


FIGURE 12.1 Simple expand/collapsible image element.

LISTING 12.1 image_hide.html HTML File Basic Web Used to Display the Collapsible Image Element

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Image Hide</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/image_hide.js"></script>
    <link rel="stylesheet" type="text/css" href="css/image_hide.css">
  </head>
  <body>
    <div>
      <div id="title"><span id="handle">-</span>Canon</div>
      <img id="photo" src="/images/liberty.jpg" width="300px"/>
      <div id="footer">Image Ready</div>
    </div>
  </body>
</html>
```

LISTING 12.2 image_hide.js jQuery and JavaScript Code That Implements the Collapsible Image

Click here to view code image

```
$(window).load(function(){
  $("#handle").click(toggleImage);
});
function toggleImage(){
  if ($("#handle").html() == '+'){
    $("#photo").show(1000, function(){$("#footer").show();});
    $("#handle").html('-');
  } else {
    $("#footer").hide();
    $("#photo").hide(1000);
    $("#handle").html('+');
  }
}
```

LISTING 12.3 image_hide.css CSS Code That Styles the Collapsible Image Element

Click here to view code image

```
div{ width:300px; text-align:center; }
#title, #handle, #footer{
  background-color:blue; color:white;
  font-weight:bold;
}
#handle{
  display:inline-block; width:20px; float:left;
  background-color:black; cursor:pointer;
}
#footer{
  font-size:10px; background-color:black;
  margin-top:-5px
}
```

Animating Visibility

jQuery also provides animation capability in fade methods attached to the jQuery objects. In the end, the fade methods are equivalent to using `.animate()` on the opacity property.

The following sections describe applying animation to the various fading methods.

fadeIn()

The `.fadeIn( [duration] [, easing] [, callback])` method provides the optional `duration`, `easing`, and `callback` options allowing you to animate fading the `opacity` of an element from its current value to 1.

For example, the following code applies an animation of 1 second with `swing` easing to all image elements:

Click here to view code image

```
$("img").fadeIn(1000, "swing");
```

fadeOut()

The `.fadeIn( [duration] [, easing] [, callback])` method provides the optional `duration`, `easing`, and `callback` options allowing you to animate fading the `opacity` of an element from its current value to 0.

For example, the following code applies an animation of 1 second with `swing` easing to all image elements and then, when completed, fades them back in again:

Click here to view code image

```
$("img").fadeOut(1000, "swing", function() { $(this).fadeIn(1000);});
```

fadeToggle()

The `.fadeToggle( [duration] [, easing] [, callback])` method provides the optional `duration`, `easing`, and callback options allowing you to animate fading the `opacity` of an element from its current value to 0 or 1, depending on its current value.

For example, the following code applies an animation of 3 seconds with `swing` easing to all image elements. Images that are currently visible are faded out, and images that are currently transparent are faded in:

Click here to view code image

```
$("img").fadeToggle(3000, "swing");
```

fadeTo()

The `.fadeTo( duration, opacity [, easing] [, callback])` method provides the `duration` and `opacity` options that specify a specific opacity to end at and how long to animate the transition. It also provides optional `easing` and `callback` arguments.

For example, the following code applies an animation of 2 seconds for all images to transition from the current opacity to .5:

```
$("img").fadeTo(2000, .5);
```

Try it Yourself: Using Fade Animation to Implement an Image Selection Effect

In this example, you apply a fade animation to an image set, allowing you to alter the transparency as the mouse hovers over an image. The purpose of the exercise is to provide you with a chance to use some of the fading techniques discussed in this section. The code for the example is in Listings 12.4, 12.5, and 12.6.

Use the following steps to create the image selection page:

1. In Eclipse, create the lesson12/image_fade.html, lesson12/js/image_fade.js, and lesson12/css/image_fade.css files.

2. Add the code shown in Listing 12.4 and Listing 12.6 to the HTML and CSS files—just a basic `<div>` with five `<img>` elements.

3. Open the image_fade.js file and add the following `load()` function that adds a `mouseover` and `mouseout` handler to all the `<img>` elements. The `mouseover` handler fades the image to an `opacity` of 1, which makes it fully opaque; the mouse out fades it back to .3 so that it will be transparent when the mouse is not over it:

Click here to view code image

```
$(window).load(function(){
  $("img").mouseover(function(){$(this).fadeTo(1000, 1);});
  $("img").mouseout(function(){$(this).fadeTo(1000, .3);});
});
```

4. Save all three files and then open the HTML document in a web browser, as shown in Figure 12.2. You should be able to move the mouse over the images and see the transparency change.

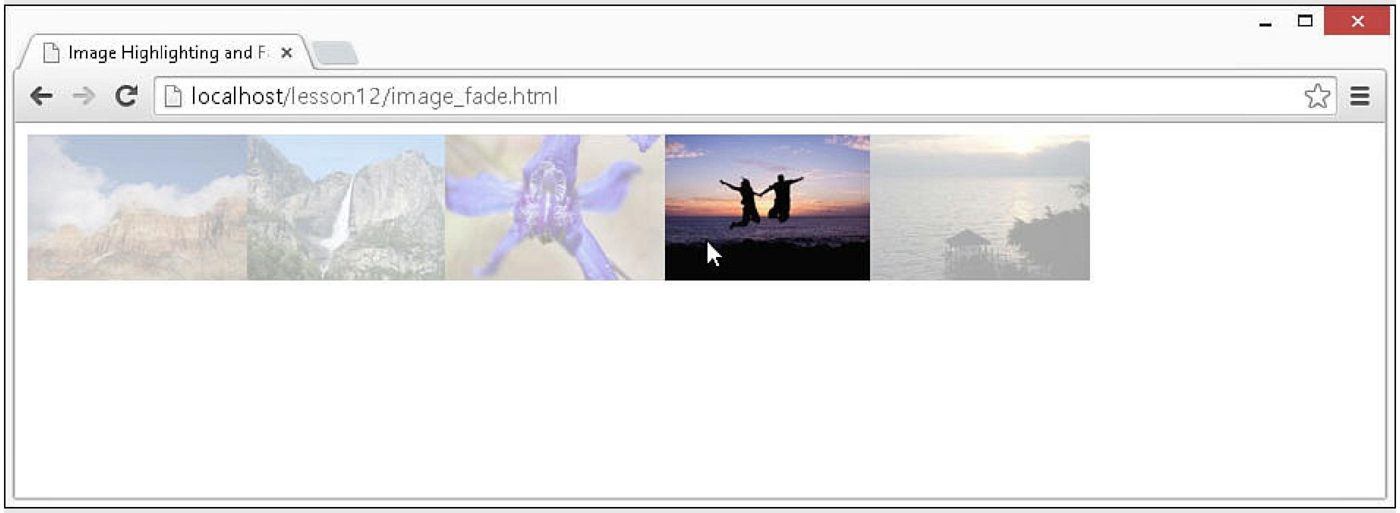


FIGURE 12.2 A simple image selection that adjusts transparency as the mouse

moves over an image.

LISTING 12.4 image_fade.html HTML File Basic Web Used to Display the Images

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Image Highlighting and Fading</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/image_fade.js"></script>
    <link rel="stylesheet" type="text/css" href="css/image_fade.css">
  </head>
  <body>
    <div>
      <div id="photos">
      <img src="/images/misty_mountains.jpg"/>
      <img src="/images/falls.jpg"/>
      <img src="/images/flower2.jpg"/>
      <img src="/images/shadow_jump.jpg"/>
      <img src="/images/beachhouse.jpg"/>
    </div>
  </body>
</html>
```

LISTING 12.5 image_fade.js jQuery and JavaScript Code That Implements Image Selection Fades

Click here to view code image

```
$(window).load(function(){
  $("img").mouseover(function(){$(this).fadeTo(1000, 1);});
  $("img").mouseout(function(){$(this).fadeTo(1000, .3);});
});
```

LISTING 12.6 image_fade.js CSS Code That Styles the Collapsible Image Element

```
div{padding:0px;}
img{
  float:left;
  opacity:.3;
  height: 100px;
}
```

Sliding Elements

A common animation is the sliding effect. A sliding effect transitions an element from taking no space to taking space from a starting edge to the finish edge. Using sliding animations gives the user a richer experience because menus, images, and other elements can be “tucked” away until the user moves the mouse over them or clicks them.

You can use a couple of ways to create a sliding element. One way is to use the built-in jQuery slide methods. The second is to animate the height and width properties. The following sections describe each of these methods.

Animating **slideUp()**, **slideDown()**, and **slideToggle()**

The `.slideUp( duration [, easing] [, callback])`, `.slideDown( duration [, easing] [, callback])`, and `.slideToggle( duration [, easing] [, callback])` methods provide the duration, easing, and callback arguments allowing you to animate sliding effects in the vertical direction.

For example, the following code applies an animation of 1 second to slide an element down, and then applies a delay of 3 seconds and slides the element back up:

Click here to view code image

```
$("#menu").slideDown(1000).delay(3000).slideUp(1000);
```

You can also animate the `.slideToggle()` method in a similar fashion. For example, the following code animates visibility of a `<div>` element using a slide animation:

```
$("div").slideToggle(1000);
```

Sliding Using Width and Height

We also like to use the width and height properties to create a sliding element. You can create a vertical slide animation by animating the height and create a horizontal slide animation by animating the width.

There are a couple of tricks. You need to provide both a width and a height value for the element if you want to have the full slide effect and not just a resize effect. Also, if you want the element to maintain space on the page, you cannot animate the value all the way down to 0. However, you can animate down to .1 and the other dimension will retain its space.

The following example shows an animation that animates sliding down by changing the `height` to 100 and then back up by changing the `height` to .1:

Click here to view code image

```
$("img").animate({height:100}, 1000);
$("img").animate({height:.1}, 1000);
```

Try it Yourself: Using Sliding Animation to Implement a Dynamic Menu

In this example, you create a simple web element that provides a title bar with a sliding effect that reveals an image menu. As you hover over each item in the menu, an image slides down and then slides back up as you leave the menu. The purpose of the exercise is to provide you with a chance to use some of the sliding techniques discussed in this section. The code for the example is in Listings 12.7, 12.8, and 12.9.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson12/sliding_images.html, lesson12/js/sliding_images.js, and lesson12/css/sliding_images.css files.
2. Add the code shown in Listing 12.7 and Listing 12.9 to the HTML and CSS files—just basic stuff: a `<div>` with a `<p>` for a title bar, and an inner `<div>` containing the `<span>` used for menus and `<img>` elements.
3. Open the sliding_images.js file and a basic `$(window).load()` function. Inside the `load()` function, add the following line that will hide the inner `<div>`:

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

4. Add the following statements to add `mouseover` and `mouseout` handlers to the `<span>` elements. On `mouseover`, you get the index of the `<span>` and then use it to find the corresponding `<img>` element and animate setting the `height` to 100, thus sliding down the image. On `mouseout`, you do the opposite, setting the `height` to 1 to slide it up:

Click here to view code image

```
$("span").mouseover(function(){
  var i = $(this).index("span");
  $("img").eq(i).animate({height:100}, 1000);
  });
$("span").mouseout(function(){
  var i = $(this).index("span");
  $("img").eq(i).animate({height:.1}, 1000);
  });
```

5. Add the following statements to add `mouseenter` and `mouseleave` handlers to the `#container` element. On `mouseenter`, you first stop propagation so that you apply the slide toggle only once during bubbling. Then you stop all animation on the `#images` element and use `slideToggle()` to animate sliding the entire menu. The same occurs on `mouseleave`. The

result is that when the mouse is over the `#container` element, it slides down and slides back up when the mouse leaves:

Click here to view code image

```
$("#container").mouseenter(function(e){
    e.stopPropagation();
    $("#images").stop(true).slideToggle(1000);
  });
$("#container").mouseleave(function(e){
    e.stopPropagation();
    $("#images").stop(true).slideToggle(1000);
    });
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 12.3. You should be able to expand the menu, and as you hover over each menu item, the image should slide down and back up when you leave.

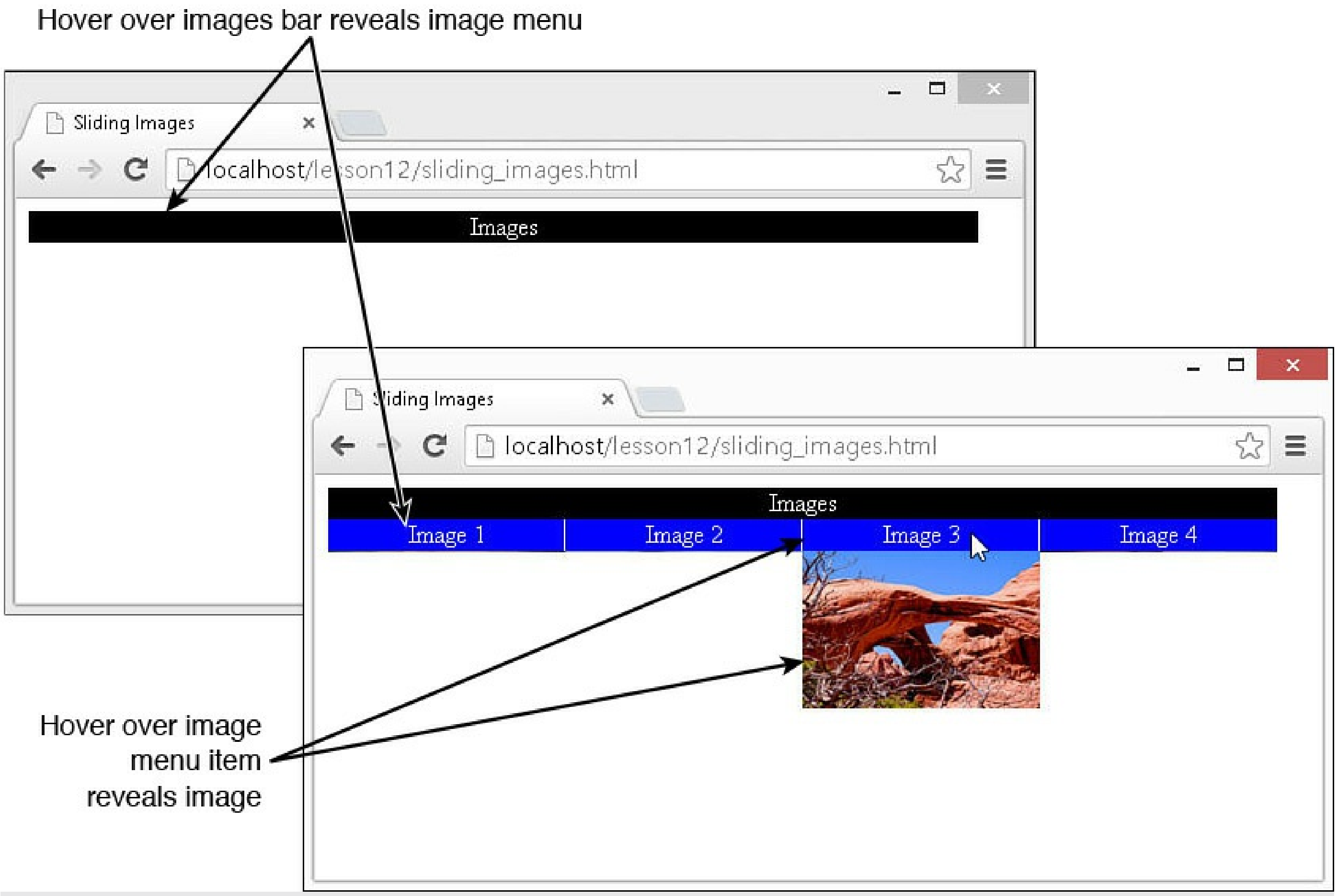

Hover over images bar reveals image menu
Sliding Images
localhost/lesson12/sliding_images.html
Images
Sliding Images
localhost/lesson12/sliding_images.html
Images
Image 1
Image 2
Image 3
Image 4
Hover over image menu item reveals image

FIGURE 12.3 Simple sliding menu element.

LISTING 12.7 sliding_images.html HTML File Basic Web Used to Display the Sliding Menu Element

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sliding Images</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/sliding_images.js"></script>
    <link rel="stylesheet" type="text/css"
href="css/sliding_images.css">
  </head>
  <body>
    <div id="container">
      <p>Images</p>
      <div id="images">
        <span>Image 1</span>
        <span>Image 2</span>
        <span>Image 3</span>
        <span>Image 4</span><br>
        <img src="/images/flower.jpg" />
        <img src="/images/img7.jpg" />
        <img src="/images/img4.jpg" />
        <img src="/images/jump.jpg" />
      </div>
    </div>
  </body>
</html>
```

LISTING 12.8 sliding_images.js jQuery and JavaScript Code That Implements the Sliding Image Menu

Click here to view code image

```
$(window).load(function(){
  $("div div").hide();
  $("span").mouseover(function(){
    var i = $(this).index("span");
    $("img").eq(i).animate({height:100}, 2000);
    });
  $("span").mouseout(function(){
    var i = $(this).index("span");
    $("img").eq(i).animate({height:1}, 2000);
    });
  $("#container").mouseenter(function(e){
      e.stopPropagation();
      $("#images").stop(true).slideToggle(1000);
    });
  $("#container").mouseleave(function(e){
      e.stopPropagation();
      $("#images").stop(true).slideToggle(1000);
      });
});
```

LISTING 12.9 sliding_images.css CSS Code That Styles the Sliding Menu

Click here to view code image

```
img{
  display:inline-block; width:150px; height:1px;
  margin:0px; padding:0px; float:left;
}
p, span {
  display:inline-block; width:600px;
  background-color:black; color:white;
  margin:0px; padding:0px; text-align:center;
}
span {
  width:150px; margin:-1px;
  border:1px solid; background-color:blue; float:left;
}
#container { width:610px; }
```

Creating Resize Animations

Similar to using the height and width to create a sliding effect, you can also use them to create a resize animation. The difference between a slide and a resize is that the aspect ratio of the image is maintained on a resize, giving the overall appearance that the element is growing or shrinking rather than being unfolded or untucked.

The trick to creating a resize animation is that you either need to specify both height and width in the `.animate()` statement, or one of them has to be auto in the CSS settings, and you animate only the one that has a value.

For example, the following code shows a resize animation of an image up to 500 pixels over 1 second, and then slowly over 5 seconds back down to 400 pixels:

Click here to view code image

```
$("img").animate({height:500, width:500}, 1000).animate({height:500,
width:500},
5000);
```

Try it Yourself: Using a Resize Animation to Create a Simple Image Gallery View

In this example, you create a simple image gallery view that resizes an image and applies opacity changes in the same animation. The purpose of the exercise is to provide you with a chance to use some of the resizing techniques discussed in this section. The code for the example is in Listings 12.10, 12.11, and 12.12.

Use the followings steps to create the dynamic web page:

1. In Eclipse, create the lesson12/animated_resize.html, lesson12/js/animated_resize.js, and lesson12/css/animated_resize.css files.

2. Add the code shown in Listing 12.10 and Listing 12.12 to the HTML and CSS files—just a basic `<div>` with `<img>` elements. In the CSS file, the `<img>` elements are styled to a width of 100px, but no height is set. That way we can animate the size without using the height property.

3. Now open the animated_resize.js file and a basic `load()` function that adds `mouseover` and `mouseout` handlers to the `<img>` elements. On `mouseover`, you get the animation by increasing the `width` and `opacity`, and do the opposite on `mouseout`:

Click here to view code image

```
$(window).load(function(){
  $("img").mouseover(function(){
      $(this).animate({width:"200px", opacity:1}, 1000);
    });
  $("img").mouseout(function(){
      $(this).animate({width:"100px", opacity:.3}, 1000);
    });
});
```

4. Save all three files and then open the HTML document in a web browser, as shown in Figure 12.4. You should be able to hover over the images and see the resize animation.

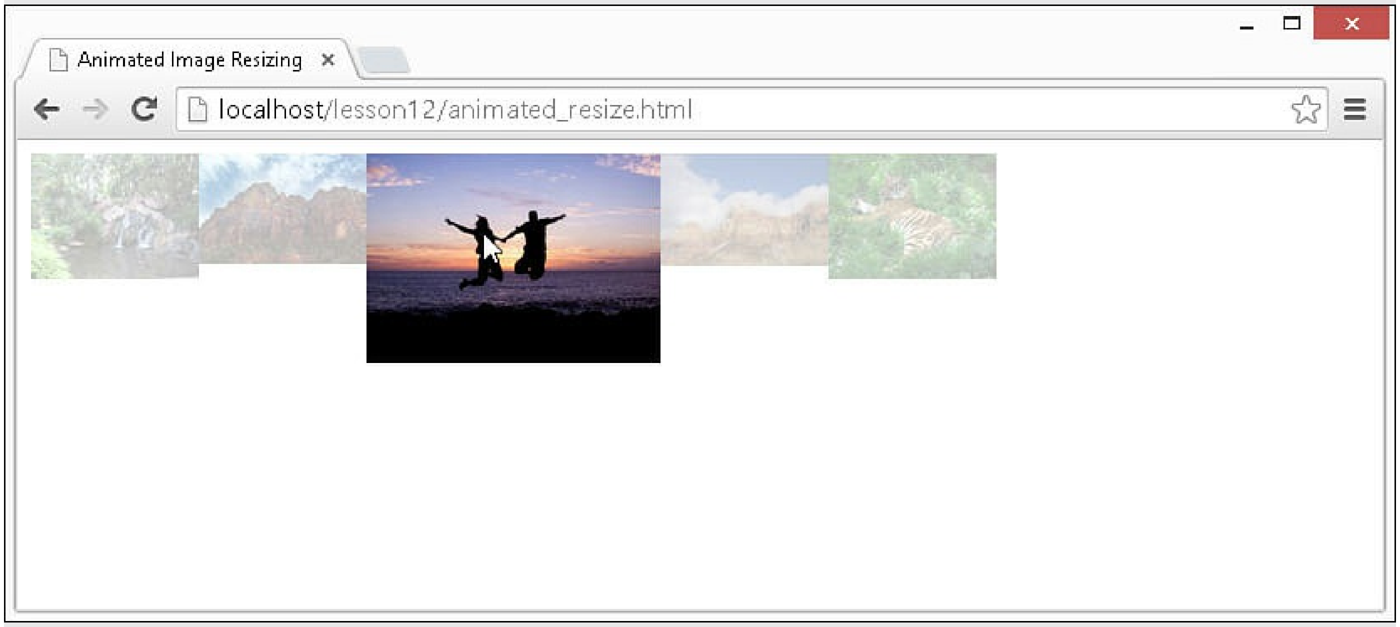


FIGURE 12.4 Simple sliding menu element.

LISTING 12.10 animated_resize.html HTML File Basic Web Used to Display the Images

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Animated Image Resizing</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/animated_resize.js"></script>
    <link rel="stylesheet" type="text/css" href="css/animated_resize.css">
  </head>
  <body>
    <div id="photos">
      <img src="/images/pool.jpg"/>
      <img src="/images/peak.jpg"/>
      <img src="/images/shadow_jump.jpg"/>
      <img src="/images/misty_mountains.jpg"/>
      <img src="/images/tiger.jpg"/>
    </div>
  </body>
</html>
```

LISTING 12.11 animated_resize.js jQuery and JavaScript Code That Implements the Resize Effect

Click here to view code image

```
$(window).load(function(){
  $("img").mouseover(function(){
      $(this).animate({width:"175px", opacity:1}, 1000);
    });
  $("img").mouseout(function(){
      $(this).animate({width:"100px", opacity:.3}, 1000);
    });
});
```

LISTING 12.12 animated_resize.css CSS Code That Styles the Images

```
div{padding:0px;}
img{
  opacity:.3;
  width:100px;
  float:left;
}
```

Implementing Moving Elements

Another dynamic that is good to animate is repositioning of elements—specifically, moving an element from one location to another. Users hate it when they do something and page elements are all of a sudden in a different location. Animating the move enables them to see where things go and make the necessary adjustments in their thinking.

The following sections describe methods of animating the repositioning of elements. You also get a chance to implement some code that should solidify the concepts for you.

Animating Element Position Changes on Static Elements

You cannot directly alter the position of static page elements, because they simply flow with the items around them. However, you can animate the margin and padding properties. For example, the following code animates a move of all `<p>` elements to the right by animating the `margin-left` property:

Click here to view code image

```
$("p").animate({"margin-left":30}, 1000);
```

Animating Element Position Changes on Nonstatic Elements

Most of the move animation you do will be on nonstatic elements, and usually it will be on fixed elements, simply because it is much easier and safer to move those without needing to worry about other element positions.

Either way, it doesn't matter if it is a fixed, absolute, or relative positioned element—you will still be animating the same two values, `top` and `left`. To animate movement vertically, you use `top`, and to animate horizontally, you use `left`. For example, the following statements animate moving an element to the right 10 pixels and then down 10 pixels:

Click here to view code image

```
var position = $("#element").offset();
$("#element").animate({top:position.top+10}, 1000);
$("#element").animate({top:position.left+10}, 1000);
```

You can also animate in a diagonal direction by animating both `top` and `left` at the same time. For example, the following code animates movement down 10 pixels and to the right 100 pixels in the same animation:

Click here to view code image

```
var position = $("#element").offset();
$("#element").animate({top:position.top+10, top:position.left+100}, 1000);
```

Try it Yourself: Creating a Simple Flying Saucer App with jQuery Animation

In this example, you create a simple web app that provides controls to fly a flying saucer around a bunch of moons. The purpose of the exercise is to provide you with a chance to use some of the move animation techniques to move the flying saucer around the page. The code for the example is in Listings 12.13, 12.14, and 12.15.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson12/interactive_animation.html, lesson12/js/interactive_animation.js, and lesson12/css/interactive_animation.css files.
2. Add the code shown in Listing 12.13 and Listing 12.15 to the HTML and CSS files. The HTML code displays a set of images for the controls. You can use a couple of `<span>` elements as spacers for the controls. The rest of the images are the flying saucer and the moons. Each moon has its own id so that it can be positioned in CSS.
3. Open the interactive_animation.js file and the following global variables and basic `load()` function. The variables store the `height` and `width` of the browser pane so that you can use that to know when to stop the animation. It is also used in line 4 to position the plane image in the middle of the web page:

Click here to view code image

```
01 var rightEdge = window.innerWidth;
02 var bottomEdge = window.innerHeight;
03 $(window).load(function(){
$("#ship").offset({top:bottomEdge/2, left:rightEdge/2});
...
22 });
```

4. Add the following handlers for the control buttons. Notice that each handler first changes the image `src` for the plane, stops the current animation, and then animates the left or top properties to move the plane. The `#stop` handler stops all animation and clears the animation queue:

Click here to view code image

```
05   $("#up").click(function(){
06       $("#ship").attr("src","/images/saucerUp.png");
07       $("#ship").stop(true).animate({top:0}, 5000);
08     });
09   $("#left").click(function(){
10       $("#ship").attr("src","/images/saucerRight.png");
```

```
    $("#ship").stop(true).animate({left:0}, 5000);
  });
$("#right").click(function(){
    $("#ship").attr("src","/images/saucerLeft.png");
    $("#ship").stop(true).animate({left:rightEdge}, 5000);
  });
$("#down").click(function(){
    $("#ship").attr("src","/images/saucerDown.png");
    $("#ship").stop(true).animate({top:bottomEdge}, 5000);
  });
$("#stop").click(function(){ $("#ship").stop(true) });
```

5. Save all three files and then open the HTML document in a web browser, as shown in Figure 12.5. You should be able to fly the flying saucer around using the control buttons. It's a very basic example, but a good way to help you understand movement animation.

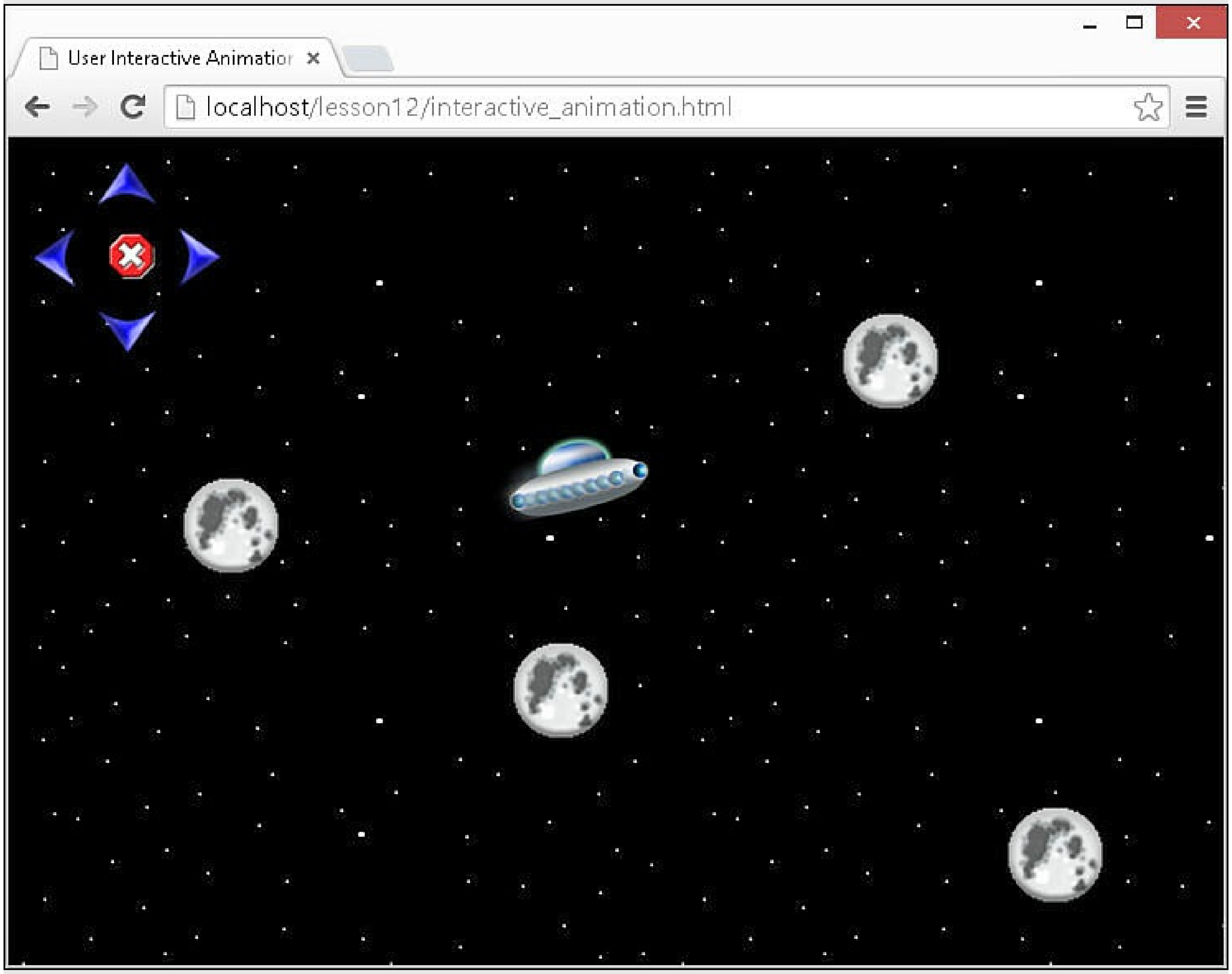

User Interactive Animatior
localhost/lesson12/interactive_animation.html

FIGURE 12.5 A simple web app that allows the user to fly a flying saucer around the screen.

LISTING 12.13 interactive_animation.html HTML File Basic Web Used to Display the Controls, Cones, and Flying Saucer

[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>User Interactive Animation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/interactive_animation.js">
</script>
    <link rel="stylesheet" type="text/css"
href="css/interactive_animation.css">
  </head>
  <body>
    <div>
      <span></span>
      <img id="up" src="/images/up2.png" /></br>
      <img id="left" src="/images/left.png" />
      <img id="stop" src="/images/stop2.png" />
      <img id="right" src="/images/right.png" /><br>
      <span></span>
      <img id="down" src="/images/down2.png" />
      <img id="ship" src="/images/saucerUp.png" />
      <img id="moon1" src="/images/night.png" />
      <img id="moon2" src="/images/night.png" />
      <img id="moon3" src="/images/night.png" />
      <img id="moon4" src="/images/night.png" />
    </div>
  </body>
</html>
```

LISTING 12.14 interactive_animation.js jQuery and JavaScript Code That Implements the Flying Saucer Movement Animation Using Click Handlers

[Click here to view code image]

```
var rightEdge = window.innerWidth;
var bottomEdge = window.innerHeight;
$(window).load(function(){
  $("#ship").offset({top:bottomEdge/2, left:rightEdge/2});
  $("#up").click(function(){
      $("#ship").attr("src","/images/saucerUp.png");
      $("#ship").stop(true).animate({top:0}, 5000);
    });
  $("#left").click(function(){
      $("#ship").attr("src","/images/saucerRight.png");
```

```
        $("#ship").stop(true).animate({left:0}, 5000);
      });
    $("#right").click(function(){
        $("#ship").attr("src","/images/saucerLeft.png");
        $("#ship").stop(true).animate({left:rightEdge}, 5000);
      });
    $("#down").click(function(){
        $("#ship").attr("src","/images/saucerDown.png");
        $("#ship").stop(true).animate({top:bottomEdge}, 5000);
      });
    $("#stop").click(function(){ $("#ship").stop(true) });
});
```

LISTING 12.15 interactive_animation.js CSS Code That Styles the Controls, Cones, and Flying Saucer

Click here to view code image

```
img{ width:40px;}
span{ width:40px; height:40px; display:inline-block;}
#ship{ position:fixed; width:100px;}
#moon1{ position:fixed; top:100px; left: 500px; width:70px;}
#moon2{ position:fixed; top:200px; left: 100px; width:70px;}
#moon3{ position:fixed; top:300px; left: 300px; width:70px;}
#moon4{ position:fixed; top:400px; left: 600px; width:70px;}
body {
  background-image:url("/images/nightSky.png");
  background-repeat:repeat;
}
```

Summary

In this lesson, you learned the basics of web animations and how to apply them to changes you make to elements. Most animations can be done using the `.animate()` method that is available on jQuery objects. You learned about the animation queue, how to stop animations and clear the queue, as well as how to delay the animations.

jQuery also provides several helper functions, such as `fadeIn()/fadeOut()` and `show()/hide()`, that simplify some of the more common animation tasks. You learned how to create some practical page elements as well as a simple app to waste time flying paper airplanes.

Q&A

Q. Is there a way to globally disable all animations?

A. Yes. You can set `jQUery.fx.off` to set all animations to the final state and disable animations. This is a useful feature if you want to disable animations for testing or if you want to allow users to disable the animations on slower devices. Setting the value to false enables animations again.

Q. Is there a way to control the number of steps required to compete the animation?

A. Sort of. You can set the `jQuery.fx.interval` value to the number of milliseconds between steps. This controls the frames per second at which the animations run. You should be careful with this setting, though; lowering the number will make the transitions smoother but will also take up more system resources.

Workshop

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

Quiz

1. Is it possible to control when an animation occurs?
2. How do you animate color changes?
3. True or false: Animating the width property only will always keep the image aspect ratio.
4. True or false. You cannot animate movement in static elements.
5. Is there a way to execute a function after all animations have completed?

Quiz Answers

1. Yes, partially. Using the `.delay()` method or `setTimeout()`, you can delay an animation for a period of time.
2. You must use the hex color values, such as `#ffffff`.
3. False. If height is auto, then that is correct; however, if height is a specific value, the aspect ratio will be ignored.
4. False. You can animate margins and padding to provide some movement animations.
5. Yes, using the `.promise()` method.

Exercises

1. Open up the code in Listing 12.1. Add a second `<span>` to the `#title`

`<div>` to place the "Image" text in. Then hide and expand that `<span>` as well as the image.

2. Open the code in Listings 12.10, 12.11, and 12.12. Modify the code by adding a click handler for the images. In the click handler, stop all animation without completing the current animation. Then turn off the mouseover and mouseout event handler using `$(this).off("mouseover");` and so on.

3. Open the code for Listings 12.13, 12.14, and 12.15. Modify the code to add four new buttons for diagonal movement. You will also need to add handlers for those buttons that apply the movement animation by setting the top and left properties in the same animation.

Lesson 13. Interacting with Web Forms in jQuery and JavaScript

What You’ll Learn in This Lesson:

- How to get and set form element values
- Dynamically building form elements
- How to use form element animations to help users navigate the form
- How to automatically focus elements for users
- Creating a basic e-commerce web form
- Simple methods of validating web forms

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

Web forms can be a positive or negative experience for users. Think about it for a second. They know they need to input data. If it seems difficult to input the data because the form is clunky or difficult to understand, they hate it and have a bad experience. If the form elegantly guides them through fast and efficiently, then they say, “Wow! That was easy.”

So how do you give the user a good experience?

1. Use dynamic programming to access the data they are entering to provide a dynamic workflow that fits what they are doing.
2. Provide visual effects, such as highlights or expanding elements, that help guide them through the page.
3. Provide inline validation to let them know as soon as possible if they begin to go astray.

The following sections go through each of the concepts. By the time you are finished, you will be able to implement some fantastic web forms.

Note

jQuery and JavaScript provide a lot of great functionality when working with web forms. The AngularJS adds to JavaScript and jQuery with the capability to bind web form elements to values in the data model for your web page. You will learn more about that in the AngularJS section of lessons.

Accessing Form Elements

The most important part of interacting with web forms is being able to access the data that they represent. Accessing the form data allows you to get and set the values, change selections, and serialize the data to be used in other ways.

There are numerous types of form elements. Unfortunately, there is not a single standard method for getting and setting the value the element represents. The following sections describe ways to access the data in each of the types of form elements.

Getting and Setting Form Element Values

Getting the value a form element represents depends on the form element type. For example, a text element represents the text inside, whereas a select element represents the value(s) of the selected element(s).

In the following sections, the form elements are broken into groups based on the methods to get the form values. For example, all the textual elements are accessed in the same way, regardless of type.

Accessing Form Element Attributes

The different elements have many of the same attributes as well as a few unique to the element type. There are several attributes you need to access when implementing dynamic code:

- **id**—Used to query for and identify the form element.
- **name**—Used in multiple fashions. For radio inputs, this attribute is used to group the elements together so that only one can be selected at a time. For serialization and submission of the form, the name attribute is used as the name given to the element's value.
- **type**—Used to identify the type of `<input>` element.
- **value**—Stores a value that is associated with the element. For text elements, the value is displayed in the text box; for buttons, it is the string in the button; for `<option>` elements, it is the value associated with the option.
- **checked**—Used to access the selection state of a radio or check box `<input>` element.

These attributes can be accessed directly in JavaScript by attribute `name`. For example, to get and set the value attribute, you could use the following:

```
domObj.value = "New Text";
var newValue = domObj.value;
```

In jQuery, there are three ways to get the properties and attributes of the form objects:

the .attr(), .prop(), and .val() methods. The .attr() method is used to access attributes of the DOM object that correspond to the HTML attributes, such as id, name, and type, whereas the .prop() method is used to access properties of the DOM object that are more JavaScript specific, such as selectedIndex of <select> elements.

Caution

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

---

jQuery provides the .val() method to access values represented by the form element. In jQuery, the value can be accessed using the .val() method of the jQuery object. For example, the following statements set the value of all <input> elements with type="text" element and then get the value of the first:

Click here to view code image

```
$("input:text").val("New Text");
var newValue = $("input:text").val();
```

Accessing Text Input Elements

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

Although all of these are a bit different in usage, they all render the same basic text box and are accessed in the same basic way. Each has a value attribute that can be set in HTML that will be displayed in the text box as the image is rendered. For example, the following code shows a basic example of rendering a text form element with an initial value, as shown in Figure 13.1:

Click here to view code image

```
<input type="text" value="Initial Text"/>
```

FIGURE 13.1 A simple text input with an initial value.

That value can be accessed directly from JavaScript by accessing the value attribute. For example, the following sets the value of a text input element and then gets the value:

Click here to view code image

```
textDomObj.value = "New Text";
var newValue = textDomObj.value;
```

In jQuery, the value can be accessed using the `.val()` method of the jQuery object. For example, the following statements set the value of all `<input>` elements with `type="text"` element and then get the value of the first:

Click here to view code image

```
$("input:text").val("New Text");
var newValue = $("input:text").val();
```

Accessing Check Box Inputs

Check box input elements have a Boolean value based on whether the element is checked. The value is accessed by getting the value of the `checked` attribute. If the element is checked, then `checked` has a value such as `true` or `"checked"`; otherwise, the value is `undefined` or `false`.

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

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

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

```
jObj.is(":checked");
```

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

```
jObj.prop("checked", true);
```

To set the state of a jQuery object representing check boxes to unchecked is similar. You need to remove the `checked` attribute using `prop()` and setting the value to `false`. For example:

Click here to view code image

```
jObj.prop("checked", false);
```

Accessing Radio Inputs

Individual radio inputs work the same way as check boxes. You can access the checked state the same way.

However, radio inputs are almost always used in groups. The value of a radio input group represents is not Boolean. Instead, it is the `value` attribute of the currently selected element.

When submitting the form or serializing the form data, the value of the radio input group is automatically populated. To get the value of a radio input group in code, you need to first access all the elements in the group, find out which one is selected, and then get the `value` attribute from that object. The following code will get the value of a radio input group that is grouped by `name="myGroup"` in jQuery:

Click here to view code image

```
var groupValue = $("input[name=myGroup]").filter(":checked").val();
```

The code first finds the `<input>` elements with `name="myGroup"`, then filters them down to the ones that are checked, and finally returns the value.

Accessing Select Inputs

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

Note

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

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

HTML:

Click here to view code image

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

jQuery:

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

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

```
"one"
```

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

```
["one"]
```

On a multiple select, the value returned by the jQuery statement if the first three options are selected is the following:

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

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

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

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

Click here to view code image

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

Accessing Button Inputs

For the most part, you will not need to access button data, with the possible exception of the `value` attribute, which defines the text displayed on the button. The `adding/removing` event handlers and CSS properties are all the same as for other HTML elements.

Accessing File Inputs

The file input is an interestingly different type of form element. It provides both a button and text box. The button links into the OS's file browser and the text box displays the path to the file that needs to be uploaded to the web server.

The `value` attribute of the file input will be the name of the file, so you can access it directly from JavaScript or by using the `.val()` method in jQuery.

In Chrome and Firefox, the DOM object also provides a files attribute that is an array of file objects representing the files selected by this element. You can access files selected by the user from JavaScript using the following code:

HTML:

Click here to view code image

```
<input id=fileSelect type="file" />
```

JavaScript:

Click here to view code image

```
var fileSelector = document.getElementById("fileSelect");
var fileList = fileSelector.files;
for (var i in fileList){
  var fileObj = fileList[i];
  var fileName = fileObj.name;
  var filePath = fileObj.mozFullPath;
  var fileSize = fileObj.size;
  var fileType = fileObj.type;
}
```

Caution

You must be careful when playing around the file input because that seems to be a sore spot for malicious behavior. Internet Explorer will fail a submit event on a page if it detects that the file objects have been tampered with. Also, many browsers have security settings that prevent some to all of the file information from being available to scripts.

Each file object contains several attributes that are useful in dynamic programming. Some of the most commonly used are listed in Table 13.1.

Attribute	Description
`name`	Filename excluding the path.
`path`	This is a bit different for each browser. Mozilla provides this attribute as `mozFullPath`, `webkitRelativePath`.
`size`	Size of the file in bytes.
`type`	File type specified by the HTTP standard. For example: `image/jpeg`.

TABLE 13.1 Properties of the DOM File Object

Accessing Hidden Inputs

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

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

HTML:

Click here to view code image

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

jQuery:

Click here to view code image

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

Serializing Form Data

Many of the input elements in a form can be easily serialized into strings or arrays. Serializing the form data makes it easier to deal with when storing it, sending it to a server, or dynamically making adjustments based on a form event.

For a form to be serialized, it needs two things: a `name` attribute and a `value` attribute that can be assigned to the name. Table 13.2 describes the different value sources for types of attributes.

Input Type	**Value Source**
`textarea`, `color`, `date`, `datetime`, `datetime-local`, `email`, `month`, `number`, `password`, `range`, `search`, `tel`, `text`, `time`, `url`, `week`	Text value displayed inside the input's text box.
`checkbox`	The value is "on" if no value attribute is specified; otherwise, it is the value of the value attribute.
`radio`	The value of the selected radio input for the group.
`select`	Value of the selected option for single selects. For multiple selects, it is an array of the values of the selected options.

TABLE 13.2 Properties of the DOM File Object

Converting a Form into a Query String

One of the most common serialization tasks is converting the form data into a serialized array. jQuery makes this a snap with the `.serialize()` method. The `.serialize()` method will access the form and convert the name/value pairs into an URL-encoded string ready to be transmitted across the web.

For example, check out the following code that creates a form and then serializes it based on the values set in Figure 13.2:

HTML:

Click here to view code image

```
<form id="simpleForm">
  <input name="title" type="text" /><br>
  <select name="mySelection" multiple size=3 id="mySelect">
    <option value="one">One</option>
    <option value="two">Two</option>
    <option value="three">Three</option>
  </select><br>
  <input type="radio" name="gender" value="male">Male</input>
  <input type="radio" name="gender" value="female">Female</input><br>
</form>
```

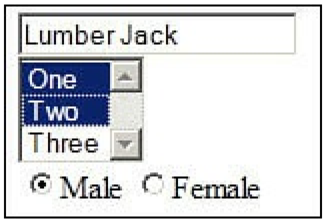


FIGURE 13.2 Simple form with `text`, `select`, and `radio` inputs.

jQuery:

Click here to view code image

```
var qString = $("#simpleForm").serialize();
```

Value of qString:

Click here to view code image

```
title=Lumber+Jack&mySelection=one&mySelection=two&gender=male
```

Converting a Form into a JavaScript Object

Another very useful serialization technique is to convert the form data into a JavaScript object that can then be accessed. The jQuery `.serializeArray()` method will do just that. All name/value pairs are converted to an array that can be accessed via your code.

For example, consider the following jQuery statement running on the same form shown in Figure 13.2:

Click here to view code image

```
var formArr = $("#simpleForm").serializeArray();
```

The resulting value of `formArr` would be the following:

Click here to view code image

```
{0: {"name":"title", "value":"Lumber Jack"},
1: {"name":"mySelection", "value":"one"},
2: {"name":"mySelection", "value":"two"},
3: {"name":"gender", "value":"male"}};
```

Try it Yourself: Accessing and Manipulating Form Element Data

Now that you have had a chance to review the methods of accessing and interacting with form data, it is time to jump in and do it yourself. In this exercise, you create a couple of forms to get data from and the other to set the data in. The result will be that as you update one form, the other is updated as well.

The purpose of the exercise is to provide you with a chance to get and set form element values in a variety of ways. The code for the example is in Listings 13.1, 13.2, and 13.3.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson13, lesson13/js, and lesson13/css folders, and then add the lesson13/form_manipulation.html, lesson13/js/form_manipulation.js, and lesson13/css/form_manipulation.css files.

2. Add the code shown in Listing 13.1 and Listing 13.3 to the HTML and CSS files. There is a lot of HTML code, but it is all just defining form elements. Notice that there are two forms: formA and formB.

3. Open the form_manipulation.js file and a basic `.ready()` function that you will add lines of code to that implement event handlers to update `formB` from `formA` data.

4. Add the following line that attaches a `keyup` handler to the text input so that when it is changed in `formA`, the value also changes in `formB`. The values are retrieved and set by `.val()` method:

Click here to view code image

```
$("#formA input:text").keyup(function(){
  $("#formB input:text").val($(this).val());});
```

5. Add the following lines that attach a keyup handler to the text and `<textarea>` inputs so that when it is changed in `formA`, the value also changes in formB. The values are retrieved and set by `.val()` method:

Click here to view code image

```
$("#formA textarea").keyup(function(){
  $("#formB textarea").val($(this).val());});
```

6. Add the following lines that attach a change handler to the radio input group. Notice that to get the same radio input element in the other form, you need to get the input element with the same value using `$("#formB input[value=" + $(this).val() + "]");`. The value is set using `.prop()`:

Click here to view code image

```
$("#formA input:radio").change(function(){
    var radioB = $("#formB input[value=" +
        $(this).val() + "]");
    radioB.prop("checked", $(this).is(":checked"));
  });
```

7. Add the following lines that attach a click handler to the check box in `formA` and uses the `.prop()` method to check the same check box in `formB` when clicked:

Click here to view code image

```
$("#formA input:checkbox").click(function(){
    $("#formB input:checkbox").prop("checked",
        $(this).prop("checked"));
  });
```

8. Add the following line that attaches a click handler to the selection in `formA` so that when the selection changes, the `.val()` can be called to get the value from `formA` and set `formB`:

Click here to view code image

```
$("#formA select").change(function(){
  $("#formB select").val($(this).val());});
```

9. Add the following lines that attach a `click` handler to the image input so that when it is clicked, the `src` attribute of the image input in `formB` will be changed to match:

Click here to view code image

```
$("#formA input:image").click(function(e){
    $("#formB input:image").attr("src", $(this).attr("src"));
    e.preventDefault();
  });
```

10. Add the following click handler for the Reset button. The handler calls the `.reset()` function on `formB` by getting the `formB` DOM element using `$("#formB").get(0)`. It then removes the `check` attribute from all checked elements and resets the `src` attribute of the image element:

Click here to view code image

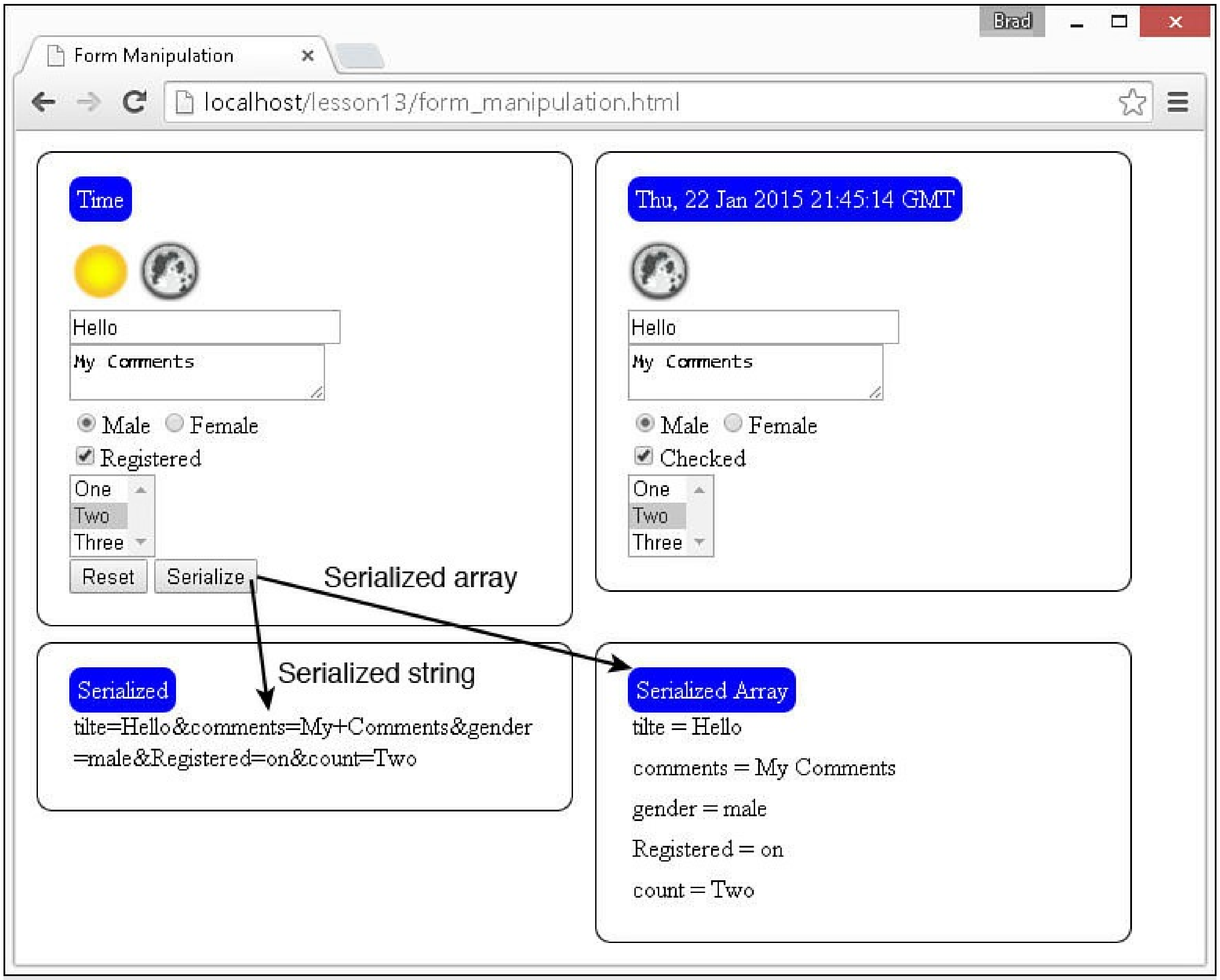
```
$("#resetB").click(function(){
    $("#formB").get(0).reset();
    $("#formB input:checked").prop("checked", false);
    $("#formB input:image").attr("src", "");
  });
```

11. Add the following click handler for the Serialize button. The handler first calls `.serialize()` on form and writes the string out to the serialized paragraph element. Then it retrieves a serialized array by calling `.serializeArray()`. The `jQuery.each()` method is used to iterate through the array and append a new paragraph with name and value pair to the `serializedA` paragraph:

Click here to view code image

```
$("#serializeB").click(function(e){
    $("#serialized").html($("#formA").serialize());
    $("#serializedA").empty();
    var arr = $("#formA").serializeArray();
    jQuery.each(arr, function(i, prop){
      $("#serializedA").append($("<p>" + prop.name + " = " +
                                 prop.value + "</p>"));
    });
  });
```

12. Save all three files and then open the HTML document in a web browser, as shown in Figure 13.3. You should be able to change the elements in the left form and see them also change in the right. When you click the Serialize button, the two `<div>` elements at the bottom should be populated with the serialized data from `formA`.



tilte=Hello&comments=My+Comments&gender=male&Registered=on&count=Two

tilte = Hello

comments = My Comments

gender = male

Registered = on

count = Two

FIGURE 13.3 Form-to-form manipulation illustrating how to read and write data to forms as well as serialize the form values.

LISTING 13.1 form_manipulation.html HTML Document That Implements the Form Elements Used in the Example

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/form_manipulation.js">
</script>
    <link rel="stylesheet" type="text/css"
href="css/form_manipulation.css">
```

```
  </head>
  <body>
      <div><form id="formA">
        <label>Time</label><br>
        <input type="image" src="/images/day.png" />
        <input type="image" src="/images/night.png" /><br>
        <input name="tilte" type="text" /><br>
        <textarea name="comments"></textarea><br>
        <input type="radio" name="gender" value="male">Male
        <input type="radio" name="gender" value="female">Female<br>
        <input type="checkbox" name="Registered">Registered<br>
        <select size=3 multiple name="count">
          <option>One</option><option>Two</option><option>Three</option>
        </select><br>
        <input id="resetB" type="button" value="Reset"></input>
        <input id="serializeB" type="button" value="Serialize"></input>
      </form></div>
      <div><form id="formB">
        <label>Destination</label><br>
        <input type="image" alt="No Image"></input><br>
        <input type="text" /><br>
        <textarea></textarea><br>
        <input type="radio" name="gender" value="male">Male</input>
        <input type="radio" name="gender" value="female">Female</input><br>
        <input type="checkbox">Checked</input><br>
        <select size=3 multiple>
          <option>One</option><option>Two</option><option>Three</option>
        </select>
      </form></div><br>
      <div><label>Serialized</label><p id="serialized"></p></div>
      <div><label>Serialized Array</label><span id="serializedA"></span></div>
  </body>
</html>
```

LISTING 13.2 form_manipulation.js jQuery and JavaScript Code That Implements a Series of Event Handlers That Read Data from an Element in One Form as It Changes and Updates the Second

Click here to view code image

```
$(document).ready(function(){
  $("#formA input:text").keyup(function(){
    $("#formB input:text").val($(this).val());});
  $("#formA textarea").keyup(function(){
    $("#formB textarea").val($(this).val());});
  $("#formA input:radio").change(function(){
      var radioB = $("#formB input[value=" +
          $(this).val() + "]");
      radioB.prop("checked", $(this).is(":checked"));
    });
  $("#formA input:checkbox").click(function(){
      $("#formB input:checkbox").prop("checked",
          $(this).prop("checked"));
    });
  $("#formA select").change(function(){
    $("#formB select").val($(this).val());});
  $("#formA label").click(function(){
      $("#formB label").html(new Date().toUTCString());
    });
  $("#formA input:image").click(function(e){
      $("#formB input:image").attr("src", $(this).attr("src"));
      e.preventDefault();
    });
  $("#resetB").click(function(){
      $("#formB").get(0).reset();
      $("#formB input:checked").prop("checked", false);
      $("#formB input:image").attr("src", "");
    });
  $("#serializeB").click(function(e){
      $("#serialized").html($("#formA").serialize());
      $("#serializedA").empty();
      var arr = $("#formA").serializeArray();
      jQuery.each(arr, function(i, prop){
        $("#serializedA").append($("<p>" + prop.name + " = " +
                                   prop.value + "</p>"));
      });
    });
});
```

LISTING 13.3 form_manipulation.css CSS Code That Styles the Form Elements

Click here to view code image

```
input[type=image] {height:40px; margin-top:15px;}
div{
  vertical-align:top; width:300px; height:auto;
  display:inline-block; padding:20px; margin:5px;
  border-radius:10px; border:1px solid;
}
label{ background-color:blue; color:white;
  border-radius:8px; padding:5px; }
p { margin:1px; padding:2px; width: 100%;
  border-radius:8px; display:inline-block;
  word-wrap: break-word; }
span {width:300px;}
```

Intelligent Form Flow Control

Another important aspect of dynamic form control is dynamically helping the user navigate through the web form. This is especially true when working with more complex web forms. The way that you help users navigate through the web form is by automatically changing the focus for them, hiding elements that become irrelevant, showing new elements when needed, and disabling elements that cannot be changed.

Using these methods provide the web form with an interactive and rich feel. This will make the web form a lot easier for the user to go through. The following sections discuss the basic concepts of web form control.

Automatically Focusing and Blurring Form Elements

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

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

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

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

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

Intelligently Hiding and Showing Elements

Another great trick when providing flow control for a web form is to dynamically hide and show elements. In effect, less is more, meaning that you shouldn't necessarily show users more than the elements they will need to fill out.

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

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

Disabling Elements

Disabling web elements will still display them, but the user will not be able to interact with them. Typically, it makes sense to disable a form element instead of hiding it only if you still want the user to be able to see the values of the elements.

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

Click here to view code image

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

To reenable a disabled element, you remove the `disabled` value. For example:

Click here to view code image

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

Controlling Submit and Reset

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

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

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

Click here to view code image

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

jQuery does not provide an event handler for the form `reset` event for some reason. To get past this in jQuery, change the input `type` from `reset` to `button` for the Reset button. Then add a `click` handler to the button event where it will call `.reset()` on the DOM element of the form. The following code does just that based on a user prompt:

Click here to view code image

```
$("#resetB").click(function(e){
    if(confirm("Are you sure?")){ $("form").get(0).reset(); }
  });
```

Try it Yourself: Adding Dynamic Flow Control to Forms

Now that you have had a chance to review the concepts of form flow control, you are ready to try some yourself. In this exercise, you add some basic flow control to an e-commerce web form for accepting payments.

The purpose of the exercise is to provide you with experience hiding, showing, disabling, and autofocusing web elements as the user interacts with the form. The code for the example is in Listings 13.4, 13.5, and 13.6.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson13/form_flow.html, lesson13/js/form_flow.js, and lesson13/css/form_flow.css files.
2. Add the code shown in Listing 13.4 and Listing 13.6 to the HTML and CSS files. There is a lot of HTML and CSS code, but you should be able to follow it by now. The code defines and styles a basic payment form.
3. Open the form_flow.js file and create the `.ready()` function shown in Listing 13.4. You should recognize the first few lines as just adding click handlers, which is discussed later. Add the lines shown next and implement the flow control for submitting and resetting the form. The `submit` handler displays a message that the form submission is not implemented and prevents the default submit. The `click` handler for the reset button prompts the user with a message before calling `.reset()` to reset the form:

Click here to view code image

```
$("form").submit(function(e){
    alert("Sorry. Not yet Implemented.");
    e.preventDefault();
  });
$("#resetB").click(function(e){
    if(confirm("Are you sure?")){ $("form").get(0).reset(); }
  });
```

4. Add the `updateAddr()` function that will update the billing address to match that of the shipping and disable the billing elements if the check box is selected. If the check box is not selected, line 22 will remove the `disabled` value so that the user can choose a different billing address:

Click here to view code image

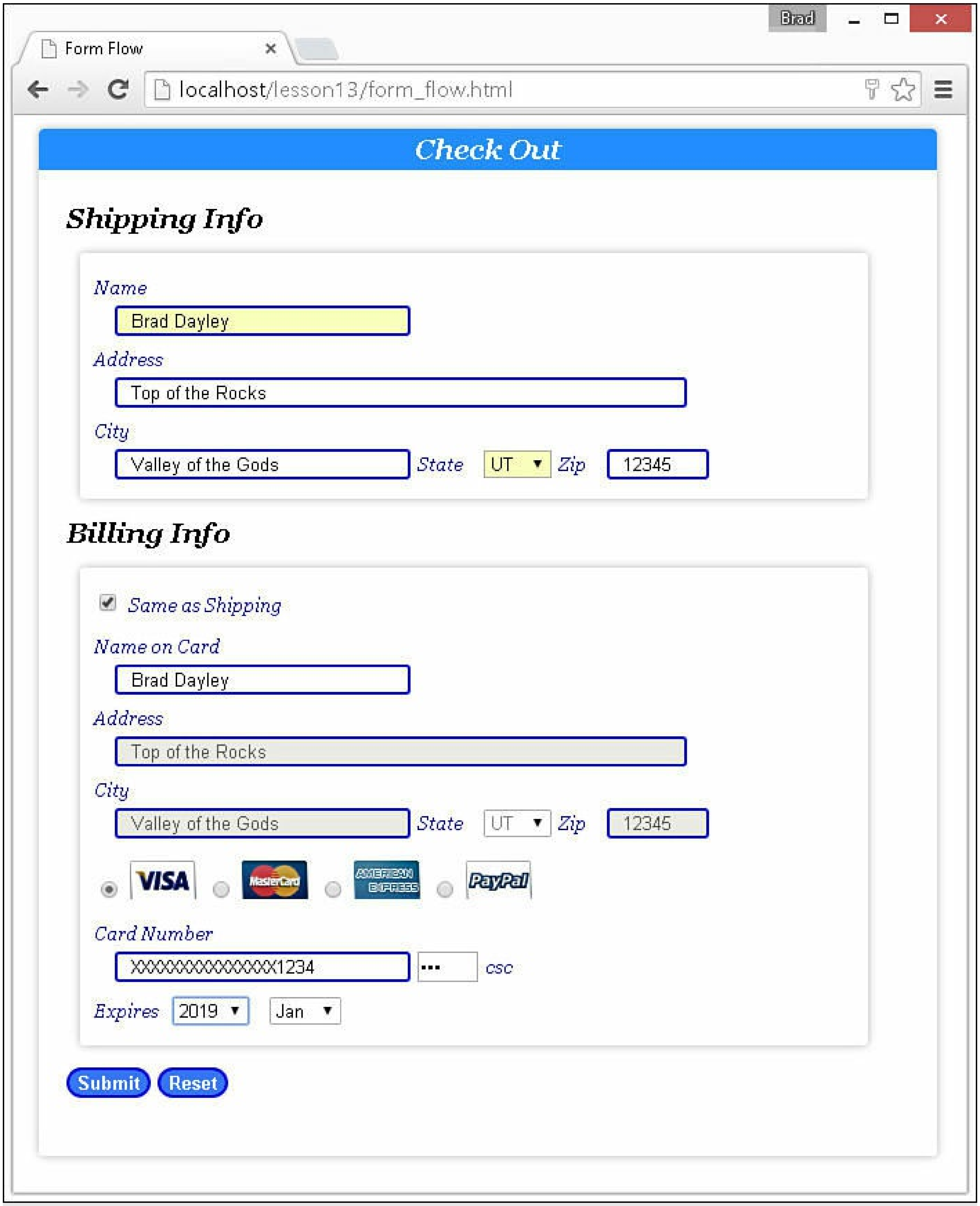
```
13 function updateAddr(){
14   var cb = $("#cbSame");
15   if (cb.prop("checked")){
16     $("#nameB").val($("#name").val());
17     $("#addrB").val($("#addr").val());
18     $("#cityB").val($("#city").val());
19     $("#stateB").val($("#state").val());
20     $("#zipB").val($("#zip").val());
21     $("#addrB, #cityB, #stateB, #zipB").prop("disabled",
"disabled");
22   } else{ $("#addrB, #cityB, #stateB, #zipB").prop("disabled", "");
}
}
```

5. Add the following function to update the payments section when the user selects a credit card or PayPal. Notice that when users select a credit card, the PayPal information is hidden and the credit card information is shown; the reverse occurs if PayPal is selected. Also notice that the focus is changed in both instances to the first text element in the area where the user will begin entering payment information. That way, the user doesn't need to tab to or click the next element:

Click here to view code image

```
function updatePaymentType(){
  if(this.id == "ppal"){
    $("#ccInfo").hide();
    $("#ppInfo").show();
    $("#ppEmail").focus();
  } else {
    $("#ppInfo").hide();
    $("#ccInfo").show();
    $("#cardNum").focus();
  }
}
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 13.4. Populate the shipping information and then see it populate the billing information when the same check box is selected. Also try selecting different payment types and watch the elements hide and the autofocus work.


Form Flow

localhost/lesson13/form_flow.html

Check Out

Shipping Info

Name	Brad Dayley
Address	Top of the Rocks
City	Valley of the Gods
State	UT
Zip	12345

Billing Info

☑ Same as Shipping

Name on Card	Brad Dayley
Address	Top of the Rocks
City	Valley of the Gods
State	UT
Zip	12345

[✓ VISA] MasterCard AMERICAN EXPRESS PayPal

Card Number	XXXXXXXXXXXXXXX1234
csc	•••
Expires	2019 Jan

FIGURE 13.4 Basic e-commerce web form with intelligent flow control as the user enters data.

LISTING 13.4 form_flow.html HTML Document That Implements the Payment Form Used in the Example

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Flow</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/form_flow.js"></script>
    <link rel="stylesheet" type="text/css" href="css/form_flow.css">
  </head>
  <body>
    <div id="box">
      <p>Check Out</p>
      <form>
        <span>Shipping Info</span><br>
          <div id="billInfo">
          <label class="headLabel">Name</label>
          <input type="text" id="name"/><br>
          <label class="headLabel">Address</label>
          <input type="text" id="addr" /><br>
          <label class="headLabel">City</label>
          <input type="text" id="city" />
          <label>State</label>
          <select class="state" id="state"></select>
          <label>Zip</label><input type="text"  id="zip"/><br>
        </div>
        <span>Billing Info</span><br>
        <div id="billInfo">
          <input type="checkbox" id="cbSame"/>
            <label for="cbSame">Same as Shipping</label><br>
          <label class="headLabel">Name on Card</label>
          <input type="text"  id="nameB"/><br>
          <label class="headLabel">Address</label>
          <input type="text"  id="addrB"/><br>
          <label class="headLabel">City</label>
          <input type="text"  id="cityB"/>
          <label>State</label>
          <select class="state"  id="stateB"></select>
          <label>Zip</label><input type="text"  id="zipB"/><br>
          <input type="radio" name="ptype" id="visa" />
            <label for="visa"><img src="/images/visa.png"/></label>
          <input type="radio" name="ptype" id="mc" />
            <label for="mc"><img src="/images/mc.png" /></label>
          <input type="radio" name="ptype" id="amex" />
            <label for="amex"><img src="/images/amex.png"/></label>
          <input type="radio" name="ptype" id="ppal" />
            <label for="ppal"><img src="/images/ppal.png"/></label>
          <br>
          <div id="ccInfo">
            <label class="headLabel">Card Number</label>
            <input type="text" id="cardNum"/>
            <input type="password"  id="csc"/>
            <label>csc</label><br>
            <label>Expires</label><select id="expiresY"></select>
            <select id="expiresM"></select><br>
          </div>
          <div id="ppInfo">
            <input type="text"  id="ppEmail"/>
            <label>PayPal Email</label><br>
            <input type="text"  id="ppPW"/>
            <label>PayPal Password</label><br>
          </div>
        </div>
        <input type="submit" value="Submit" id="submitB" />
        <input type="button" value="Reset" id="resetB" />
      </form>
    </div>
  </body>
</html>
```

LISTING 13.5 form_flow.js jQuery Code That Provides the Intelligent Flow Control for the Payment Form

Click here to view code image

```
var months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
              "Aug", "Sep", "Oct", "Nov", "Dec" ];
var sArr = ["AK","AL","AR","AS","AZ","CA","CO","CT","DC","DE","FL",
  "GA","GU","HI","IA","ID","IL","IN","KS","KY","LA","MA","MD","ME","MH",
  "MO","MS","MT","NC","ND","NE","NH","NJ","NM","NV","NY","OH","OK","OR",
  "PW","RI","SC","SD","TN","TX","UT","VA","VI","VT","WA","WI","WV","WY"

function buildSelects(){
  for(var i in sArr){ $("#state, #stateB").append($('<option>'+sArr[i]+'</option>"')); }
  for(var i in months){ $("#expiresM").append($('<option>'+months[i]+'</option>"')); }
  for(var y=2015; y<2020;y++){ $("#expiresY").append($('<option>'+y+'</option>"')); }
}
function updateAddr(){
  var cb = $("#cbSame");
  if (cb.prop("checked")){
    $("#nameB").val($("#name").val());
    $("#addrB").val($("#addr").val());
    $("#cityB").val($("#city").val());
    $("#stateB").val($("#state").val());
    $("#zipB").val($("#zip").val());
    $("#addrB, #cityB, #stateB, #zipB").prop("disabled", "disabled");
  } else{ $("#addrB, #cityB, #stateB, #zipB").prop("disabled", ""); }
}
function updatePaymentType(){
  if(this.id == "ppal"){
    $("#ccInfo").hide();
    $("#ppInfo").show();
    $("#ppEmail").focus();
  } else {
    $("#ppInfo").hide();
    $("#ccInfo").show();
    $("#cardNum").focus();
  }
}
$(document).ready(function(){
  $("#ppInfo").hide();
  buildSelects();
  $("#cbSame").click(updateAddr);
  $("input:radio").click(updatePaymentType);
  $("form").submit(function(e){
      alert("Sorry. Not yet Implemented.");
      e.preventDefault();
    });
  $("#resetB").click(function(e){
      if(confirm("Are you sure?")){ $("form").get(0).reset(); }
    });
});
```

LISTING 13.6 form_flow.css CSS Code That Styles the Payment Form Elements

Click here to view code image

```
input[type=text] {
   width:200px; margin-left:15px; padding-left:10px;
   border-radius: 3px; border:2px groove blue;}
select {margin-left:10px}
img {margin-top:10px; }
#addr, #addrB { width:400px; }
#zip, #zipB { width:60px ; }
#csc { width:40px; }
#box, #billInfo, #shipInfo{
   font:italic 20px/30px Georgia, serif;
   width:650px; height:auto; padding-bottom:20px; margin:10px;
   border-radius: 3px; box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3); }
#billInfo, #shipInfo { width:550px; padding:10px;}
#submitB, #resetB{
   background-color:#3377FF; color:white; font-weight:bold;
   border:2px groove blue; border-radius:15px; }
p{ color:white; background-color:dodgerblue;
   font-weight:bold; margin:0px;
   text-align:center; border-radius: 4px 4px 0px 0px ; }
span{ dispaly:inline-block;
  margin-left:-15px -15px; color:black; font-weight:bold; }
```

```
form{ padding:20px; }
.headLabel{ display:block; margin-bottom:-8px;}
label{ color:#000088; font-size:14px;}
```

Dynamically Controlling Form Element Appearance and Behavior

In addition to the intelligent flow control, it is also helpful to provide visual indicators to users about what is happening as they interact with the form. These indicators may adjust font or element sizes, change classes, add/change borders, and so on.

Many of the changes can be made by simple CSS settings based on element states, such as `:hover` or `:checked`. However, with the capability to add animations to these changes, you can give your forms more of a rich application look and feel.

Try it Yourself: Adding Animated Elements to Improve User Experience

You already have all the tools necessary to add whatever graphical interactions to your web forms that you want. The purpose of this section is to give you an idea of some of the things that you can do. The example is a very basic registration form, but includes a select, styled radios and check boxes, and some other form effects. The code is designed to give you a chance to see several techniques in the same example with the smallest amount of code. Feel free to expand on any of these as you play around with the example.

The code for the example is in Listings 13.7, 13.8, and 13.9. Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson13/form_effects.html, lesson13/js/form_effects.js, and lesson13/css/form_effects.css files.
2. Add the code shown in Listing 13.7 and Listing 13.9 to the HTML and CSS files. The HTML is a pretty basic form. The CSS is fairly big and has a few intermediate selectors that are based on attribute values, but you should be able to follow it. The CSS styles the radio inputs to look like buttons by hiding the radio and showing only a styled label. A similar technique is used for the check box displaying an image instead of a check box.
3. Open the form_effects.js file and create a basic `.ready()` function to implement the handlers in the following steps.
4. Add the following lines of code that hide the form and then slowly show it as the user clicks the header, and then hides it when the form is submitted:

Click here to view code image

```
  $("form").hide();
  $("p").click(function(){$("form").toggle(1000);
    return false;});
  $("input:submit").mousedown(function(){
    $("form").toggle(1000); return false;});
```

5. Add the following lines that will animate resizing the `<textarea>` element when it is in and out of focus:

Click here to view code image

```
  $("textarea").focus(function(){
    $(this).animate({width:350, height:100}, 1000);});
  $("textarea").blur(function(){
    $(this).animate({width:200, height:50}, 1000);});
```

6. Add the following handler for when the radio inputs change. The handler animates the opacity down to .1; then when that is complete, it changes the class to a darker color and animates the opacity back up. This gives the appearance of clicking the button. With the CSS style, this makes the radio input group act like a button bar:

Click here to view code image

```
function changeRadio(){
  $(this).animate({opacity:.1}, 400, function(){
    $("input:radio").next("label").removeClass("rb_checked");
    $(this).addClass("rb_checked");
    $(this).animate({opacity:1}, 800);
  });
}
```

7. Add the following handler for when the check boxes are clicked. The handler animates changing the size of the image, border, and opacity, giving the element a visual indicator of whether the character has a horse or armor:

Click here to view code image

```
function changeCheckbox(){
  var checkbox = $("#"+$(this).attr("for"));
  if(checkbox.prop("checked")){
    $(this).children("img").animate({opacity:.25, height:20, "border-size":1}, 500);}
  else {
    $(this).children("img").animate({opacity:1, height:40, "border-size":.5}, 500); }
}
```

8. Add the following lines that attach a handler to the text inputs so that as the user types, the labels are replaced with the text content:

Click here to view code image

```
  $("input:text").keyup(function(){
      $(this).next("label").html($(this).val());});
```

9. Add the code in lines 32–40 that animate the Submit button size as it is in focus or hovered over by the user.

10. Add the following lines that attach a `mouseover` handler to the `optgroup` element and animate a font size increase as the mouse hovers over them, giving users a better idea of what they will select. We use the `optgroup` for animating the font size because the `option` element alone won't allow dynamic editing of font styles:

Click here to view code image

```
41   $("optgroup").mouseover(function(){
42     $(this).stop(true).animate({"font-size":"18px"}, 200);
43     return false;});
44   $("optgroup").mouseout(function(){
45     $(this).stop(true).animate({"font-size":"15px"}, 200);
46     return false;});
```

11. Save all three files and then open the HTML document in a web browser, as shown in Figure 13.5. You should be able interact with the web form and see the visual elements that you have implemented.

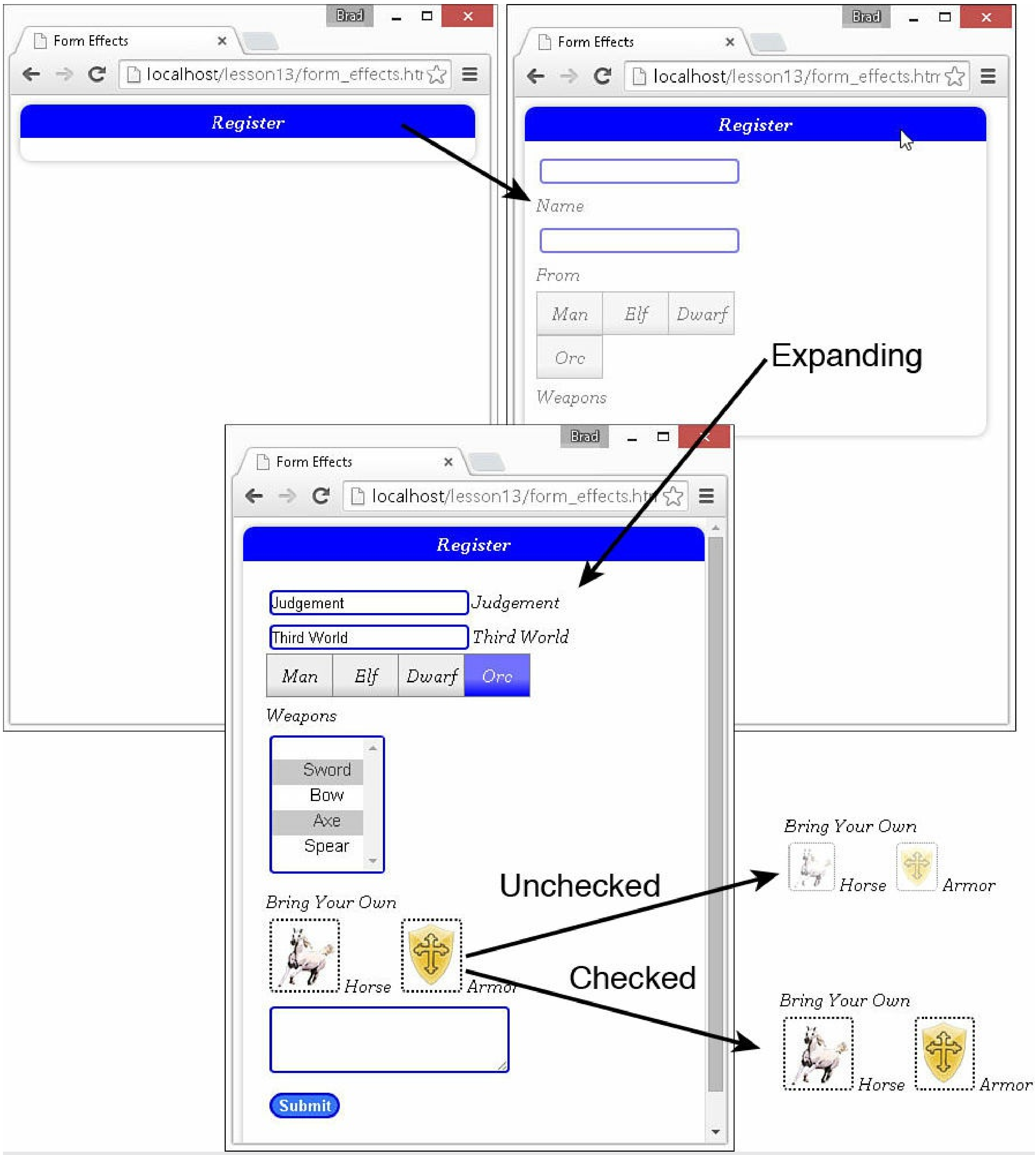


FIGURE 13.5 A simple registration form that provides some dynamic visual effects to make the form more interactive and user friendly.

LISTING 13.7 form_effects.html HTML Document That Implements the Registration Form Used in the Example

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Effects</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/form_effects.js"></script>
    <link rel="stylesheet" type="text/css" href="css/form_effects.css">
  </head>
  <body>
    <div class="collapsibleForm">
      <p>Register</p>
      <form>
        <input type="text" /><label>Name</label><br>
        <input type="text" /><label>From</label><br>
        <input type="radio" name="race" id="man" />
        <label class="rb" for="man">Man</label>
        <input type="radio" name="race" id="elf" />
        <label class="rb" for="elf">Elf</label>
        <input type="radio" name="race" id="dwarf" />
        <label class="rb" for="dwarf">Dwarf</label>
        <input type="radio" name="race" id="orc" />
        <label class="rb" for="Orc">Orc</label><br>
        <label>Weapons</label><br>
        <select size=4 multiple>
          <optgroup><option>Sword</option></optgroup>
          <optgroup><option>Bow</option></optgroup>
          <optgroup><option>Axe</option></optgroup>
          <optgroup><option>Spear</option></optgroup>
        </select>
        <div>
          <label>Bring Your Own</label><br>
            <input type="checkbox" id="horse" />
              <label class="cb" for="horse">
              <img src="/images/horse.png" />
              <span>Horse</span></label>
            <input type="checkbox" id="shield" />
              <label class="cb" for="shield">
              <img src="/images/shield.png" />
              <span>Armor</span></label><br>
         </div>
        <textarea>Additional Comments</textarea><br>
        <input type="submit" value="Submit" id="submit" />
      </form>
    </div>
  </body>
</html>
```

LISTING 13.8 form_effects.js jQuery Code Implements the Animated Visual Elements

Click here to view code image

```
function changeCheckbox(){
  var checkbox = $("#"+$(this).attr("for"));
  if(checkbox.prop("checked")){
    $(this).children("img").animate({opacity:.5, height:40,
      "border-width":"1px"}, 500);}
  else {
    $(this).children("img").animate({opacity:1, height:60,
      "border-width":"2px"}, 500); }
}
function changeRadio(){
  $(this).animate({opacity:.1}, 400, function(){
    $("input:radio").next("label").removeClass("rb_checked");
    $(this).addClass("rb_checked");
    $(this).animate({opacity:1}, 800);
  });
}
$(document).ready(function(){
  $("form").hide();
  $("p").click(function(){$("form").toggle(1000);
  return false;});
  $("input:submit").mousedown(function(){
    $("form").toggle(1000); return false;});
  $("textarea").focus(function(){
    $(this).animate({width:350, height:100}, 1000);});
  $("textarea").blur(function(){
    $(this).animate({width:200, height:50}, 1000);});
  $(".rb").click(changeRadio);
  $("input:checkbox").prop("checked",false);
  $(".cb").click(changeCheckbox);
  $("input:text").keyup(function(){
      $(this).next("label").html($(this).val());});
  $("#submit").mouseover(function(){
    $(this).animate({"background-color":"#0000FF",
      "width":"140px"}, 400, "linear");});
  $("#submit").mouseout(function(){
    $(this).animate({"background-color":"#3377FF",
      "width":"60px"}, 400, "linear");});
  $("#submit").focus(function(){
    $(this).animate({"background-color":"#0000FF",
      "border-width":"5px"}, 400, "linear");});
  $("optgroup").mouseover(function(){
    $(this).stop(true).animate({"font-size":"18px"}, 200);
    return false;});
  $("optgroup").mouseout(function(){
    $(this).stop(true).animate({"font-size":"15px"}, 200);
    return false;});
});
```

LISTING 13.9 form_effects.js CSS Code That Styles the Form Elements and Provide Classes for Dynamic Adjustments

Click here to view code image

```
br{clear:both;}
option, optgroup { padding:0px; margin:0px; height:22px; }
select, textarea, input{
  border-radius: 4px; margin:3px;
  border:2px groove blue; }
select:focus, textarea:focus, input:focus {
  border-radius: 4px; margin:3px;
  border:2px groove #3377FF; }
select{height: 120px; width:100px; text-align:center;}
img{}
input[type="radio"], input[type="checkbox"] {display:none;}
.rb {
  background: -moz-linear-gradient(bottom, #CCCCCC, #EEEEEE 10px);
  background: -webkit-linear-gradient(bottom, #CCCCCC, #EEEEEE 10px);
  background: -ms-linear-gradient(bottom, #CCCCCC, #EEEEEE 10px);
  width: 50px; padding: 3px;
  margin-right:-1px; display: inline-block;
  text-align:center; float:left; border: 1px solid gray; }
.rb_checked {
  background: -moz-linear-gradient(bottom, #0000FF, #7272FF 15px);
  background: -webkit-linear-gradient(bottom, #0000FF, #7272FF 15px);
  background: -ms-linear-gradient(bottom, #0000FF, #7272FF 15px);
  color:white;
}
.cb { padding: 3px; margin-right:-1px; width:100px; }
.cb img{ border:1px dotted; border-radius:4px;
  opacity:.5; height:40px; }
#submit{
  background-color:#3377FF; color:white; font-weight:bold;
  border:2px groove blue; border-radius:15px; }
.collapsibleForm{
  width:400px; height:auto; padding-bottom:20px;
  border-radius: 10px; box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
  font:italic 15px/30px Georgia, serif;}
p{
  color:white; background-color:blue; font-weight:bold;
  margin:0px; text-align:center;
  border-radius: 10px 10px 0px 0px ; }
form{ padding:20px; }
```

Validating a Form

One of the most important parts of web forms is validating the information that is being entered. There are many different types and levels of validation depending on the data being entered.

Some elements, for example, are totally optional and require no validation. Others, such as a comment, may be required but the contents do not really matter. Others, such as an email address or data, may be required and must adhere to specific formatting requirements, perhaps even value ranges.

Often, this is done by a server-side script. The problem with that method is that all the data must be sent to the server and then the error is formulated and returned back to the user.

A much better option is to validate the form data in your jQuery code before it is submitted to the server. The following sections describe the different methods for validating form data.

Manually Validating a Web Form

The most basic way of validating forms is by accessing their values and checking the actual contents against the requirements. This can be done when the user is entering data by adding, for example, a `blur`, `change`, or `keypress` event handler, and then inside the event handler checking the value of the data. The following code implements validation in a `keypress` handler, validating that the value of the form element is numeric, and if it isn't, adding warning text to a label:

Click here to view code image

```
$("input:text").keypress(function (){
    if(!$.isNumeric($(this).val())) {
      $("#validLabel").html("Not a Number"));
    }
  });
```

You can also implement the validation when the user submits the form by adding the validation to a `submit` handler. For example, the following code adds a validation function to a submit handler so that if the element does not begin with the text `"vfx"`, an `alert` is displayed and the form is not submitted:

Click here to view code image

```
$("form").submit(function (){
    if (!$("#vfxField").val().match("^vfx")){
      alert("Invalid vfx element");
      e.preventDefault();
    }
  });
```

Getting the jQuery Validation Plug-In

Using the techniques in the previous section, you can validate pretty much any type of form element. The problem is that it takes a lot of code and time to add validation. Rather than reinventing the wheel, you can use the jQuery validation plug-in that takes care of most of the validation needs.

As a plug-in, the validator is implemented by loading the .js file after the jQuery.js file has been loaded. For example:

Click here to view code image

```
<script type="text/javascript" src="../js/jquery.min.js"></script>
<script type="text/javascript" src="js/jquery.validate.min.js"></script>
```

The plug-in can be downloaded from:

http://jqueryvalidation.org/

The documentation for the validation plug-in can be found at:

http://docs.jquery.com/Plugins/Validation/

Applying Simple jQuery Validation Using HTML

The validator can be implemented in HTML using the class and title attributes. The validator uses a set of rules, such as `required` or `email`, to validate the form element. Setting the `class` attribute to one or more of these rules applies the validation when `.validate()` is called on the form. Table 13.3 lists several of the validation rules.

Rule	Value Source/Usage
`accept`	Element requires a certain file extension. `accept:"jpg|png|gif"`
`creditcard`	Element requires a valid credit card number. `creditcard:true`
`date`	Element requires a valid date. `date:true`
`dateISO`	Element requires a valid ISO date. `dateISO:true`
`digits`	Element requires digits only. `digits:true`
`email`	Element requires a valid email. `email:true`
`equalTo`	Requires the element to have the same value as another. `equalTo:"#passwordVerify"`
`min`	Element value requires a given minimum. `min:10`
`minlength`	Element value string length must be more than. `minlength:8`
`max`	Element value requires a given maximum. `max:100`
`maxlength`	Element value string length must be less than. `maxlength:12`
`number`	Element requires a decimal number. `number:true`

`rangelength`	Element value string length must fall in a specific range. `rangelength:[3,9]`
`remote`	Will execute an HTTP AJAX request that calls a server-side script that must return back a valid JSON response as true or false. The parameter name and value will be passed to the server-side script as the query string in GET request. `remote:"validation.php"`
`required`	Element requires a value. `required:true`
`range`	Element requires a given value range. `range:[1,5]`
`url`	Element requires a valid URL. `url:true`

TABLE 13.3 Validation Rules and Usage

For example, to validate that a text element has text and is in email format, you use the following code:

HTML:

Click here to view code image

```
<form id="myForm">
  <input type="text" class="required email" />
</form
```

jQuery:

Click here to view code image

```
$(document).ready(function(){
    $("#myForm").validate();
  };
```

The `.ready()` function adds the validation to the form, and as the user types in the element, a message displays that the element is required if it is empty and that a valid email address is required if it does not contain a valid email address, as shown in Figure 13.6.

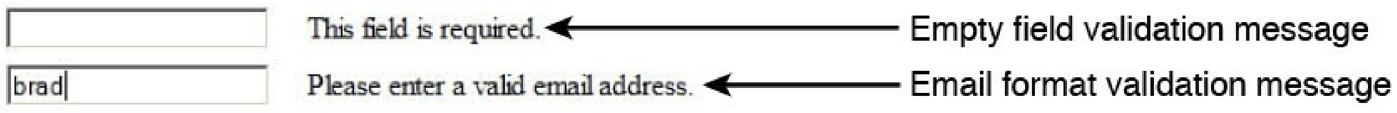

This field is required.
Empty field validation message
brad
Please enter a valid email address.
Email format validation message

FIGURE 13.6 Validation messages of an email element.

You can also add your own text messages instead of the defaults provided by the validation library. This is done by setting the title attribute to the message string you want to display. For example, the following input statement adds the title element that will be displayed if the element is invalid:

Click here to view code image

```
<input type="text" class="required email"
  title="This Element requires a valid Email Address such as
name@here.com"/>
```

Applying Complex Validation

A better way to add validation using the validation library is inside the jQuery validate method. The validate method accepts an object that defines how a form is validated. This section covers using the values of rules and messages. The validation object supports several attributes; you can review the doc to see rest.

Adding Validation Rules

The rules method allows you to define the different rules, listed in Table 13.3, that apply to a form element by referencing the `name` attribute. For example, the following code adds the `required` and `email` rules to an element with `name="emailField"` using the validator object:

HTML:

Click here to view code image

```
<form id="myForm">
  <input type="text" name="emailField" />
</form
```

jQuery:

Click here to view code image

```
$(document).ready(function(){
    $("#myForm").validate({
        emailField: {
          required:true,
          email:true
        }
      );
  };
```

Tip

The `validate()` method accepts a debug option that will disable form submission, allowing you to implement and test the form validation without submitting data to the server. To implement the form validation debug mode, use the following:

```
$(form).validate({
        debug:true,
        other options . . .
      });
```

Adding Validation Messages

The messages attribute of the validation object allows you to specify custom messages that will be applied to the individual rules for the element. The `messages` attribute is also based on the `name` attribute of the element. The following code illustrates adding messages to the validation:

Click here to view code image

```
$(document).ready(function(){
  $("#simpleForm").validate(
    {rules: {
      emailField: {
        required:true,
        email:true,
        accept:"jpg|cvs"
        }
      },
      messages: {
        emailField: {
          required:"Must add Email",
          email:"Email format = me@here.com"
        }
      }
    }
  );
};
```

The `.validate()` method returns a `Validator` object, which provides several methods that are valuable for checking form validation. Table 13.4 lists the methods available from the `Validator` object. The following code illustrates using the `Validator` object to check the validity of the form by calling the `.form()` method:

Click here to view code image

```
$(document).ready(function(){
  var validator = $("#simpleForm").validate( ...);
  if(!validator.form()){ alert("Form is not valid"));
});
```

Rule	Value Source/Usage
`element( element)`	Returns true if the element is valid; otherwise, false.
`form()`	Returns true if the form elements are all valid; otherwise, false.
`numberOfInvalids()`	Returns the number of form elements failing validation.
`resetForm()`	Resets the values in the form.
`showErrors(errors)`	Shows the specified error messages.

TABLE 13.4 Validator Object Methods

When errors are detected in a form element, a label is added to the form to display the error message. The plug-in adds a class named `error` to both the element and the new label for the message. This makes it very easy to style the message as well as the element.

To style the element, you can add a CSS rule for the element type and add the `.error`. For example, to style input elements that have errors to display the text in red, you use the following CSS rule:

```
input.error { color:red; }
```

You can use the same method to style the error message displayed using the `label.error` selector. For example, the following sets the font color of the error messages to red:

```
label.error { color:red; }
```

Placing Validation Messages

The validation plug-in places the error messages directly after the element that was validated. That is often not the desired location. For example, you may want the validation to come before some elements or in a totally different location.

Note

The jQuery Validation tool provides a simple method for localizing web form validation. If you include the messages_##.js file found in the localization folder, the validation error messages will be localized. The two-digit number (##) in messages_##.js stands for the two-character ISO country code.

A good example of this is radio input groups. If you validate the group, the message appears after the first radio input element, as shown in Figure 13.7. That is not the desired location. Instead, you want the error message to come after the full set of radio inputs and their labels.

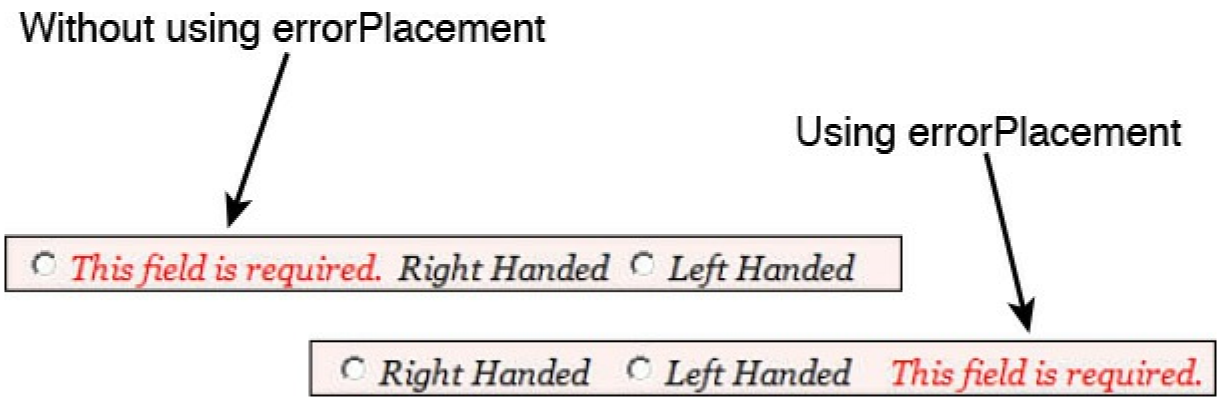

Without using errorPlacement
Using errorPlacement
This field is required. Right Handed Left Handed
Right Handed Left Handed This field is required.

FIGURE 13.7 Using the `errorPlacement` option in the `.validate()` method allows you to define the placement of the error messages.

That is where the `errorPlacement` attribute of the `Validation` object comes in handy. The `errorPlacement` attribute value is a function that gets called to add an error element to the web form. The `errorPlacement` function will be passed two parameters. The first parameter is the jQuery object that represents the new error message, and the second is a jQuery object that represents the element that failed validation. Using these parameters, you can define your own function that places the elements wherever you want them.

To illustrate this, look at the following code. The `errorPlacement` is set to a function that receives the `error` and `element` arguments. The `element` is tested to see if it is a radio. If the `element` is a radio input, the `error` is inserted after the label that follows the last radio input in the form, as shown in Figure 13.7. If the `element` is not a radio input, the `error` is inserted immediately after the `element`:

Click here to view code image

```
$("#simpleForm").validate({
  rules: {
    gender: {required:true}
  errorPlacement: function(error, element){
    if (element.is(":radio")){
      error.insertAfter($("input:radio:last").next("label"))}
    else {error.insertAfter(element)}}
});
```

Try it Yourself: Validating Web Forms Using jQuery

In this example, you use the jQuery validation plug-in to validate a basic web form. The purpose of the example is to help you understand how to implement the validation plug-in in your own scripts.

The code for the example is in Listings 13.10, 13.11, and 13.12. Use the following steps to create the web form with validation:

1. In Eclipse, create the lesson13/form_validation.html, lesson13/js/form_validation.js, and lesson13/css/form_validation.css files.
2. Add the validation plug-in. You can download the plug-in from the following location or copy the code/lesson13/js/jquery.validation.min.js file from the book’s website to lesson13/js/jquery.validation.min.js in your project:

   `http://jqueryvalidation.org/`

   You will also need to add the following line in Listing 13.10 to load the validation plug-in:

Click here to view code image

```
07     <script type="text/javascript" src="js/jquery.validate.min.js">
</script>
```

3. Add the code shown in Listing 13.10 and Listing 13.12 to the HTML and CSS files. The HTML is a basic form and shouldn’t contain any surprises. The CSS does include some specific formatting for the validation code. Line 3 formats the `<input>` and `<select>` elements to have a red background and text to make them stand out. Line 4 formats the message to be in red text:

Click here to view code image

```
03 input.error, select.error{ background-color:#FFDDDD; color:darkred}
04 label.error{color:red;}
```

4. Open the form_validation.js file and create a basic `.ready()` function to implement the handlers in the following steps. Also add a `.validate()` call on the form as shown next. Include a `rules`, `messages`, and `errorPlacement` section. Notice that the `validationObj` object variable is defined in line 2 so that we can use it later:

Click here to view code image

```
02   var validationObj = $("#simpleForm").validate({
03     rules: {
. . .
12     messages: {
. . .
17     errorPlacement: function(error, element){
. . .
23   });
02   var validationObj = $("#simpleForm").validate({
03     rules: {
. . .
11     messages: {
. . .
16     errorPlacement: function(error, element){
. . .
22   });
```

5. Add the following validation rules for the form elements. The elements are all referenced by the element's `name` attribute. The `password1` element has three rules: `required`, `rangelength`, and `equalTo`. The `equalTo` is set to the `id` attribute of the `password2` element. The `weapon` element uses a `rangelength` of `[2,2]`, which forces the user to pick exactly two weapons:

Click here to view code image

```
04       name: { required:true, minlength:5 },
05       email: { required:true, email:true },
06       password1: { required:true, rangelength:[6,12],
07                    equalTo:"#password2" },
08       birthDate: { required:true, date:true },
09       class: { required:true, rangelength:[2,2]},
10       hands: {required:true},
11       armor: {required:true, minlength:2 }},
```

6. Add the following custom messages to be more specific for the `password`, `weapon`, and `armor` elements:

Click here to view code image

```
13         password1: { equalTo:"Passwords Do Not Match"},
14         class: { rangelength:"Select 2 class types"},
15         armor: { minlength:"2 Pieces of Armor Required"},
```

7. Finish off the `errorPlacement` function. There is a radio group for `hands` and a button group for `armor`. You want the error message to be displayed after the `<label>` of the last element in the group, so you modify the `.insertAfter()` method to query for the last `checkbox` or `radio` and then get the very next `label`:

Click here to view code image

```
17     errorPlacement: function(error, element){
18       if (element.is(":radio")){
19         error.insertAfter($("input:radio:last").next("label"))}
20       else if (element.is(":checkbox")){
21         error.insertAfter($("input:checkbox:last").next("label"))}
22       else {error.insertAfter(element)}}
```

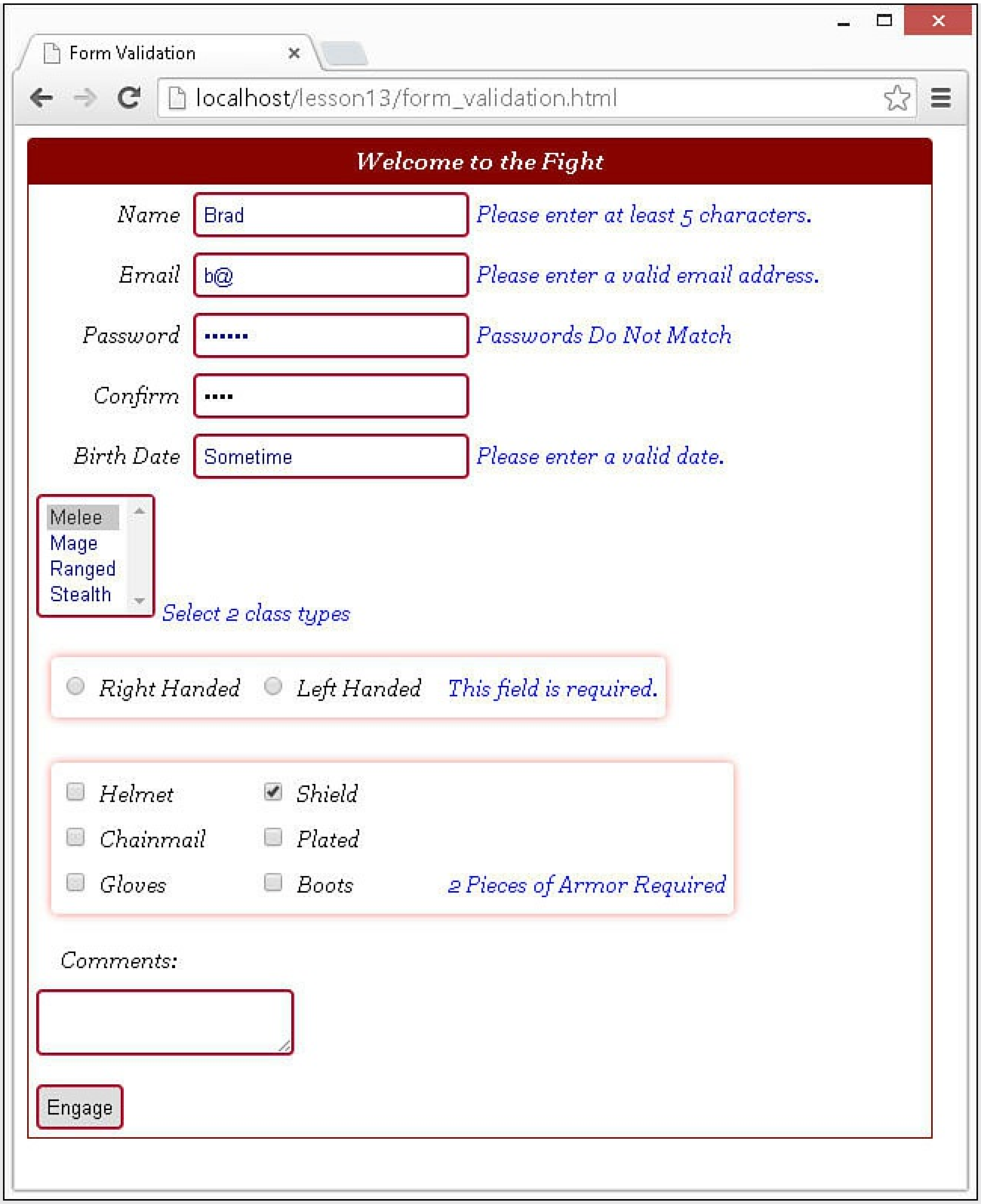
8. Add the following lines. Line 23 validates the form, which will display the requirements after the page has loaded. Lines 24 and 25 are a submit handler. The handler checks to see whether the form is valid by calling `.form()`. If the form is invalid, an alert message is displayed and the submit is prevented by `.preventDefault()`:

Click here to view code image

```
24   validationObj.form();
```

```
$("form").submit(function(e){
  if(!validationObj.form()){
    alert("Form Errors");
    e.preventDefault(); } });
```

9. Save all three files and then open the HTML document in a web browser, as shown in Figure 13.8. You should immediately see validation errors. Watch the errors change and disappear as you fill out the form. Also try to submit the form with and without errors to see the alert message.


Form Validation

localhost/lesson13/form_validation.html

Welcome to the Fight

Name	Brad	Please enter at least 5 characters.
Email	b@	Please enter a valid email address.
Password	••••••	Passwords Do Not Match
Confirm	••••	
Birth Date	Sometime	Please enter a valid date.

Melee
Mage
Ranged
Stealth

Select 2 class types

Right Handed Left Handed This field is required.

☐ Helmet	✓ Shield	
☐ Chainmail	☐ Plated	
☐ Gloves	☐ Boots	2 Pieces of Armor Required

Comments:

Engage

FIGURE 13.8 Simple registration form with validation.

LISTING 13.10 form_validation.html HTML Document That Implements the Registration Form Used in the Example

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Validation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/jquery.validate.min.js"></script>
    <script type="text/javascript" src="js/form_validation.js"></script>
    <link rel="stylesheet" type="text/css" href="css/form_validation.css">
  </head>
  <body>
    <div id="container"><p>Welcome to the Fight</p>
    <form action="test.html" method="get" name="test"
          id="simpleForm">
      <label>Name</label>
      <input name="name" type="text" /><br>
      <label>Email</label>
      <input name="email" type="text" /><br>
      <label>Password</label>
      <input name="password1" type="password" /><br>
      <label>Confirm</label>
      <input name="password2" id="password2" type="password" /><br>
      <label>Birth Date</label>
      <input name="birthDate" type="text" /><br>
      <select name="class" multiple size=4>
        <option>Melee</option>
        <option>Mage</option>
        <option>Ranged</option>
        <option>Stealth</option>
      </select><br>
      <div class="buttonGroup">
          <input type="radio" name="hands" value="right" />
          <label>Right Handed</label>
          <input type="radio" name="hands" value="left" />
          <label>Left Handed</label>
      </div><br>
      <div class="buttonGroup">
        <input type="checkbox" name="armor" value="helmet" />
        <label>Helmet</label>
        <input type="checkbox" name="armor" value="shield" />
        <label>Shield</label><br>
        <input type="checkbox" name="armor" value="chainmail" />
        <label>Chainmail</label>
        <input type="checkbox" name="armor" value="plated" />
        <label>Plated</label><br>
        <input type="checkbox" name="armor" value="gloves" />
        <label>Gloves</label>
        <input type="checkbox" name="armor" value="boots" />
        <label>Boots</label>
      </div><br>
      <label>Comments:</label><br><textarea></textarea><br>
      <input type="submit" value="Engage"/>
    </form>
    </div>
  </body>
</html>
```

LISTING 13.11 form_validation.js jQuery Code Implements the Validation of Form Elements

Click here to view code image

```
$(document).ready(function(){
  var validationObj = $("#simpleForm").validate({
    rules: {
      name: { required:true, minlength:5 },
      email: { required:true, email:true },
      password1: { required:true, rangelength:[6,12],
                   equalTo:"#password2" },
      birthDate: { required:true, date:true },
      class: { required:true, rangelength:[2,2]},
      hands: {required:true},
      armor: {required:true, minlength:2 }},
    messages: {
        password1: { equalTo:"Passwords Do Not Match"},
        class: { rangelength:"Select 2 class types"},
        armor: { minlength:"2 Pieces of Armor Required"},
      },
    errorPlacement: function(error, element){
      if (element.is(":radio")){
        error.insertAfter($("input:radio:last").next("label"))}
      else if (element.is(":checkbox")){
        error.insertAfter($("input:checkbox:last").next("label"))}
      else {error.insertAfter(element)}}
  });
  validationObj.form();
  $("form").submit(function(e){
    if(!validationObj.form()){
      alert("Form Errors");
      e.preventDefault(); } });
});
```

LISTING 13.12 form_validation.css CSS Code That Styles the Form Elements and Errors

Click here to view code image

```
label{display:inline-block;min-width:100px; text-align:right;}
```

```
input+label{text-align:left;}
input.error, select.error{
color:darkblue}
label.error{color:blue;}
br{clear:both;}
form{ margin:15px; margin:0px;
  border: 1px solid darkred;}
div{vertical-align:middle;
  padding-bottom:20px;
  font:italic 15px/30px Georgia, serif;}
#container{
    width:600px; height:auto;}
.buttonGroup { display: inline-block; padding: 5px;
  border-radius: 4px; margin:15px;
  box-shadow: 0px 0px 8px rgba(255, 0, 0, 0.5);}
p{
  color:white; background-color:darkred;
  font-weight:bold; margin:0px;
  text-align:center; border-radius: 4px 4px 0px 0px ; }
select, textarea, input{
  border-radius: 4px; margin:5px;
  border:2px groove crimson; padding:5px;}
select:focus, textarea:focus, input:focus {
  border-radius: 4px; margin:5px;
  border:2px groove #3377FF; }
```

Summary

In this lesson, you learned how to access the data that form elements represent. Using that data, you can provide dynamic workflow by setting values automatically, autofocusing the next element, showing new necessary elements, or hiding unnecessary ones.

You also learn how to apply some great visual effects to your elements, allowing you to guide users better. Some of the effects can be created in CSS; however, jQuery provides much more variations and control.

Validation of web forms is one of the most critical pieces of dynamic programming. You can save a lot of user time by catching problems right at the browser in jQuery rather than after the form is submitted to the server.

Q&A

Q. Is there a way to create my own validation rules with the jQuery Validation plug-in?

A. Yes. You can call the `.addMethod(name, function(value, element, parameters) [,message])` method on the `Validator` object returned by `.validate()`. The `name` you specify will be the new rule;

the function will be called, and the current value and element will be passed to it as the `value` parameter. The function should return `true` or `false` if the element value is valid. The option message parameter is the message that will be displayed.

Q. Isn't there a way to hide the ugly slide bar on a `<select>` element?

A. Not really. There are a bunch of workarounds, such as adding a `<div>`, that masks it. One thing you can do is create a `<div>` element the exact size of the element, and use CSS clipping to try to hide it. None of them is a perfect solution, though.

Workshop

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

Quiz

1. How do you access the value of a `<select>` element from jQuery?
2. How do you disable a form element from jQuery?
3. List two methods to tell if a check box input is checked from jQuery.
4. What does the following statement do?

```
$("input:text").blur();
```

Quiz Answers

1. You call the `.val()` method on the jQuery object representing it.
2. You call `.prop("disabled","disabled")` on the jQuery object representing it.
3. `.prop("checked")` or `.is(":selected")` on the jQuery object.
4. It removes focus from all `<input>` elements of type `text`.

Exercises

1. Open the code in Listings 13.1, 13.2, and 13.3 and modify the code so that `formB` also updates some of the elements in `formA`.
2. Open the code in Listings 13.4, 13.5, and 13.6 and modify the code so that when the user selects a new year, the month element is autofocused.
3. Open the code in Listings 13.10, 13.11, and 13.12 and add a new check box group for skills such as archery, strategy, hand to hand, or just make up your own. Add

validation that the users must select between two and four skills.

Lesson 14. Creating Advanced Web Page Elements in jQuery

What You'll Learn in This Lesson:

- ▶ Creating an image gallery with slider controls
- ▶ Adding sorting to table elements
- ▶ Adding filtering to table elements
- ▶ How to create a multitype tree view
- ▶ How to implement custom dialogs
- ▶ Creating visual elements using basic HTML and jQuery
- ▶ How to use jQuery to create dynamic sparklines

In this lesson, you have some fun creating some more advanced page elements. The purpose of this lesson is to give you a chance to apply everything that you have learned so far in this book in some more advanced ways. The elements you build in this lesson include an image gallery, interactive table, tree view, overlays, equalizer, and sparklines.

The following sections are written as examples that first explain some of the concepts used in the advanced elements; then you step through implementing them yourself. You will be able to apply the concepts learned in this lesson to implement a variety of types of advanced elements in your own web pages.

Adding an Image Gallery

One of the most visually interactive components of web pages are image galleries. An image is truly worth a thousand words. Images allow users to make decisions faster than any other type of visual element. The best way to apply images for allowing users to make decisions is to provide an interactive gallery.

An interactive gallery should enable users to quickly scan through several images and then easily select one to lead to the next step on the web page. Therefore, the necessary components are image thumbnails and controls. The thumbnails should be small versions of the original image that allow you to provide several options to select from at a time. Controls are whatever you apply to navigate through the thumbnails.

The most effective method of implementing an image gallery is to create a slider that contains the thumbnails with arrows that slide through the thumbnails until the user finds and clicks the desired image. Sliders are a good choice because they can be implemented in a vertical or horizontal fashion to fit the design of the web page.

To implement the slider, you create a container `<div>` element that has a fixed size and hidden overflow, then a child <div> element that contains all the thumbnail images. The position of the child <div> can then be adjusted to reveal different sets of the images in the slider. The following section describes that process.

Try it Yourself: Adding a Slider-Based Image Gallery

The purpose of the exercise is to give you some ideas about how to implement an image gallery with slider controls. The code for the example is in Listings 14.1, 14.2, and 14.3.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson14, lesson14/js, lesson14/css, and lesson14/images folders, and then add the lesson14/image_slider.html, lesson14/js/image_slider.js, and lesson14/css/image_slider.css files.

2. Add the code shown in Listing 14.1 and Listing 14.3 to the HTML and CSS files. The HTML code lays out a set of <div> elements that will contain the image slider, control buttons, and the displayed image. The CSS code styles the images and the <div> elements. Specifically, notice the following lines. These lines define the slide selector area with a specific width with hidden overflow, a child <div> area that contains all the image elements, and the look of the images themselves:

Click here to view code image

```
#selector {
    max-width:640px; height:140px;
    overflow-x:hidden; overflow-y:hidden;
  background-color:#DDDDDD; border:3px ridge grey; }
#imageSlide {
    position:relative; top:0px; left: 0px; height:100px; }
#imageSlide img {
  height:100px; opacity:.7; vertical-align:top;  margin:10px;
  border:3px ridge grey; box-shadow: 5px 5px 5px #888888; }
```

3. Open the image_slider.js file and add a basic `.ready()` function that you will use to implement all the handlers to handle the interactions.

4. Add the following lines to add `mouseenter` and `mouseleave` event handlers to the `#left` and `#right` elements to provide a hover effect:

Click here to view code image

```
  $("#left").mouseenter(function(){
    slide(50); });
  $("#left").mouseleave(function(){
    $("#imageSlide").stop(true); return false; });
  $("#right").mouseenter(function(){
    slide(-50); });
  $("#right").mouseleave(function(){
```

```
  $("#imageSlide").stop(true); return false; });
```

5. Add the following lines that will add a hover animation effect for the thumbnails in the `#imageSlide` element. The hover effect animates increasing the `opacity` and `size` for images that are hovered over to give the user an indication that the item is clickable and which one is selected. A click handler is also added so that when the user clicks a thumbnail, it sets the main image:

Click here to view code image

```
$("#imageSlide img").mouseenter(function(){
  $(this).stop(true).animate({height:120, opacity:1},500);
                             return false; });
$("#imageSlide img").mouseleave(function(){
  $(this).stop(true).animate({height:100, opacity:.5},500);
                             return false; });
$("#imageSlide img").click(setPhoto);
```

6. Add the following line that calls the `click()` handler on the first thumbnail image to initially select a main image:

Click here to view code image

```
$("#imageSlide img:first").click();
```

7. Add the following `setPhoto()` click handler that animates changing the main image. Notice that the code fades the image out and then back in. Also, because the aspect ratios of the images are not all the same, the aspect ratio of the thumbnail is used to determine whether to set the `height` or `width` property for the image to fit properly:

Click here to view code image

```
function setPhoto(){
  var newPhoto = $(this).attr("src");
  var horizontal = (minHorizontalRatio >
        $(this).height() / $(this).width());
  $("#photo img").stop(true).fadeTo(500, .1, "linear",
    function (){
      $("#photo img").attr("src", newPhoto); });
  if (horizontal) {
    $("#photo img").css({width:600,height:"auto"}) }
  else {
    $("#photo img").css({width:"auto",height:400}) }
  $("#photo img").fadeTo(500, 1);
  return false;
}
```

8. Add the following `slide()` code that handles the sliding of the thumbnail images. Basically, this code determines where the new `left` position of the

#imageSlide element should be. When the slide is all the way to the left, the left position is 0; when it is all the way to the right, the left position is set to a negative value clear off the web page so that only the right edge of the slide is visible:

Click here to view code image

```
function slide(value){
  var oldLeft = sliderLeft;
  sliderLeft = sliderLeft + value;
  if (sliderLeft >= 0) { sliderLeft = 0; }
  if (sliderLeft <= sliderMax) { sliderLeft = sliderMax; }
  if(oldLeft != sliderLeft) {
    $("#imageSlide").animate({left:sliderLeft}, 300, 'linear',
        function(){
          slide(value); });
  }
  return false;
}
```

9. Add the initialization function addImages() and the addImages() call shown in Listing 14.2. This code dynamically adds a set of image elements to the #imageSlide element.

10. Save all three files and then open the HTML document in a web browser, as shown in Figure 14.1. You should see the thumbnail slider and be able to navigate through it by hovering over the arrows. When you click a thumbnail image, the main image should change.

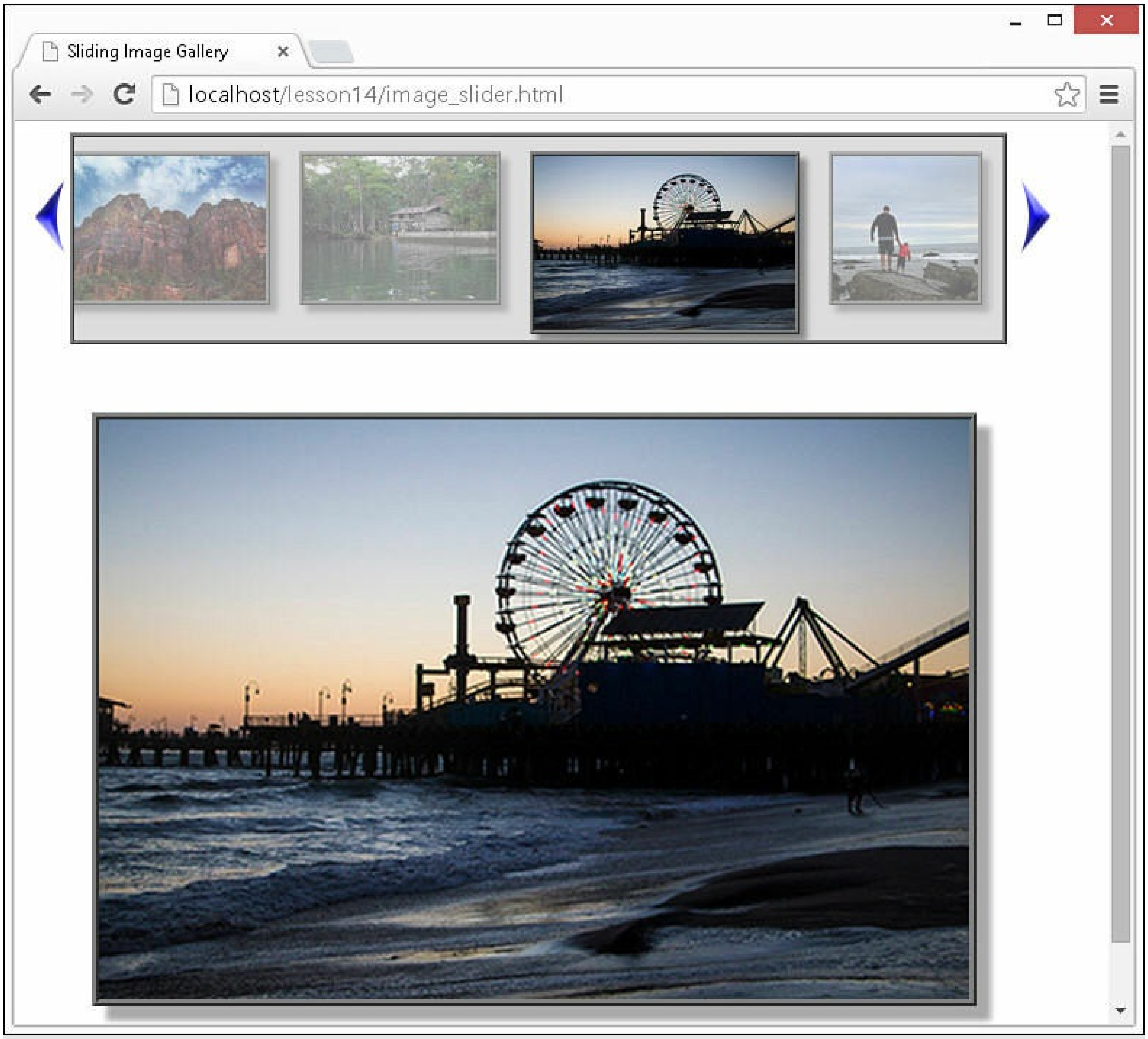


FIGURE 14.1 Image gallery with slider controls to select an image from thumbnails.

LISTING 14.1 image_slider.html HTML Document That Implements the Slider, Control, and Image Elements

[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sliding Image Gallery</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/image_slider.js"></script>
    <link rel="stylesheet" type="text/css" href="css/image_slider.css">
  </head>
  <body>
    <div id="viewer">
      <div id="left"><img src="/images/left.png" /></div>
      <div id="selector">
        <div id="imageSlide"></div>
      </div>
      <div id="right"><img src="/images/right.png" /></div><br>
      <div id="photo"><img src="" /></div>
    </div>
  </body>
</html>
```

LISTING 14.2 image_slider.js jQuery and JavaScript Code Implements the Mouse Event Handlers for the Image Slider Controls and Thumbnails

Click here to view code image

```
var sliderMax = sliderWidth = sliderLeft = 0;
var minHorizontalRatio = 400/600;
function addImages(){
  var images = ["img3", "img1", "volcano", "peak",
                "river", "wheel", "img7"];
  for (i in images){
    $("#imageSlide").append('<img src="/images/'+
                            images[i] + '.jpg" />'); }
  setTimeout(function() {
      $("#imageSlide img").each(function(){
        sliderWidth += $(this).width() + 26; });
      sliderWidth += 40;
      $("#imageSlide").width(sliderWidth);
      sliderMax = $("#selector").width() - sliderWidth;
    }, 1000);
}
function slide(value){
  var oldLeft = sliderLeft;
  sliderLeft = sliderLeft + value;
  if (sliderLeft >= 0) { sliderLeft = 0; }
  if (sliderLeft <= sliderMax) { sliderLeft = sliderMax; }
  if(oldLeft != sliderLeft) {
    $("#imageSlide").animate({left:sliderLeft}, 300, 'linear',
        function(){
          slide(value); });
  }
  return false;
}
function setPhoto(){
  var newPhoto = $(this).attr("src");
  var horizontal = (minHorizontalRatio >
        $(this).height() / $(this).width());
  $("#photo img").stop(true).fadeTo(500, .1, "linear",
    function (){
      $("#photo img").attr("src", newPhoto); });
  if (horizontal) {
    $("#photo img").css({width:600,height:"auto"}) }
  else {
    $("#photo img").css({width:"auto",height:400}) }
  $("#photo img").fadeTo(500, 1);
  return false;
}
$(document).ready(function(){
  addImages();
  $("#left").mouseenter(function(){
    slide(50); });
  $("#left").mouseleave(function(){
    $("#imageSlide").stop(true); return false; });
  $("#right").mouseenter(function(){
    slide(-50); });
  $("#right").mouseleave(function(){
    $("#imageSlide").stop(true); return false; });
  $("#imageSlide img").mouseenter(function(){
    $(this).stop(true).animate({height:120, opacity:1},500);
                              return false; });
  $("#imageSlide img").mouseleave(function(){
    $(this).stop(true).animate({height:100, opacity:.5},500);
                              return false; });
  $("#imageSlide img").click(setPhoto);
  $("#imageSlide img:first").click();
});
```

LISTING 14.3 image_slider.css CSS Code That Styles the Images and Controls

Click here to view code image

```
div {
    display:inline-block; }
#viewer2 {
    background-color:black; border:10px ridge blue; }
#right, #left {
    width:30px; height:100px; float:left; color:white; }
#right {
    float:right; }
#right img, #left img {
    margin-top:30px; width:32px; height:56px; }
#selector {
    max-width:640px; height:140px;
    overflow-x:hidden; overflow-y:hidden;
  background-color:#DDDDDD; border:3px ridge grey; }
#imageSlide {
    position:relative; top:0px; left: 0px; height:100px; }
#imageSlide img {
  height:100px; opacity:.7; vertical-align:top;  margin:10px;
  border:3px ridge grey; box-shadow: 5px 5px 5px #888888; }
#photo {
  height:500px; width:700px;
  display:table-cell; vertical-align:middle; text-align:center;
}
#photo img {
    border:5px ridge grey; box-shadow: 10px 10px 5px darkgrey; }
```

Implementing Tables with Sorting and Filters

Tables are basic HTML. You should already be familiar with the concepts of adding tables with rows that include one or more `<th>` or `<td>` elements that provide column values. Tables are a great way to present data that can be organized into columns or rows.

For simple amounts of data, a basic static HTML table is adequate. But, what do you do if there is a lot of data in the table? Some web pages scroll on and on with tabular data. They are a pain to navigate, and in the end it's easier to use the web browser search capability to try to find what you are looking for.

One solution to this is to implement sorting and filtering in server-side scripts via an AJAX request. Occasionally that is the best method when dealing with extremely large amounts of data stored in a database or extremely complex filtering algorithms. However, for most web pages, it is overkill and requires a bunch of slow server requests that make the web page interaction disjointed.

A much better approach is to add filtering and sorting to those large tables directly in jQuery and JavaScript. That allows the user to filter the data down to a specific set and then order by columns very quickly. The page flow is much smoother, and no external server requests are required. The following section takes you through the process of implementing column sorting and filtering to a table.

Try it Yourself: Creating an Interactive Table with Sorting and Filtering

The purpose of the exercise is to provide you with a chance to get and set form element values in a variety of ways. The code for the example is in Listings 14.4, 14.5, and 14.6.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson14/interactive_table.html, lesson14/js/interactive_table.js, and lesson14/css/interactive_table.css files.
2. Add the code shown in Listing 14.4 and Listing 14.6 to the HTML and CSS files. The HTML code defines the table and headers; the body will be added dynamically. The CSS code styles the table elements. Notice that the following

lines of CSS define background images that will be used to indicate that the column is sortable, sorted ascending or sorted descending. The state of the column can then be changed by switching classes to `.ascending`/`.descending`. A search icon also is added as a background image to the `<input>` element to indicate its purpose:

Click here to view code image

```
input {
    height:20px; width: 20px; border-radius:15px;
    padding: 0 7px 2px 25px;
  background:url("/images/search.png") no-repeat scroll left bottom 0 #FFFFFF;
  background-size:20px 20px;
}
input:focus {
    width:100px; }
span {
    background:url("/images/sort2.png") no-repeat scroll left;
  background-size:20px 20px;
  padding: 0 7px 2px 25px;
}
.ascending{
    background:url("/images/up2.png") no-repeat scroll left;
  background-size:20px 20px;
}
.descending{
    background:url("/images/down2.png") no-repeat scroll left;
  background-size:20px 20px;
}
```

3. Open the interactive_table.js file and the following `.ready()` function that will populate the table data, and add the sorting and filter handlers to the `<th>` elements. Notice that you use the jQuery `.data()` method to store the order and indicate whether the column data type is numeric. You also need to add the `randInt()` and `buildData()` functions shown Listing 14.5 that will populate the table data:

Click here to view code image

```
$(document).ready(function(){
  buildData();
  $("th").each(function(i) {
      var header = $(this);
      header.data({numeric:header.hasClass("numeric"), order:-1});
      header.children("span").click(function(){
        sortColumn(header, i); });
      var filter = $('<input type="text" />');
      filter.keyup(function(){
        filterColumn(filter, i); });
      header.append(filter);
    });
68 });
```

4. Add the following `keyup` event handler `filterColumn()` that will filter the column based on the text typed in the input for that column. The `input` and `column` arguments are passed to the handler in line 65. The `.each()` method is used to get the cell value of the specified column for each row. For numeric columns, if the value is less than the input value, the row is hidden. For string columns, the row is hidden if the string value of input is not found in the string value of the cell:

Click here to view code image

```
30 function filterColumn(input, column){
31   $("tbody tr").show().each(function(){
32     var header = $("th:eq("+ column +")");
33     var filterVal = input.val();
34     var rowVal = this.cells[column].innerHTML;
35     if(header.data("numeric") ){
36       if(parseFloat( filterVal ) > parseFloat( rowVal )) {
37         $(this).hide(); }}
38     else {
39       if(rowVal.indexOf(filterVal) < 0) { $(this).hide(); }}
40   });
41 }
```

5. Add the following `compare()` function that will accept two row elements, `a` and `b`, a `column` number, a `numeric` flag, and a sort `order` value. The code compares the column value of row `a` with row `b` and returns the correct sorting value based on whether the cell value is bigger, smaller, or equal:

Click here to view code image

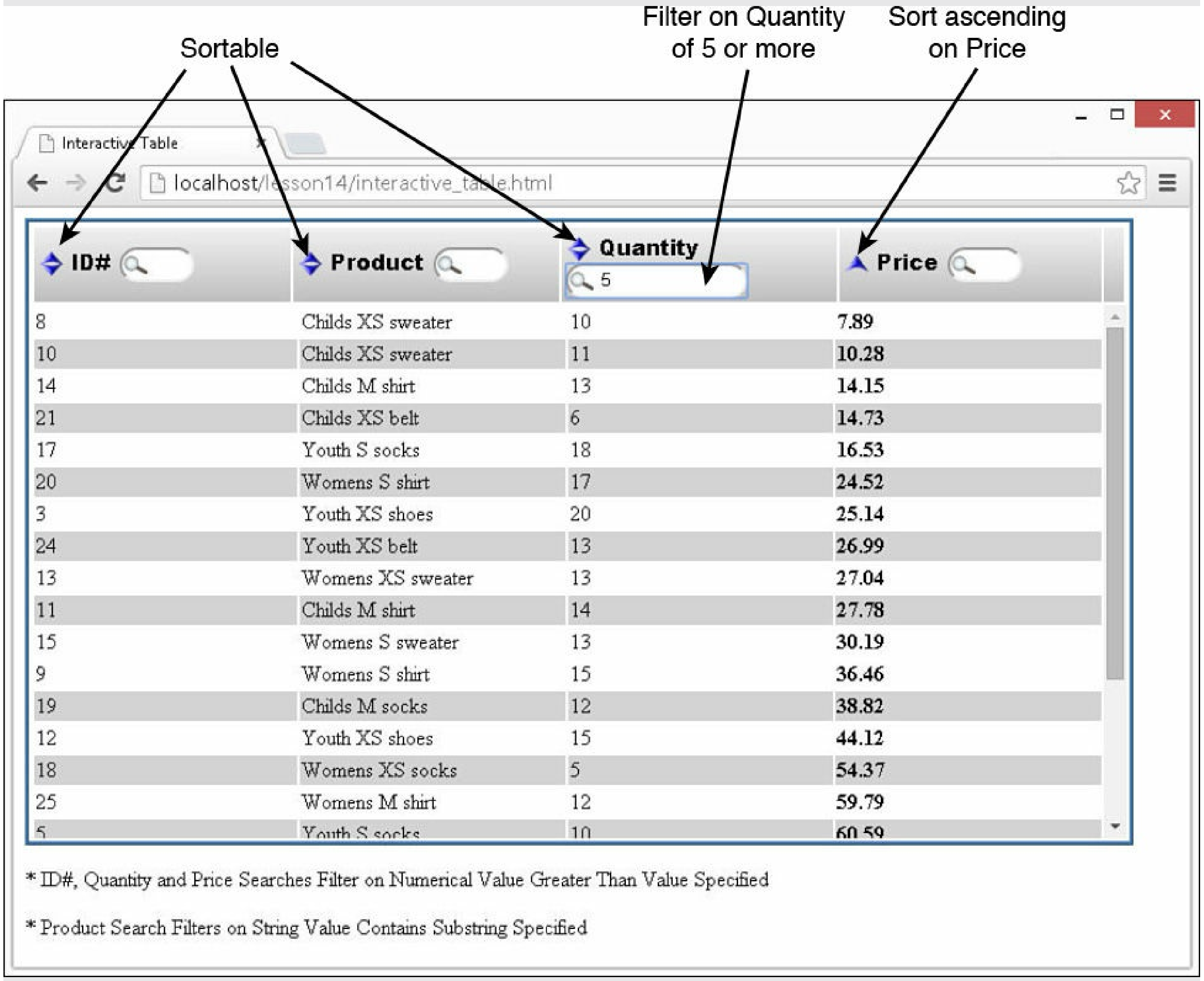
```
04 function compare(a, b, column, numeric, order){
05   var aValue = a.cells[column].innerHTML;
06   var bValue = b.cells[column].innerHTML;
07   if (numeric) { aValue = parseFloat(aValue );
08                  bValue = parseFloat(bValue ); }
09   if (aValue < bValue) return order;
10   if (aValue > bValue) return -order;
11   return 0;
12 }
```

6. Add the following `click` handler `sortColumn()` that will sort the column when the user clicks the column heading. The handler uses the `compare()` function to sort the table rows and then appends them back to the table in the correct order. The handler also sets the correct classes for the header and cell elements for the column:

Click here to view code image

```
13 function sortColumn(header, column){
 var rows = $("tbody tr");
 rows.sort(function(a, b){
   return compare(a, b, column, header.data("numeric"),
           header.data("order"));
  });
 $(rows).each(function(){ $("tbody").append($(this)); });
 header.data("order", -header.data("order"));
 $("span").removeClass("ascending descending");
 $("td").removeClass("sortColumn");
 if(header.data("order") > 0) {
  header.children("span").addClass("ascending"); }
 else {
  header.children("span").addClass("descending"); }
 $("tbody tr td:nth-child("+ (column + 1) +")")
  .addClass("sortColumn");
}
```

7. Save all three files and then open the HTML document in a web browser, as shown in Figure 14.2. You should be able to change the sort order for the columns and use the text inputs to filter the rows displayed.


Sortable
Filter on Quantity of 5 or more
Sort ascending on Price
Interactive Table
localhost/lesson14/interactive_table.html

ID#	Product	Quantity	Price
		5	
8	Childs XS sweater	10	7.89
10	Childs XS sweater	11	10.28
14	Childs M shirt	13	14.15
21	Childs XS belt	6	14.73
17	Youth S socks	18	16.53
20	Womens S shirt	17	24.52
3	Youth XS shoes	20	25.14
24	Youth XS belt	13	26.99
13	Womens XS sweater	13	27.04
11	Childs M shirt	14	27.78
15	Womens S sweater	13	30.19
9	Womens S shirt	15	36.46
19	Childs M socks	12	38.82
12	Youth XS shoes	15	44.12
18	Womens XS socks	5	54.37
25	Womens M shirt	12	59.79
5	Youth S socks	10	60.59

* ID#, Quantity and Price Searches Filter on Numerical Value Greater Than Value Specified

* Product Search Filters on String Value Contains Substring Specified

FIGURE 14.2 Dynamic table that allows you to sort and filter the elements displayed.

LISTING 14.4 interactive_table.html HTML Document That Implements the Table Elements Used in the Example

[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Interactive Table</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/interactive_table.js"></script>
    <link rel="stylesheet" type="text/css" href="css/interactive_table.css">
  </head>
  <body>
    <table>
      <thead><tr>
        <th class="numeric"><span>ID#</span></th>
        <th ><span>Product</span></th>
        <th class="numeric"><span>Quantity</span></th>
        <th class="numeric"><span>Price</span></th>
        <td class="spacer"></td>
      </tr></thead>
      <tbody></tbody>
    </table>
    <p>* ID#, Quantity and Price Searches Filter
       on Numerical Value Greater Than Value Specified</p>
    <p>* Product Search Filters on String Value Contains
       Substring Specified</p>
  </body>
</html>
```

LISTING 14.5 interactive_table.js jQuery and JavaScript Code Define the Interactions of the Table, Including Sorting and Filtering

[Click here to view code image]

```
var tArr = ["Mens", "Womens", "Youth", "Childs"];
var sArr = ["XL", "M", "S", "XS"];
var kArr = ["pants", "shirt", "shoes", "socks", "sweater", "belt"];
function compare(a, b, column, numeric, order){
  var aValue = a.cells[column].innerHTML;
  var bValue = b.cells[column].innerHTML;
  if (numeric) { aValue = parseFloat(aValue );
                 bValue = parseFloat(bValue ); }
  if (aValue < bValue) return order;
  if (aValue > bValue) return -order;
  return 0;
}
function sortColumn(header, column){
  var rows = $("tbody tr");
  rows.sort(function(a, b){
      return compare(a, b, column, header.data("numeric"),
                     header.data("order"));
    });
  $(rows).each(function(){ $("tbody").append($(this)); });
  header.data("order", -header.data("order"));
  $("span").removeClass("ascending descending");
  $("td").removeClass("sortColumn");
  if(header.data("order") > 0) {
    header.children("span").addClass("ascending"); }
  else {
    header.children("span").addClass("descending"); }
  $("tbody tr td:nth-child("+ (column + 1) +")")
    .addClass("sortColumn");
}
function filterColumn(input, column){
  $("tbody tr").show().each(function(){
    var header = $("th:eq("+ column +")");
    var filterVal = input.val();
    var rowVal = this.cells[column].innerHTML;
    if(header.data("numeric") ){
      if(parseFloat( filterVal ) >= parseFloat( rowVal )) {
        $(this).hide(); }}
    else {
      if(rowVal.indexOf(filterVal) < 0) { $(this).hide(); }}
  });
}
function randInt(max) {
  return Math.floor((Math.random()*max)+1); }
function buildData(){
  for(var x=1;x<26;x++){
      var row =$("<tr></tr>");
      row.append($("<td></td>").html(x));
      row.append($("<td></td>").html(
        tArr[randInt(3)] + " " + sArr[randInt(3)] +
        " " + kArr[randInt(5)]));
      row.append($("<td></td>").html(randInt(20)));
      row.append($("<td></td>")
          .html(((Math.random()*80)+5).toFixed(2)));
      $("tbody").append(row);}
}
$(document).ready(function(){
  buildData();
  $("th").each(function(i) {
      var header = $(this);
      header.data({numeric:header.hasClass("numeric"), order:-1});
      header.children("span").click(function(){
        sortColumn(header, i); });
      var filter = $('<input type="text" />');
      filter.keyup(function(){
        filterColumn(filter, i); });
      header.append(filter);
    });
});
```

LISTING 14.6 interactive_table.css CSS Code That Styles the Table Elements

[Click here to view code image]

```
table{
    border:3px ridge steelblue; padding:0px;}
thead {
    display:block; width:820px; text-align:left; }
tbody {
    display:block;  max-height:400px; width:820px;
    overflow-y:scroll; }
th {
    background-image: -moz-linear-gradient(top , #f1f1f1, #BFBFBF);
  background-image: -webkit-linear-gradient(top , #f1f1f1, #BFBFBF);
  background-image: -ms-linear-gradient(top , #f1f1f1, #BFBFBF);
  width:200px; height:30px; max-width: 200px;
  font:16px Arial Black; padding:3px;
}
.spacer {
    background-image: -moz-linear-gradient(top , #f1f1f1, #BFBFBF);
  background-image: -webkit-linear-gradient(top , #f1f1f1, #BFBFBF);
  background-image: -ms-linear-gradient(top , #f1f1f1, #BFBFBF);
  width:15px; border:none;
}
td {
    width:200px; border: .5px dotted; }
.sortColumn {
    font-weight:bold; }
tr:nth-child(even){
    background-color:lightgrey; }
img {
    height:26px; }
input {
  height:20px; width: 20px; border-radius:15px;
  padding: 0 7px 2px 25px;
  background:url("/images/search.png") no-repeat scroll left bottom 0
#FFFFFF;
  background-size:20px 20px;
}
input:focus {
    width:100px; }
span {
  background:url("/images/sort2.png") no-repeat scroll left;
  background-size:20px 20px;
  padding: 0 7px 2px 25px;
  cursor: pointer;
}
.ascending{
    background:url("/images/up2.png") no-repeat scroll left;
  background-size:20px 20px;
}
.descending{
    background:url("/images/down2.png") no-repeat scroll left;
  background-size:20px 20px;
}
```

Creating a Tree View

A tree view can be one of the most useful components when trying to present a large number of options to users. That is why tree views are used in so many places. Virtually all users will be familiar with them to at least a certain extent because it is the most common element used by OSes when displaying files and folders.

A tree view is simple to implement in jQuery and JavaScript. A very cool thing about tree views in your web pages is that they can contain any content that you want to display. The following section takes you through the process of implementing a tree view using jQuery and JavaScript.

Try it Yourself: Adding a Dynamic Tree View with Expanding and Collapsing Branches

The purpose of the exercise is to provide you with a chance to implement a tree view with items of different types. Each node in the tree is composed of a series of elements defined next:

Click here to view code image

```
<div class="tree">
  <span></span><label></label>
  <div class="content"></div>
  <div class="tree">one or more child elemnts</div>
</div>
```

The `<span>` element is used to display the collapsed or expanded image. The `<label>` element displays the text shown next to the Expand/Collapse button. The `<div class="content">` element can contain whatever content you want to include. You can also add additional `<div class="tree">` elements that are child nodes.

The code for the example is in Listings 14.7, 14.8, and 14.9. Use the following steps to create the dynamic web page:

1. In Eclipse, add the lesson14/dynamic_tree.html, lesson14/js/dynamic_tree.js, and lesson14/css/dynamic_tree.css files.
2. Add the code shown in Listing 14.7 and Listing 14.9 to the HTML and CSS files. The code in these files is fairly basic and should be familiar to you.
3. Open the dynamic_tree.js file and add the following `.ready()` function that calls `addLevels()`. Add the `addLevels()`, `getRandomItem()`, and `randInt()` functions shown in Listing 14.8. This code is used to populate the tree with random items, including images and form elements. This is to show you that the content of each node can be totally heterogeneous:

Click here to view code image

```
$(document).ready(function(){
  var root = addLevels($("#treeContainer"), 4);
  root.show();
});
```

4. Add the following `addItem()` function that is used to add a new node to the tree view. This function accepts jQuery object `parentItem` and `item`. The `item` argument is a JavaScript object with a label and content value. These are used to build up the full node and append it to the parent:

Click here to view code image

```
function addItem(parentItem, item){
  var newItem = $('<div class="tree"></div>').hide();
  newItem.append($("<span></span>").click(toggleItem));
  newItem.append($("<label></label>").html(item.label));
  newItem.append($('<div class="content"></div>')
      .append(item.content).hide());
  parentItem.append(newItem);
  return newItem;
}
```

5. Add the following `click` handler `toggleItem()` that toggles the visibility of the child `<div>` elements when the user clicks the expand/collapse button:

Click here to view code image

```
function toggleItem(){
  $(this).parent().children("div").toggle();
  $(this).toggleClass("collapse");
  return false;
}
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 14.3. You should be able to expand and collapse levels of the tree view.

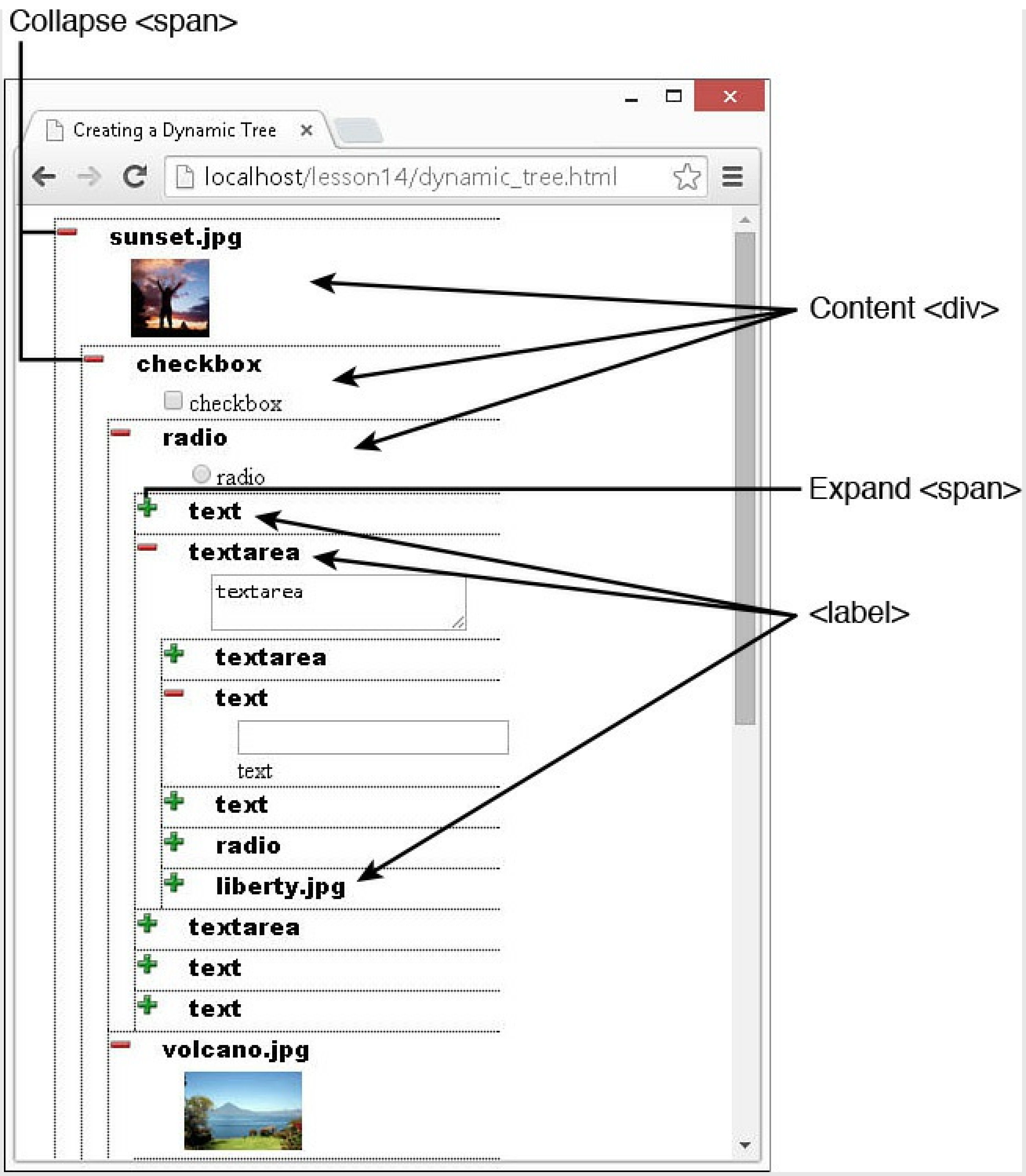


FIGURE 14.3 Collapsible tree view with multiple types of content.

LISTING 14.7 dynamic_tree.html HTML Document That Implements the Root Tree Element

Click here to view code image

```
<!DOCTYPE html>
```

```
<html>
  <head>
    <title>Creating a Dynamic Tree</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/dynamic_tree.js"></script>
    <link rel="stylesheet" type="text/css" href="css/dynamic_tree.css">
  </head>
  <body>
    <div id="treeContainer"></div>
  </body>
</html>
```

LISTING 14.8 dynamic_tree.js jQuery and JavaScript Code Populates and Controls the Expansion and Collapsing of the Tree

Click here to view code image

```
function toggleItem(){
  $(this).parent().children("div").toggle();
  $(this).toggleClass("collapse");
  return false;
}
function addItem(parentItem, item){
  var newItem = $('<div class="tree"></div>').hide();
  newItem.append($("<span></span>").click(toggleItem));
  newItem.append($("<label></label>").html(item.label));
  newItem.append($('<div class="content"></div>')
      .append(item.content).hide());
  parentItem.append(newItem);
  return newItem;
}
function randInt(max) {
  return Math.floor((Math.random()*max)); }
function getRandomItem(){
  var itemTypes=["image", "input", "textarea"];
  var images=["volcano.jpg", "liberty.jpg", "falls2.jpg",
              "wheel.jpg", "sunset.jpg"];
  var inputs=["text","checkbox", "radio"];
  switch(itemTypes[randInt(3)]){
    case "image":
      var img = images[randInt(5)];
      return { label:img,
               content:$('<img src="/images/' + img + '" />')};
    case "input":
      var type = inputs[randInt(3)];
      return { label:type,
               content:$('<input type="'+type+'">'+type+
                         '</input>')};
    case "textarea":
      return { label:"textarea",
               content:$('<textarea>textarea</textarea>')};
  }
}
function addLevels(parent, levels){
  var element = addItem(parent, getRandomItem());
  if( levels > 0 ){
    for(var x=0; x<5; x++){ addLevels( element, levels-1 ) }; }
  return element;
}
$(document).ready(function(){
  var root = addLevels($("#treeContainer"), 4);
  root.show();
});
```

LISTING 14.9 dynamic_tree.css CSS Code That Styles the Form Elements

[Click here to view code image]

```
#treeContainer {
    width:300px; }
.tree {
    margin-left: 16px; border-top:1px dotted;
    border-left:1px dotted;}
.content {
    margin-left: 48px; }
.tree span {
  display:inline-block; height:24px; width:24px;
  border-radius:8px; vertical-align:middle; margin-right:10px;
  background:url("/images/expand.png") no-repeat scroll 0px 0px/15px 15px;
}
.tree span.collapse {
  background:url("/images/collapse.png") no-repeat scroll 0px 0px/15px 15px;
}
img {
    height:50px; }
.tree label {
    font: 15px/20px "Arial Black";}
```

Using Overlay Dialogs

You have already learned how to add several types of dialogs to your web pages using JavaScript. The problem is that the built-in dialogs are extremely limited in what you can do with them. You could open a new pop-up window to get more control, but pop-up windows have their own problems, the biggest of which is that users hate them and usually disable them in their browsers.

A much better option is to create your own dialog element using jQuery and JavaScript. Implementing a dialog requires two things: an overlay element that will mask off everything that is happening behind on the web pages and a dialog component that provides the dialog interaction.

The overlay needs to have a higher `z-index` than the main web page elements below. The overlay should also be somewhat transparent so that the user can see the web page in the background but is unable to click any of the web page elements under the overlay.

The dialog needs to have a higher `z-index` than the overlay so that it is visible when the overlay is displayed. The dialog also needs to have a means of closing, which makes the dialog and overlay elements hidden. You can include anything else in the dialog that you desire to provide user interaction.

The dialog and overlay need to be in a fixed position and initially hidden. When you want to display the dialog, unhide the overlay and dialog and then hide them when you are finished displaying the dialog. The following section takes you through that process.

Try it Yourself: Adding Dynamic Dialogs Using Overlays in jQuery and JavaScript

The purpose of the exercise is to provide you with the capability to implement your own custom dialogs with an overlay that masks off the rest of the web page. The code for the example is in Listings 14.10, 14.11, and 14.12.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson14/dynamic_dialog.html, lesson14/js/dynamic_dialog.js, and lesson14/css/dynamic_dialog.css files.
2. Add the code shown in Listing 14.10 and Listing 14.12 to the HTML and CSS files. The HTML code implements a basic web page with two exceptions. Notice the `overlay` and `dialog <div>` elements. Those elements are not part of the main web page and will be used to simulate a pop-up dialog with only jQuery and JavaScript.
3. Open the dynamic_dialog.js file and add the following `.ready()` function that hides the overlay and dialog `<div>` elements, and then add a click handler to display them and another to update the web page and hide the dialog:

Click here to view code image

```
$(document).ready(function(){
  $("#overlay, #dialog").hide();
  $("span").click(function(){
    $("#overlay, #dialog").show(); });
  $("#updateB").click(update);
});
```

4. Add the following `click` handler `update()` that updates the values of the web page based on information obtained in the dialog `<div>`. This illustrates the interaction between the dialog and the rest of the web page:

Click here to view code image

```
function update(){
  $("#overlay, #dialog").hide();
  $("#title p").html($("#titleT").val());
  $("#content").html($("#contentT").val());
  $("#leftNav span").remove();
  $("input:checkbox").each(function(){
    if($(this).prop("checked")){
      $("#leftNav").append($("<span></span>")
          .html($(this).val()));
    }
  });
}
```

5. Save all three files and then open the HTML document in a web browser, as shown in Figure 14.4. You should be able to open the dialog by clicking the Update button. Update the form and then see the web page close when the Update button in the dialog is clicked.

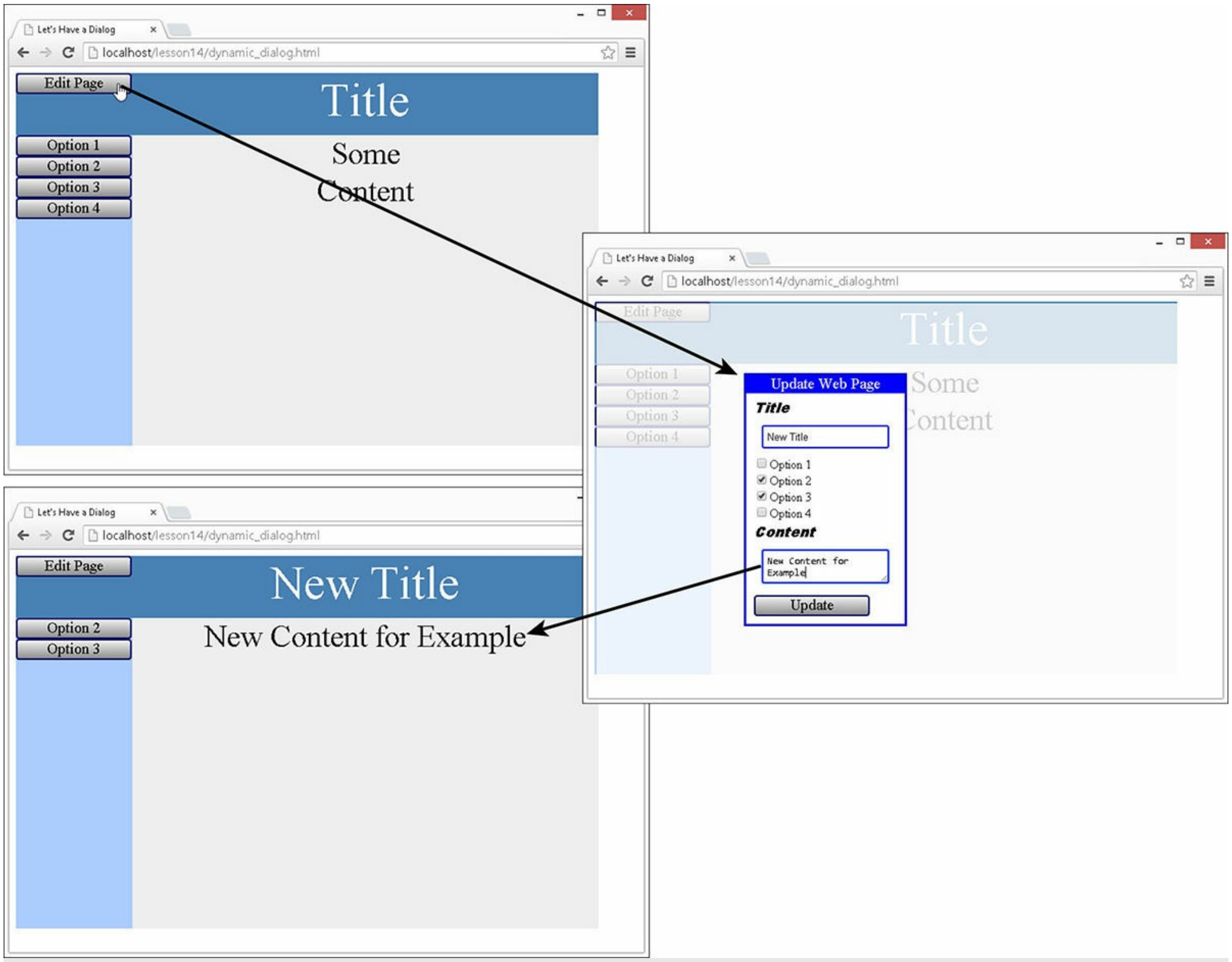


FIGURE 14.4 Form to page manipulation illustrating how to read data from forms and use it to update other elements on the web.

LISTING 14.10 dynamic_dialog.html HTML Document That Implements the Page, Overlay, and Dialog

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Let's Have a Dialog</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/dynamic_dialog.js"></script>
    <link rel="stylesheet" type="text/css" href="css/dynamic_dialog.css">
  </head>
  <body>
```

```
    <div><div id="title">
        <span id="edit">Edit Page</span><p>Title</p></div><br>
    <div>
      <div id="leftNav">
        <span>Option 1</span>
        <span>Option 2</span>
        <span>Option 3</span>
        <span>Option 4</span>
      </div>
      <div id="content">Some<br>Content</div>
    </div></div>
    <div id="overlay"></div>
    <div id="dialog">
      <p id="dialogTitle">Update Web Page</p>
      <label>Title</label><br>
      <input id="titleT" type="text" /><br>
      <input type="checkbox" value="Option 1" checked>Option 1</input>
<br>
      <input type="checkbox" value="Option 2" checked>Option 2</input>
<br>
      <input type="checkbox" value="Option 3" checked>Option 3</input>
<br>
      <input type="checkbox" value="Option 4" checked>Option 4</input>
<br>
      <label>Content</label><br>
      <textarea id="contentT"></textarea><br>
      <span id="updateB">Update</span>
    </div>
  </body>
</html>
```

LISTING 14.11 dynamic_dialog.js jQuery and JavaScript Code That Shows and Hides the Dialog and Updates the Web Page

Click here to view code image

```
function update(){
  $("#overlay, #dialog").hide();
  $("#title p").html($("#titleT").val());
  $("#content").html($("#contentT").val());
  $("#leftNav span").remove();
  $("input:checkbox").each(function(){
    if($(this).prop("checked")){
      $("#leftNav").append($("<span></span>")
          .html($(this).val()));
    }
  });
}
$(document).ready(function(){
  $("#overlay, #dialog").hide();
  $("#edit").click(function(){
    $("#overlay, #dialog").show(); });
  $("#updateB").click(update);
});
```

LISTING 14.12 dynamic_dialog.css CSS Code That Styles the Page, Overlay, and Dialog Elements

Click here to view code image

```
div {
    margin:0px; display:inline-block;
    float:left; text-align:center; }
span {
    background-image: -moz-linear-gradient(top , #f1f1f1, #8F8F8F);
  background-image: -webkit-linear-gradient(top , #f1f1f1, #8F8F8F);
  background-image: -ms-linear-gradient(top , #f1f1f1, #8F8F8F);
  color:black; border:2px ridge darkblue;
  font-size:20px; float:left; cursor:pointer;
  width:145px; text-align:center; border-radius: 4px; }
p {
    margin:0px; }
#title {
    background-color:steelblue; color:white;
    height:80px; width:750px; font-size:60px; }
#leftNav{
    width:150px; height:400px; font-size:20px;
    background-color:#AACCFF; }
#content{
    height:400px; width: 600px; font-size:40px;
    background-color:#EEEEEE; }
#overlay {
    position:fixed; top:10px; left:10px;
    height:480px; width:750px;
  opacity:.8; background-color:white; }
#dialog {
    border: 3px groove blue;  text-align:left;
    padding:10px; position:fixed;
    top:100px; left:200px; background-color:white; }
#dialogTitle {
    text-align:center; font:20px bold;
    background-color:blue; color:white;
    margin:-10px -10px 5px -10px; }
#contentT, #titleT {
    border-radius: 3px;  width:150px; padding:5px;
  margin:10px; border:2px groove blue; }
label {
    font:bold italic 18px "Arial Black" }
```

Implementing a Graphical Equalizer Display

The total purpose of this section is to help you see a method of using basic HTML elements along with dynamic jQuery and JavaScript interactions with data to provide a rich user experience with a visual indicator of what is happening with data. The data could be coming from a variety of sources, including JavaScript running on the web page or external services collected by AJAX requests.

A graphic equalizer element provides a great way to view several values at once and is something that many users are already familiar with. The following sections walk you through the process of implementing a graphical equalizer.

Try it Yourself: Creating a Dynamic Graphic Equalizer with Simple jQuery and CSS

The purpose of the exercise is to illustrate how to implement graphical elements using the basic web elements with CSS styling and some background jQuery. In this exercise, you create a bunch of `<span>` elements, use CSS to style them into an element representing a graphical equalizer display, and then add jQuery to dynamically update the display to provide a rich UI element. The code for the example is in Listings 14.13, 14.14, and 14.15.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson14/graphic_equalizer.html, lesson14/js/graphic_equalizer.js, and lesson14/css/graphic_equalizer.css files.
2. Add the code shown in Listing 14.13 and Listing 14.15 to the HTML and CSS files. These files should be basic for you by now. The CSS provides class styles for the different colors used in the equalizer and then styles the look of the elements.
3. Open the graphic_equalizer.js file and add the following `.ready()` function that dynamically builds the equalizer by adding `<div>` and `<span>` elements:

Click here to view code image

```
$(document).ready(function(){
  for(var i=0; i< 10; i++){
    $("#equalizer").append($("<div></div>"));
  }
  $("#equalizer div").each(function (idx){
    $(this).append($("<p></p>").html(idx));
    for(var i=0; i< 2; i++){
      $(this).append($('<span class="red"></span>')); }
    for(var i=0; i< 2; i++){
      $(this).append($('<span class="orange"></span>')); }
    for(var i=0; i< 3; i++){
      $(this).append($('<span class="yellow"></span>')); }
    for(var i=0; i< 8; i++){
      $(this).append($('<span class="green"></span>')); }
    adjValues();
  });
});
```

4. Add the following `updateEqualizer()` function that updates the opacity of the span elements so that only values below a certain level show as fully opaque. The data comes from a global array:

Click here to view code image

```
function updateEqualizer(){
  $("span").css({opacity:.3});
  $("#equalizer div").each(function(i){
    $(this).children("span:gt("+ (15 - valueArr[i]) +")")
      .css({opacity:1});
    $(this).children("p:first").html(valueArr[i]);
  });
}
```

5. Add the following functions that populate the global array every .5 seconds, via `setTimeout()`, with new data that is rendered in the equalizer by calling `updateEqualizer()`:

Click here to view code image

```
var valueArr = [10,8,3,12,12,15,15,3,4,5];
function randInt(max) {
  return Math.floor((Math.random()*max)-3); }
function adjValues(){
  for (var i=0; i<valueArr.length; i++) {
      var adj = valueArr[i] +
                Math.floor((Math.random()*7)-3);
      adj = Math.max(3, adj);
      adj = Math.min(15, adj);
      valueArr[i] = adj;
    }
  updateEqualizer();
  setTimeout(adjValues, 500);
}
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 14.5. You should see the graphical equalizer element updating automatically.

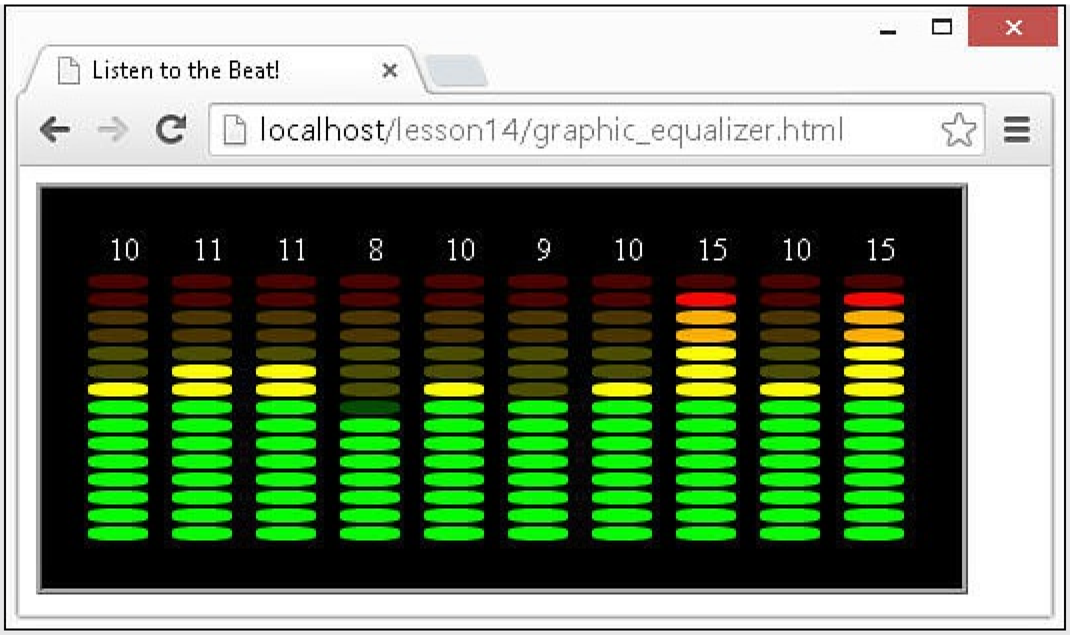


FIGURE 14.5 Graphical equalizer element.

LISTING 14.13 graphic_equalizer.html HTML Document That Implements a Web Page

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Listen to the Beat!</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/graphic_equalizer.js">
</script>
    <link rel="stylesheet" type="text/css"
href="css/graphic_equalizer.css">
  </head>
  <body>
    <div id="equalizer"></div>
  </body>
</html>
```

LISTING 14.14 graphic_equalizer.js jQuery and JavaScript Code Dynamically Build and Populate the Graphical Equalizer

Click here to view code image

```
var valueArr = [10,8,3,12,12,15,15,3,4,5];
function randInt(max) {
  return Math.floor((Math.random()*max)-3); }
function adjValues(){
  for (var i=0; i<valueArr.length; i++) {
      var adj = valueArr[i] +
                Math.floor((Math.random()*7)-3);
      adj = Math.max(3, adj);
      adj = Math.min(15, adj);
      valueArr[i] = adj;
    }
  updateEqualizer();
  setTimeout(adjValues, 500);
}
function updateEqualizer(){
  $("span").css({opacity:.3});
  $("#equalizer div").each(function(i){
    $(this).children("span:gt("+ (15 - valueArr[i]) +")")
      .css({opacity:1});
    $(this).children("p:first").html(valueArr[i]);
  });
}
$(document).ready(function(){
  for(var i=0; i< 10; i++){
    $("#equalizer").append($("<div></div>"));
  }
  $("#equalizer div").each(function (idx){
    $(this).append($("<p></p>").html(idx));
    for(var i=0; i< 2; i++){
      $(this).append($('<span class="red"></span>')); }
    for(var i=0; i< 2; i++){
      $(this).append($('<span class="orange"></span>')); }
    for(var i=0; i< 3; i++){
      $(this).append($('<span class="yellow"></span>')); }
    for(var i=0; i< 8; i++){
      $(this).append($('<span class="green"></span>')); }
    adjValues();
  });
});
```

LISTING 14.15 graphic_equalizer.css CSS code That Styles Elements to Render the Graphical Equalizer

Click here to view code image

```
#equalizer {
    background-color:black; color:white;
  width:420px; height:160px; padding:20px;
  border: 3px ridge darkgrey;
}
div{
    display:inline-block; width:40px;  padding:1px; }
p{
    text-align:center; margin:0px;}
span{
    display:block; width:30px; height:7px;
  margin:2px; border-radius: 40%; }
.green{
    background-color:#00FF00; }
.yellow{
    background-color:#FFFF00; }
.orange{
    background-color:#FFAA00; }
.red{
    background-color:#FF0000; }
```

Adding Sparkline Graphics

The purpose of this section is to help you see a method of using some of the new HTML5 elements along with dynamic jQuery and JavaScript interactions with data to provide a rich user experience with a visual indicator of trending data. The data could be coming from a variety of sources, including JavaScript running on the web page or external services collected by AJAX requests.

In this section, you implement a series of sparklines. A sparkline is a mini-graph that is updated frequently with the latest values from the web server. A sparkline element provides a great way for you to see how to use the new HTML elements to provide users with a great visual indicator of data trends. The following sections walk you through the process of implementing the sparkline.

Try it Yourself: Creating Dynamic Sparklines with Simple jQuery, JavaScript, and CSS

The purpose of the exercise is to illustrate how to implement new HTML5 graphical elements to render useful visual components to your web page. In this exercise, you use `<canvas>` elements to implement sparklines and dynamically update them using jQuery and JavaScript. The code for the example is in Listings 14.16, 14.17, and 14.18.

Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson14/dynamic_sparkline.html, lesson14/js/dynamic_sparkline.js, and lesson14/css/dynamic_sparkline.css files.
2. Add the code shown in Listing 14.16 and Listing 14.18 to the HTML and CSS files. The HTML and CSS are fairly basic. Notice that there are just `<div>`, `<span>`, `<label>`, and `<canvas>` elements.
3. Open the dynamic_sparkline.js file and add the following `.ready()`

function that populates the data used for the sparklines and starts off the `adjValues()` timer function. You also need to add the `adjValues()` and `getRandomArray()` functions shown in Listing 14.17. These functions populate and continuously update the values used for the sparklines. Notice that the data array is stored in the `<div>` element using the `.data()` method:

Click here to view code image

```
34 $(document).ready(function(){
35   $("div").each(function(){
36     $(this).data("valueArr", getRandomArray()); });
37   adjValues();
38 });
```

4. Add the following function `renderSpark()` that uses the set of values in the data array and draws a series of lines on the canvas to create the sparkline. Line 14 is a bit of a trick; setting the width of the canvas to its current value will erase the current data on the canvas, so the last sparkline is erased before drawing the new one:

Click here to view code image

```
13 function renderSpark(c, lineValues){
14   c.width = c.width;
15   var xAdj = c.width/lineValues.length;
16   var ctx = c.getContext("2d");
17   ctx.fillStyle = "#000000";
18   ctx.strokeStyle = "#00ffff";
19   ctx.lineWidth = 3;
20   var x = 1;
21   ctx.moveTo(x,(c.height));
22   for (var idx in lineValues){
23     var value = parseInt(lineValues[idx]);
24     ctx.lineTo(x+xAdj, (c.height - value));
25     x += xAdj;
26   }
27   ctx.stroke();
28 }
```

5. Save all three files and then open the HTML document in a web browser, as shown in Figure 14.6. You should see the sparklines automatically updating.

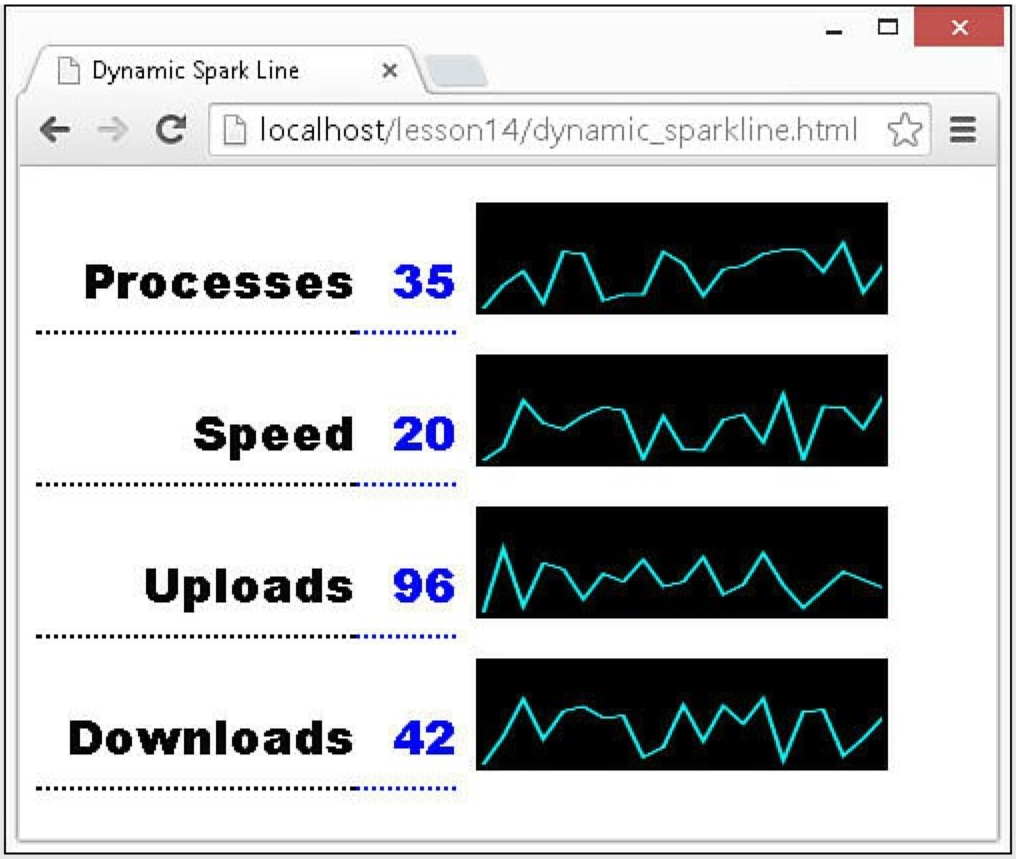


FIGURE 14.6 Web page with dynamic sparkline elements that update automatically.

LISTING 14.16 dynamic_spark.html HTML Document That Implements Page Elements

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dynamic Spark Line</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/dynamic_sparkline.js">
</script>
    <link rel="stylesheet" type="text/css"
href="css/dynamic_sparkline.css">
  </head>
  <body>
    <div>
      <label>Processes</label><span>1</span><canvas></canvas>
    </div>
    <div>
      <label>Speed</label><span></span><canvas></canvas>
    </div>
    <div>
```

```
      <label>Uploads</label><span></span><canvas></canvas>
    </div>
    <div>
      <label>Downloads</label><span></span><canvas></canvas>
    </div>
  </body>
</html>
```

LISTING 14.17 dynamic_spark.js jQuery and JavaScript Code Dynamically Populates and Updates the Sparklines

Click here to view code image

```
function randInt(max) {
  return Math.floor((Math.random()*max)+1); }
function adjValues(){
  $("div").each(function(){
    var lineValues = $(this).data("valueArr");
    lineValues.shift();
    lineValues.push(randInt(100));
    $(this).children("span").html(lineValues[0]);
    renderSpark($(this).children("canvas").get(0), lineValues);
  });
  setTimeout(adjValues, 1000);
}
function renderSpark(c, lineValues){
  c.width = c.width;
  var xAdj = c.width/lineValues.length;
  var ctx = c.getContext("2d");
  ctx.fillStyle = "#000000";
  ctx.strokeStyle = "#00ffff";
  ctx.lineWidth = 3;
  var x = 1;
  ctx.moveTo(x,(c.height));
  for (var idx in lineValues){
    var value = parseInt(lineValues[idx]);
    ctx.lineTo(x+xAdj, (c.height - value));
    x += xAdj;
  }
  ctx.stroke();
}
function getRandomArray(){
  var arr = new Array();
  for(var x=0; x<20; x++){ arr.push(randInt(100)); }
  return arr;
}
$(document).ready(function(){
  $("div").each(function(){
    $(this).data("valueArr", getRandomArray()); });
  adjValues();
});
```

LISTING 14.18 dynamic_spark.css CSS Code That Styles the Page Elements and Sparkline

Click here to view code image

```
canvas{
    height:50px; width: 200px; vertical-align:bottom;
  border:3px solid black; background-color:black; margin:10px; }
label, span {
    display:inline-block; text-align:right; width:160px;
  font:bold 24px/50px "Arial Black"; border-bottom:2px dotted; }
span{
    width:50px; color:blue; }
```

Summary

In this lesson, you implemented several more advanced web elements. You learned the basics of implementing image galleries, how to add sorting and filtering to a table, and how to dynamically create a tree view. You also implemented some graphical elements, such as sparklines and an equalizer display.

Q&A

Q. In your example, you populated the image slider using an array. Is that the best method?

A. Not necessarily. That was used for simplicity in the example. You can also use a static file located on the web server to get the list of files, or use an AJAX request to a server-side script that returns the list of images to display.

Q. Is it better to use a `<canvas>` or a `<svg>` element to draw chart type data such as the sparkline?

A. I used a `<canvas>` because it is simple to implement the changing lines. An `<svg>` chart would be too large to include in this book. The downside is that the canvas gets blurry if you zoom in on the web page. For the best charts, you should use `<svg>` so that the user can zoom in and it is still clean.

Workshop

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

Quiz

Quiz

1. How do you use jQuery to move the slider in the image gallery?
2. Why do you use an overlay element along with the dialog when creating custom dialogs?
3. How do you make the dialog box appear on top of other elements?

Quiz Answers

1. The slider is really just a big `<div>`. To adjust the position, change the relative position using the `.css()` or `.animate()` methods.
2. The overlay keeps the user from clicking on the rest of the web page until the dialog is closed.
3. Set the `z-index` to a higher value.

Exercises

1. Modify the code in Listings 14.4, 14.5, and 14.6 so that the filter for numerical columns will filter out items less than the value specified when the column is in ascending order and items greater than the value specified when the column is in descending order.
2. Modify the code in Listings 14.7, 14.8, and 14.9 to add expand all and collapse all buttons. The buttons should expand or collapse all elements in the tree.
3. This exercise is for more advanced CSS and HTML users. In the code in Listings 14.1, 14.2, and 14.3, modify the slider to be vertical. You will need to change positioning code to adjust the top instead of the left, move things around on the page, and do quite a number of CSS changes to get the slider to look right.

Lesson 15. Accessing Server-Side Data via JavaScript and jQuery AJAX Requests

What You'll Learn in This Lesson:

- Using AJAX requests to load data into page elements
- Sending AJAX GET and POST requests
- How to serialize parameters for GET and POST requests
- How to handle JSON and XML data in the web server AJAX response
- Implementing AJAX event handlers to handle completion, success, and failure events

In this lesson, you explore the world of asynchronous communication with the server using AJAX requests in jQuery and JavaScript. AJAX communications are one of the most vital parts of most websites. They allow jQuery and JavaScript to get additional data from the server and update the page instead of reloading or loading a new web page.

The following sections try to demystify AJAX a bit, and they provide you with some practical examples. By the end of this lesson, you will be able to implement AJAX in a variety of ways. AngularJS encapsulates the jQuery and JavaScript AJAX components into a very easy-to-use service called `$http` that you will learn about in the AngularJS section of lessons. It is not included here because you need to learn how AngularJS services work before you will be able utilize it.

Making AJAX Easy

Despite its importance, AJAX tends to be a bit daunting at first. With all the communication terms, it might seem easy to get confused. That really shouldn't be the case. If you take a quick step back to look at the basics of AJAX from a high level, it is easier to understand.

AJAX is a request from jQuery or JavaScript to the web server. The request may send data to the server, and the server will respond with a success or failure and possibly additional data. That is it—nothing more, nothing less. The following sections help clarify the request/response process.

Clarifying AJAX Versus Page Requests

The first step is to define the difference between AJAX and normal page linking request. You are already familiar with page links; when you click a link, a new web page appears. Often that is the case even if all the controls, tables, graphics, and so on are the same, but only some data has changed. In the case of form submission, none of the data changes, but the web page must still be reloaded from the server.

Caution

Don't confuse server-side dynamic creation of web pages with AJAX. Dynamic creation of web pages is still the old traditional method—it is just a bit easier to manage. Each request back to the server still requires a full new web page to be returned. The only advantage is that the web page is generated in memory instead of read from disk.

AJAX is completely different. An AJAX request does not request a new web page from the server. Instead, an AJAX request only sends and receives bits of data necessary. If the new data received from the web server requires the web page to be updated, then jQuery and JavaScript can update the page elements, as you have already seen.

The following is a list of a few of the benefits of AJAX requests:

- Less data requested from the web server.
- Allows the user to continue using the web page even while the request is being processed.
- Errors can be caught and handled on the client side before a request is ever made to the server.

Figure 15.1 illustrates the difference between the two methods of updating data in the browser.

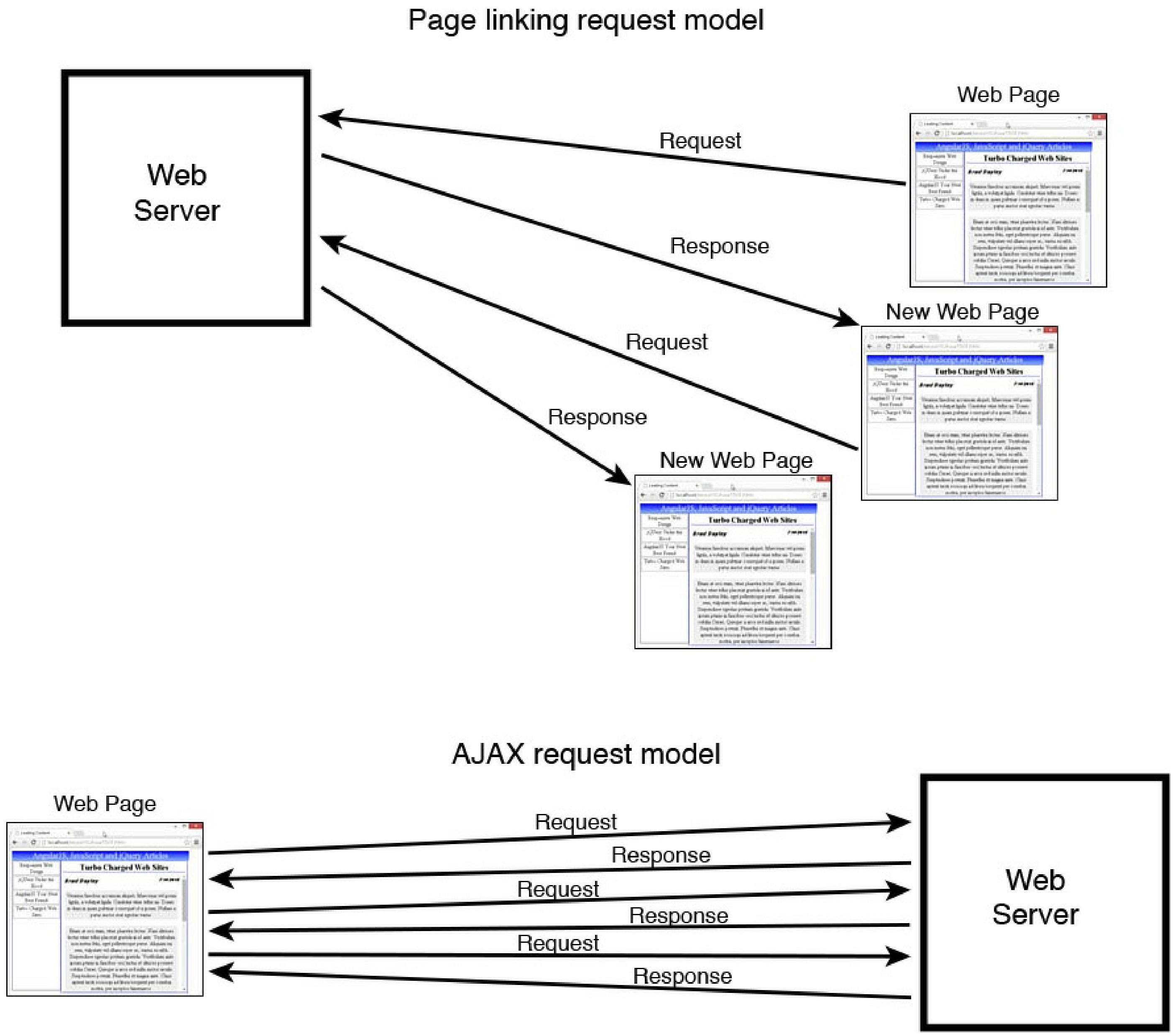


FIGURE 15.1 Comparison of AJAX requests versus traditional page linking.

Understanding Server-Side Services Such as Node.js, ASP, PHP, and MySQL

The next step is to understand how the AJAX requests are handled. Great detail is not necessarily important. What you need to understand is that for each AJAX request you send to a web server, a process will read the request, do some work, and then send back a response.

The back-end processes can be the web server returning a static HTML, text, XML, or JSON file. The back-end process often is an ASP, JSP, PHP, Python, or Node.js script that is running on the server reading data from memory, files, databases, and other sources. The back-end process can also be a myriad of other frameworks that are integrated into the web server.

None of that really matters to the AJAX script running in the browser. All that matters is that the server sends back a status and data.

Understanding Asynchronous Communication

You need to be clear on asynchronous communication. When you request a web page from the web server, the communication is synchronous to your browser. That means the browser waits until the HTML document is downloaded from the server before it begins rendering it and retrieving the additional resources necessary.

Tip

You can do synchronous AJAX requests with both jQuery and JavaScript. The jQuery `.ajax()` method and the JavaScript `send()` function both have a Boolean field that allows this. You shouldn't use this, though, because you can cause the browser to lock up, which will create some very unhappy users. In fact, in jQuery 1.8 and later, that option is deprecated.

AJAX communication is different when you send an AJAX request. Control is returned immediately back to jQuery or JavaScript, which can do additional things. Only when the request has completed or timed out will events be triggered in the browser that allow you to handle the request data.

Understanding Cross-Domain Requests

Cross-domain requests occur when you send AJAX requests to separate servers from different domains. The browser prevents this, and correctly so because of a multitude of security reasons.

The only problem with blocking cross-domain requests is that you often want to get data from services external to the current website. You can get around this in a couple of ways.

The first method to overcome the cross-domain restriction is to have the web server act as a proxy to the other servers or services, meaning that instead of directly communicating via JavaScript, you send the request to the server and have the server do it for you. The only downside is that it requires additional server-side scripting, and you pay an extra time penalty because data has to first be downloaded to the web server and then to the web browser.

Another option is to do what is called on-demand JavaScript, which is used by JSON with Padding (JSONP). This method takes advantage of the fact that you can download a script from another website as a resource using the following syntax:

Click here to view code image

```
<script type="text/javascript"
        src="http://new.domain.com/getData?jsonp=parseData">
</script>
```

The trick is that the address specified by the `src` attribute cannot return the JSON response directly. If the browser detects that the data in the script is JSON, as shown next, it throws a cross-domain error:

Click here to view code image

```
{ "title": "photoA", "rating": 7 };
```

Instead, the third-party service must support JSONP, in which case it pads the response with the function call specified by the request, as shown next. In that way, you will be able to access the data from JavaScript by calling the function:

Click here to view code image

```
parseData({ "title": "photoA", "rating": 7 });
```

Note

Another way to handle cross-domain communication that may be useful is to include the `ACCESS-CONTROL-ALLOW-ORIGIN:*` header in the response headers. This can easily be done if you are setting up your own server.

Looking at GET Versus POST Requests

This section clarifies GET and POST requests. The two main types of requests that you send to the web server are GET and POST. There are only a few differences between the two, but they are important. A GET request passes parameters as part of the URL, whereas a POST request passes them as part of the request data.

To help you quickly see this, the following illustrates the basic URL and data involved to send a first and last name to the server in a GET and POST request:

GET request URL:

Click here to view code image

```
http://localhost/code/example1.html?first=Brad&last=Dayley
```

Get request data:

```
<empty>
```

POST request URL:

Click here to view code image

```
http://localhost/code/example1.html
```

POST request data:

```
first=Brad
last=Dayley
```

Which request type should you use? The basic rule is to use a GET for retrieving information from the server and a POST if you are changing data on the server.

Understanding Response Data Types—Binary Versus Text Versus XML Versus JSON

It's important to be clear on the response data types that may come back from the server. The data that comes back to an AJAX request will be either in some sort of binary or text format. Typically, binary is reserved for images, zip files, and the like. However, text may be raw text, HTML, XML, JSON, and so on.

What does that mean? This is where it can get a bit tricky. When the browser receives a response from the server, it attempts to determine the type of data received by the server and create the appropriate response object for you to use.

For XML and HTML data, a DOM object is created, whereas for JSON, a JavaScript object is created. For raw text or binary objects, an object is constructed that provides access to the raw text or binary data. You will see how this works in the following sections.

Implementing AJAX

Now that you have your head wrapped around the AJAX concepts, you are ready to begin implementing the basic details. To implement AJAX requests, you need to access the HTTP request object, format the data that needs to be sent to the server, send the request, and then handle the response.

Both JavaScript and jQuery have methods to send an AJAX request to the web server. You'll see both methods in the following sections. jQuery is able to do everything that JavaScript can do, but in a much easier and extensible manner.

The following sections discuss and provide some examples of implementing basic AJAX requests.

AJAX from JavaScript

To implement an AJAX request in JavaScript, you need access to a new `window.XMLHttpRequest` object. Table 15.1 describes the important methods and attributes of the `XMLHttpRequest` object.

Method/Attribute	Description
`open()`	Allows you to construct a GET or POST request.
`send()`	Sends the request to the server.
`onreadystatechange`	Event handler that will be executed when the state of the `XMLHttpRequest` object changes.
`response`	Object created by the browser based on the data type.
`responseText`	Raw text returned in the response data from the server.
`setRequestHeader`	This allows you to set HTTP request headers necessary to implement the request.
`status`	Status response code from server; that is, 200 for success, 404 for not found, and so on.

TABLE 15.1 Important Methods and Attributes of the XMLHttpRequest Object

To illustrate this, check out the following code that sends a GET AJAX request to the server to get the email address based on first and last name parameters:

Click here to view code image

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    { document.getElementById("email").innerHTML=xmlhttp.responseText;}
  }
xmlhttp.open("GET","/getUserEmail?userid=brad",true);
xmlhttp.send();
```

The following code illustrates how to send a basic POST AJAX request to the server to set the email address. Notice that for the POST request, the added `Content-length` and `Content-type` headers make sure that the data is treated correctly at the server. The `Content-length` is set to the length of the `params` string:

Click here to view code image

```
var xmlhttp = new XMLHttpRequest();
var params = "first=Brad&last=Dayley&email=brad@dayleycreations.com";
http.setRequestHeader("Content-type", "text/plain");
http.setRequestHeader("Content-length", params.length);
xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    { alert("Email Updated");
  }
xmlhttp.open("POST","/setUserEmail",true);
xmlhttp.send(params);
```

Note

In Internet Explorer browsers earlier than IE7, you need to use the following line to create the `xmlhttp` object because the window object doesn't support the XMLHttpRequest attribute:

Click here to view code image

```
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
```

AJAX from jQuery

This section looks at implementing AJAX requests from jQuery. jQuery provides very simple-to-use wrapper methods to implement AJAX requests, `.load()`, `.get()`, and `.post()`.

The helper function are wrappers around the `.ajax()` interface, which is discussed later. These helper functions are the following:

- **.load(url [, data] [, success(data, textStatus, jqXHR)])**—This method is used to load data directly elements represented by the jQuery object.
- **.getScript (url [, data] [, success(data, textStatus, jqXHR)]))**—This method is used to load and then immediately execute a JavaScript/jQuery script.
- **.getJSON(url [, data] [, success(data, textStatus, jqXHR)]))**—This method is used to load data in JSON format using a JSONP request to servers on different domains.
- **.get(url [, data] [, success(data, textStatus, jqXHR)] [, dataType]))**—This method is used to send a generic GET request to the server.
- **.post(url [, data] [, success(data, textStatus, jqXHR)] [, dataType]))**—This method is used to send a generic POST request to the server.

Each of these methods enables you to specify the `url` of the request. The `.load()` method is used to load data directly into elements represented by jQuery object. The `.get()` and `.post()` methods are used to send GET and POST requests.

The `data` argument can be a string or basic JavaScript object. For example, in the following example, `obj`, `objString`, and `formString` are all valid data arguments:

Click here to view code image

```
var obj ={"first":"Brad", "last":"Dayley"};
var objString = $.param(obj);
var formString = $("form").serialize();
```

You can also specify the function that executes when the response from the server succeeds. For example, the following success handler sets the value of the `#email` element to the value response data:

Click here to view code image

```
$.get("/getEmail?first=Brad&last=Dayley", null, function (data, status,
xObj){
  $("#email").html(data);
}));
```

The `.get()` and `.post()` methods also enable you to specify a `dataType` parameter as a string, such as “`xml"`, `"json"`, `"script"` , `"html"`, and so on, that formats the expected format of the response data from the server. For example, to specify a response type of JSON, you use the following:

Click here to view code image

```
$.get("/getUser?first=Brad&last=Dayley", null, function (data, status,
xObj){
  $("#email").html(data.email);
}), "json");
```

Try it Yourself: Sending an AJAX Request from jQuery

In this exercise, you get a chance to implement some simple AJAX requests using the jQuery `.load()` method. The resulting web page contains a left navigation that uses AJAX requests to load lorem ipsum article data from the web server and populate the web page with the results. The purpose of the exercise is to give you a chance to see how the `.load()` method works.

The code for the example is in Listings 15.1, 15.2, 15.3, and 15.4. Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson15, lesson15/js, and lesson15/css folders, and then add the lesson15/load_content.html, lesson15/js/load_content.js, and lesson15/css/load_content.css. You also need to create the lesson15/data folder and the files article1.html, article2.html, article3.html, and article4.html inside it. You can also download these files from the book’s website.

2. Listing 15.4 shows an example of the data format for the article#.html files.

3. Add the code shown in Listing 15.1 and Listing 15.3 to the HTML and CSS files. The code in these files should be familiar to you, with the exception of an `article` attribute to the `<span>` element, as shown next. This is used in the jQuery to identify which article should be loaded when the span is clicked:

Click here to view code image

```
        <span class="navItem"
```

```
                 article="article1">Responsive Web Design</span>
```

4. Now open the load_content.js file and add the following `.ready()` that adds a click handler `setArticle()` to the left navigation items on the web page:

Click here to view code image

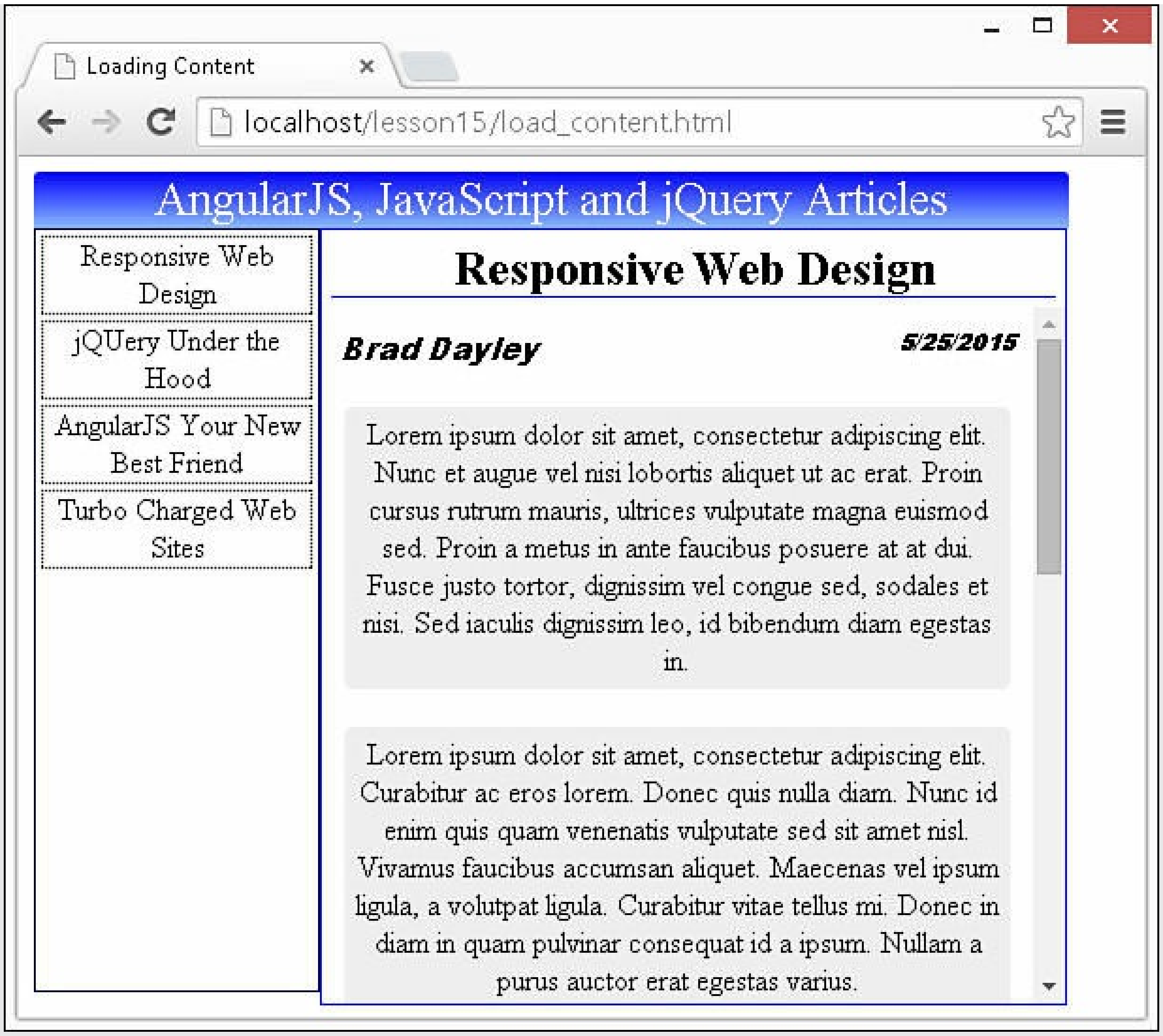
```
$(document).ready(function(){
  $(".navItem").click(setArticle);
});
```

5. Add the following click handler `setArticle()` that calls the jQuery AJAX method `.load()` and populates the `#content <div>` with new data loaded from the server. Notice that the `article` attribute in the span is read and a .html extension added so that the `.load()` method requests a different article for each link:

Click here to view code image

```
function setArticle(){
  $("#content").load("data/"+$(this).attr("article")+".html");
}
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 15.2. You should be able to click the left navigation items and load different articles.



AngularJS, JavaScript and jQuery Articles

Responsive Web Design

jQUery Under the Hood

AngularJS Your New Best Friend

Turbo Charged Web Sites

Responsive Web Design

Brad Dayley

5/25/2015

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc et augue vel nisi lobortis aliquet ut ac erat. Proin cursus rutrum mauris, ultrices vulputate magna euismod sed. Proin a metus in ante faucibus posuere at at dui. Fusce justo tortor, dignissim vel congue sed, sodales et nisi. Sed iaculis dignissim leo, id bibendum diam egestas in.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur ac eros lorem. Donec quis nulla diam. Nunc id enim quis quam venenatis vulputate sed sit amet nisl. Vivamus faucibus accumsan aliquet. Maecenas vel ipsum ligula, a volutpat ligula. Curabitur vitae tellus mi. Donec in diam in quam pulvinar consequat id a ipsum. Nullam a purus auctor erat egestas varius.

FIGURE 15.2 Article viewer that uses AJAX requests to populate the article content.

LISTING 15.1 load_content.html HTML Document That Adds Menu and Content

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Loading Content</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/load_content.js"></script>
    <link rel="stylesheet" type="text/css" href="css/load_content.css">
  </head>
  <body>
    <div id="banner">AngularJS, JavaScript and jQuery Articles</div>
    <br>
    <div>
      <div id="leftNav">
        <span class="navItem"
              article="article1">Responsive Web Design</span>
        <span class="navItem"
              article="article2">jQuery Under the Hood</span>
        <span class="navItem"
              article="article3">AngularJS Your New Best Friend</span>
        <span class="navItem"
              article="article4">Turbo Charged Web Sites</span>
      </div>
      <div id="content">
        <span id="title">&nbsp;</span>
        <div id="article"></div>
      </div>
    </div>
  </body>
</html>
```

LISTING 15.2 load_content.js jQuery and JavaScript That Implements the AJAX .load() Requests

Click here to view code image

```
function setArticle(){
  $("#content").load("data/"+$(this).attr("article")+".html");
}
$(document).ready(function(){
  $(".navItem").click(setArticle);
});
```

LISTING 15.3 load_content.css CSS Code That Styles the Page

Click here to view code image

```
div { margin:0px; display:inline-block; float:left; text-align:center;
}
p { margin:2px; }
#banner { border-radius: 3px 3px 0px 0px;
  background-image: -moz-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -webkit-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -ms-linear-gradient(top , #0000FF, #88BBFF);
  color:white; height:30px; width:550px; font-size:25px; }
#leftNav { width:150px; height:404px; border:1px groove #000088; }
.navItem { border:1px dotted; display:block; margin:3px; }
.navItem:hover { border:1px solid; background-color:#00FF00;
cursor:pointer; }
#content {border:1px solid blue;}
#article { width: 375px; height:350px; padding:10px; overflow-y:scroll;
}
#title { font-weight:bold; font-size:25px; border-bottom: 1px blue
solid;
  display:block; margin:5px; color:black; }
#by { text-align:right; font:bold italic 16px arial black; float:left;
  margin-bottom:20px; }
#date { text-align:right; font:italic 12px arial black; float:right;}
#article p {margin-top:20px; background-color:#EEEEEE; border-
radius:5px;
  clear:both; padding:5px; }
```

LISTING 15.4 article1.html Article HTML Code That Is Dynamically Loaded

Click here to view code image

```
<span id=title>Responsive Web Design</span>
<div id="article">
<span id="by">Brad Dayley</span>
<span id="date">5/25/2015</span>
<p>Lorem ipsum dolor ... </p>
<p>Lorem ipsum dolor sit amet, ...</p>
<p>...</p>
<p>...</p>
<p>....</p>
</div>
```

Handling AJAX Responses

In addition to specifying the `success` handler, the wrapper methods also enable you to attach additional handlers using the following methods:

- **.done(data, textStatus, jqXHR)**—Called when a successful response is received from the server.
- **.fail(data, textStatus, jqXHR)**—Called when a failure response is received from the server or the request times out.
- **.always(data, textStatus, jqXHR)**—Always called when a response is received from the server.

For example, the following code adds an event handler that is called when the request fails:

Click here to view code image

```
$.get("/getUser?first=Brad&last=Dayley", null, function (data, status,
xObj){
  $("#email").html(data.email);
}), "json").fail(function(data, status, xObj){
  alert("Request Failed");
});
```

Try it Yourself: Handling Request Success and Failures

In this exercise, you get a chance to implement the `.done()`, `.fail()`, and `.always()` AJAX event handlers on a basic jQuery `.get()` request. The resulting web page sends the name and color to a server-side Node.js script that checks for `name="Lancalot"` and `color="blue"`. If the correct name and color are entered, the request succeeds; otherwise, the request fails. The purpose of the exercise is to apply the AJAX handlers to a practical concept.

The code for the example is in Listings 15.5, 15.6, 15.7, and 15.8. Use the following steps to create the dynamic web page:

1. In Eclipse, create lesson15/ajax_response.html, lesson15/js/ajax_response.js, and lesson15/css/ajax_response.css files.
2. Create a file in the root of the project named server_lesson15_ajax_handling.js and place the contents of Listing 15.8 into it. This file will act as the Node.js webserver for this example only.
3. Stop the server.js webserver if it is already running for other examples. Then start the server_lesson15_ajax_handling.js webserver using the node command or Run As, Node Application if you are using Eclipse.

Note

Don't forget when you are done to stop the server_lesson15_ajax_handling.js server and start the normal server.js before moving on to additional exercises.

4. Add the code shown in Listing 15.5 and Listing 15.7 to the HTML and CSS files. The HTML and CSS code define a basic login dialog.
5. Open the ajax_response.js file and add the following `.ready()` function that will add a click handler to the input button:

Click here to view code image

```
$(document).ready(function(){
  $("#requestButton").click(askYourQuestions);
});
```

6. Add the following click handler `askYourQuestions()` that will be called when the user clicks the request button. The handler makes an AJAX `.get()` request to the /request path on the webserver. Notice that the data from the `#requestForm` is serialized using `.serialize()` to create the

query string. Also notice that `.done()`, `.fail()`, and `.always()` are attached to the `.get()` request to handle the AJAX completion events:

Click here to view code image

```
function askYourQuestions(){
  $.get("/request",
   $("#requestForm").serialize()).done(success).fail(failure).alway
  return false;
}
```

7. Add the following three AJAX event handlers to handle completion, failure, and successful login attempts:

Click here to view code image

```
function failure(){ alert("You May Not Pass!"); }
function success(){ alert("You May Pass!"); }
function always(){ alert("Questions Answered."); }
```

8. Save all three files and then open the HTML document in a web browser, as shown in Figure 15.3. You should see the completion alert pop up whenever you click the request button. You should also see the success alert when you use the correct credentials of `name="Lancalot"` and `color="blue"`.

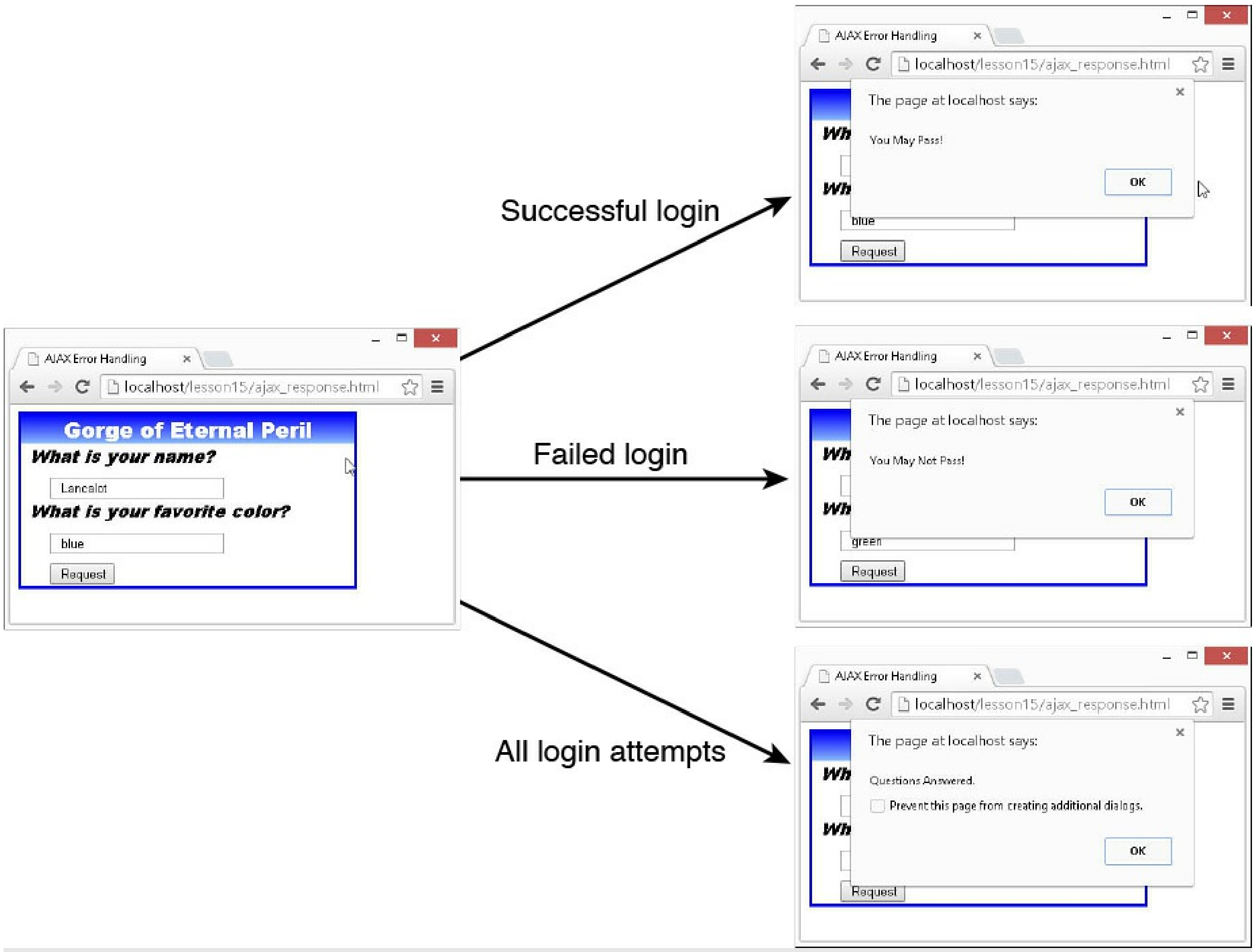


FIGURE 15.3 Simple form dialog.

LISTING 15.5 ajax_response.html HTML Document That Creates the Form Dialog

[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>AJAX Error Handling</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/ajax_response.js"></script>
    <link rel="stylesheet" type="text/css"
href="css/ajax_response.css">
  </head>
  <body>
    <div id="request">
      <div id="title">Gorge of Eternal Peril</div>
      <form id="requestForm">
        <label>What is your name? </label>
        <input type="text" name="name" /><br>
        <label>What is your favorite color? </label>
        <input type="text" name="color" /><br>
        <input id="requestButton" type="button" value="Request" />
      </form>
    </div>
  </body>
</html>
```

LISTING 15.6 ajax_response.js jQuery and JavaScript That Sends the Form Request to the Server via an AJAX GET Request and Handles Success and Failure Conditions

[Click here to view code image]

```
function failure(){ alert("You May Not Pass!"); }
function success(){ alert("You May Pass!"); }
function always(){ alert("Questions Answered."); }
function askYourQuestions(){
  $.get("/request",
   $("#requestForm").serialize()).done(success).fail(failure).always(alw
  return false;
}
$(document).ready(function(){
  $("#requestButton").click(askYourQuestions);
});
```

LISTING 15.7 ajax_response.css CSS Code That Styles the Page

[Click here to view code image]

```
#request {
  height:180px; width:350px; border: 3px ridge blue; }
#title {
    text-align:center;
  background-image: -moz-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -webkit-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -ms-linear-gradient(top , #0000FF, #88BBFF);
  height:30px; color:white;  font:bold 22px arial black; }
input {
    margin-top:10px;
  margin-left:30px;
  padding-left:10px; }
label {
    margin-left:10px;
    font:italic 18px arial black; }
```

LISTING 15.8 server_lesson15_ajax_handling.js Node.js Server That Will Handle the POST and GET Requests for This Exercise

Click here to view code image

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use('/', express.static('./'));
app.get('/request', function(req, res){
  var queryObj = req.query;
  if (queryObj.name == "Lancalot" && queryObj.color == "blue"){
    res.send(200, "Welcome To AJAX!");
  } else {
    res.send(400, "Invalid Answers!");
  }
});
app.listen(80);
```

Handling Response Data

You have already learned about the different data types that can be generated by the response. The four main types that you will be working with are script, text, JSON, and XML/HTML. The script and text are handled simply by the `.load()` and `.getScript()` methods. JSON and XML/HTML can be a bit more complex.

The following sections walk you through the process of handling JSON and XML data in the response from the server.

Try it Yourself: Handling JSON Response Data

JSON data is by far the easiest to work with in jQuery AJAX responses. This is because the response data is in object form, so you can access it via dot naming. For example, the following JSON response from the server:

Click here to view code image

```
{"first":"Brad", "last":"Dayley"}
```

Can be accessed in the response data as the following:

Click here to view code image

```
var name  = data.first + " " + data.last;
```

Even if the response data object comes as a string, you can use the `.parseJSON()` to get a JavaScript object. For example:

Click here to view code image

```
var data = $.parseJSON('{"first":"Brad", "last":"Dayley"}');
var name  = data.first + " " + data.last;
```

In this exercise, you handle JSON data coming back from an AJAX request. The resulting web page contains several images that have captions. The image caption and the image filename come from a JSON file located on the server at lesson15/data/images.json. The purpose of the exercise is to familiarize you with using JSON data returned from an AJAX request to dynamically populate a page.

The code for the example is in Listings 15.9, 15.10, 15.11, and 15.12. Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson15/load_json.html, lesson15/js/load_json.js, lesson15/css/load_json.css, and lesson15/data/images.json files.
2. Add the code shown in Listing 15.9 and Listing 15.12 to the HTML and CSS files. These are very basic files because the image elements will be dynamically generated.
3. Add the contents of Listing 15.11 to the images.json file.
4. Open the load_json.js file and add the following `.ready()` function that makes a `.get()` request to the server to get the lesson15/data/images.json file and calls the `updateImages()` AJAX request complete event handler. The contents of the JSON file can be seen in Listing 15.11:

Click here to view code image

```
$(document).ready(function(){
  $.get("data/images.json", updateImages);
});
```

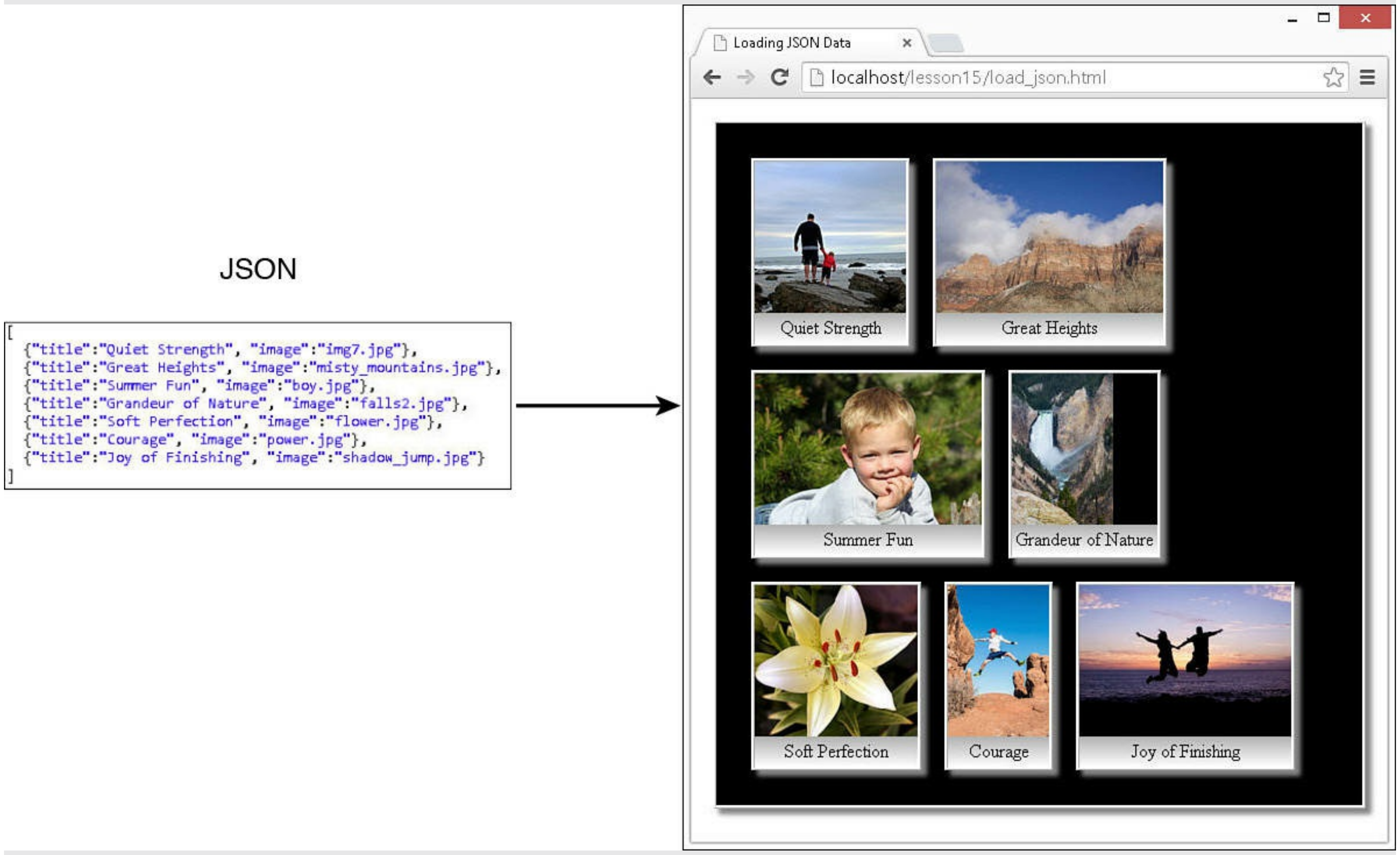
5. Add the following AJAX request complete event handler `updateImages()`. Notice that the JSON response `data` has been converted to a JavaScript object array that you are able to iterate through and create the image elements and add them to the web page:

Click here to view code image

```
function updateImages(data){
  for (i=0; i<data.length; i++){
    var imageInfo =data[i];
    var img = $('<img />').attr("src", "images/"+imageInfo.image);
    var title = $("<p></p>").html(imageInfo.title);
    var div = $("<div></div>").append(img, title);
    $("#images").append(div);
  }
}
```

6. Save all three files and then open the HTML document in a web browser, as

shown in Figure 15.4. You should see the images loaded with the captions from the JSON file.



```
[
  {"title":"Quiet Strength", "image":"img7.jpg"},
  {"title":"Great Heights", "image":"misty_mountains.jpg"},
  {"title":"Summer Fun", "image":"boy.jpg"},
  {"title":"Grandeur of Nature", "image":"falls2.jpg"},
  {"title":"Soft Perfection", "image":"flower.jpg"},
  {"title":"Courage", "image":"power.jpg"},
  {"title":"Joy of Finishing", "image":"shadow_jump.jpg"}
]
```

FIGURE 15.4 Image gallery populated with JSON data.

LISTING 15.9 load_json.html HTML Document That Loads the jQuery and JavaScript

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Loading JSON Data</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/load_json.js"></script>
    <link rel="stylesheet" type="text/css" href="css/load_json.css">
  </head>
  <body>
    <div id="images"></div>
  </body>
</html>
```

LISTING 15.10 load_json.js jQuery and JavaScript Code That Implements the AJAX Request and Handles the JSON Response

Click here to view code image

```
function updateImages(data){
  for (i=0; i<data.length; i++){
    var imageInfo =data[i];
    var img = $('<img />').attr("src", "/images/"+imageInfo.image);
    var title = $("<p></p>").html(imageInfo.title);
    var div = $("<div></div>").append(img, title);
    $("#images").append(div);
  }
}
$(document).ready(function(){
  $.get("data/images.json", updateImages);
});
```

LISTING 15.11 images.json JSON Data from the Book Website at lesson15/data/images.json Containing Image Filenames and Captions

Click here to view code image

```
[
  {"title":"Quiet Strength", "image":"img7.jpg"},
  {"title":"Great Heights", "image":"misty_mountains.jpg"},
  {"title":"Summer Fun", "image":"boy.jpg"},
  {"title":"Grandeur of Nature", "image":"falls2.jpg"},
  {"title":"Soft Perfection", "image":"flower.jpg"},
  {"title":"Courage", "image":"power.jpg"},
  {"title":"Joy of Finishing", "image":"shadow_jump.jpg"}
]
```

LISTING 15.12 load_json.css CSS Code That Styles the Images

Click here to view code image

```
div {border:3px ridge white; box-shadow: 5px 5px 5px #888888;
  display:inline-block; margin:10px; }
p { background-image: -moz-linear-gradient(top , #B1B1B1, #FFFFFF);
  background-image: -webkit-linear-gradient(top , #B1B1B1, #FFFFFF);
  background-image: -ms-linear-gradient(top , #B1B1B1, #FFFFFF);
  margin:0px; padding:3px; text-align:center; }
img { height:130px; vertical-align:top;  }
#images { background-color:black; padding:20px; }
```

Try it Yourself: Handling XML/HTML Response Data

XML/HTML data is not as easy as JSON, but jQuery does make it fairly easy to work with. XML data in the response is returned as a DOM object, which can be converted to jQuery and searched/navigated using jQuery's extensive options. For example, the following XML response from the server:

Click here to view code image

```
<person><first>Brad</first><last>Dayley</last></person>
```

Can be accessed in the response data as the following:

Click here to view code image

```
var name  = $(data).find("first").text() + " " +
$(data).find("last").text();
```

Similar to JSON, if the response data object comes as a string, you can use the `.parseXML()` to get a DOM object. For example:

Click here to view code image

```
var data = $.parseXML("<person><first>Brad</first><last>Dayley</last>
</person>");
var name  = $(data).find("first").text() + " " +
$(data).find("last").text();
```

In this exercise, you get a chance to handle XML data coming back from an AJAX request. The resulting web page contains a basic table with cell data derived from the XML data contained in the file lesson15/data/parkdata.xml located on the server. The purpose of the exercise is to familiarize you with using XML data returned from an AJAX request to dynamically populate a page.

The code for the example is in Listings 15.13, 15.14, 15.15, and 15.16. Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson15/load_xml.html, lesson15/js/load_xml.js, lesson15/css/load_xml.css, and lesson15/data/parkdata.xml files.
2. Add the code shown in Listing 15.13 and Listing 15.16 to the HTML and CSS files. These are very basic files because the table body will be dynamically generated.
3. Add the contents of Listing 15.15 to the parkdata.xml file and save it.
4. Open the load_xml.js file and add the following `.ready()` function that will make a `.get()` request to the server to get the lesson15/data/parkdata.xml file and call the `updateTable()` AJAX request complete event handler. The contents of the XML file can be seen in Listing 15.15:

Click here to view code image

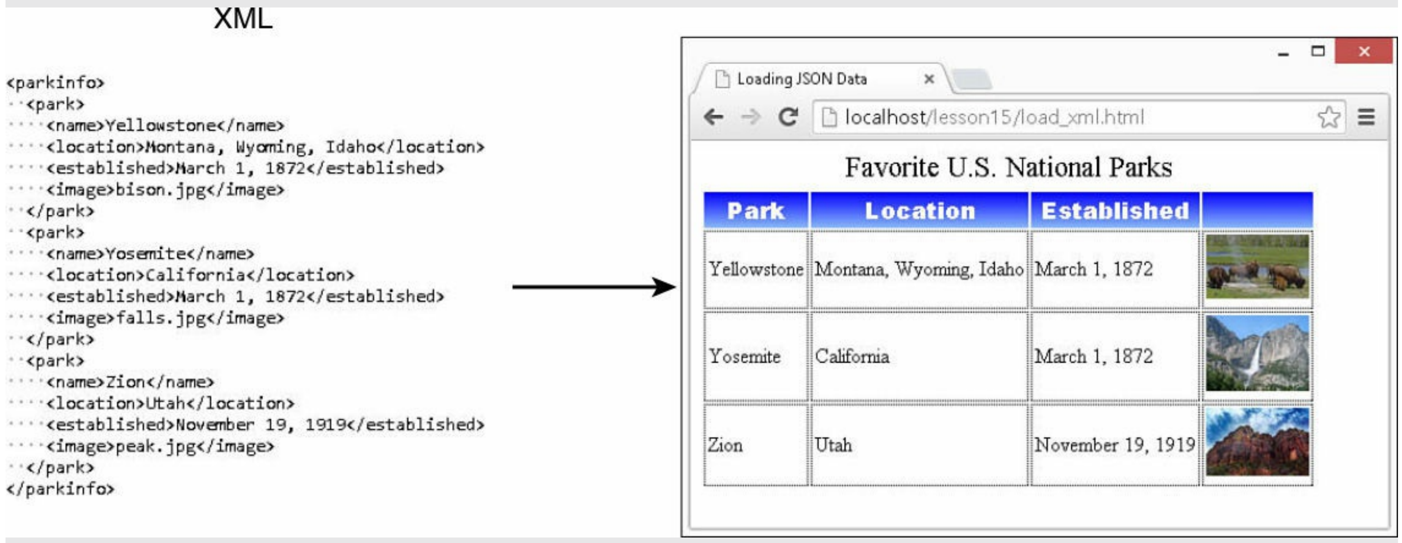
```
$(document).ready(function(){
  $.get("data/parkdata.xml", updateTable);
});
```

5. Add the following AJAX request complete event handler `updateTable()`. Notice that the XML response `data` has been converted to a DOM element that is converted to a jQuery object using `$(data)`. The jQuery object can then be iterated using `.each()`, and each element can be searched using the `.children()` method to get the different values in the XML data:

Click here to view code image

```
function updateTable(data){
  var parks = $(data).find("park");
  parks.each(function(){
    var tr = $("<tr></tr>");
    tr.append($("<td>
</td>").html($(this).children("name").text()));
    tr.append($("<td>
</td>").html($(this).children("location").text()));
    tr.append($("<td>
</td>").html($(this).children("established").text()));
    var img = $('<img />').attr("src",
"images/"+$(this).children("image").text());
    tr.append($("<td></td>").append(img));
    $("tbody").append(tr);
  });
}
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 15.5. You should see the table populated from the XML file.



```
<parkinfo>
  <park>
    <name>Yellowstone</name>
    <location>Montana, Wyoming, Idaho</location>
    <established>March 1, 1872</established>
    <image>bison.jpg</image>
  </park>
  <park>
    <name>Yosemite</name>
    <location>California</location>
    <established>March 1, 1872</established>
    <image>falls.jpg</image>
  </park>
  <park>
    <name>Zion</name>
    <location>Utah</location>
    <established>November 19, 1919</established>
    <image>peak.jpg</image>
  </park>
</parkinfo>
```

Favorite U.S. National Parks

Park	Location	Established	
Yellowstone	Montana, Wyoming, Idaho	March 1, 1872	
Yosemite	California	March 1, 1872	
Zion	Utah	November 19, 1919	

FIGURE 15.5 Table populated with XML data.

LISTING 15.13 load_xml.html HTML Document That Loads the jQuery and JavaScript

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Loading XML Data</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/load_xml.js"></script>
    <link rel="stylesheet" type="text/css" href="css/load_xml.css">
  </head>
  <body>
    <table>
      <caption>Favorite U.S. National Parks</caption>
      <thead><th>Park</th><th>Location</th><th>Established</th>
<th>&nbsp;</th></thead>
      <tbody></tbody>
    </table>
    <p></p>
  </body>
</html>
```

LISTING 15.14 load_xml.js jQuery and JavaScript Code That Implements the AJAX Request and Handles the XML Response

Click here to view code image

```
function updateTable(data){
  var parks = $(data).find("park");
  parks.each(function(){
    var tr = $("<tr></tr>");
    tr.append($("<td></td>").html($(this).children("name").text()));
    tr.append($("<td>
</td>").html($(this).children("location").text()));
    tr.append($("<td>
</td>").html($(this).children("established").text()));
    var img = $('<img />').attr("src",
"images/"+$(this).children("image").text());
    tr.append($("<td></td>").append(img));
    $("tbody").append(tr);
  });
}
$(document).ready(function(){
  $.get("data/parkdata.xml", updateTable);
});
```

LISTING 15.15 parkdata.xml XML Data File with Raw Table Data

Click here to view code image

```
<parkinfo>
  <park>
    <name>Yellowstone</name>
    <location>Montana, Wyoming, Idaho</location>
    <established>March 1, 1872</established>
    <image>bison.jpg</image>
  </park>
  <park>
    <name>Yosemite</name>
    <location>California</location>
    <established>March 1, 1872</established>
    <image>falls.jpg</image>
  </park>
  <park>
    <name>Zion</name>
    <location>Utah</location>
    <established>November 19, 1919</established>
    <image>peak.jpg</image>
  </park>
</parkinfo>
```

LISTING 15.16 load_xml.css CSS Code That Styles the Table

Click here to view code image

```
img {width:80px;}
th {
  background-image: -moz-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -webkit-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -ms-linear-gradient(top , #0000FF, #88BBFF);
  color:white;  font:bold 18px arial black; }
caption { border-radius: 10px 10px 0px 0px; font-size:22px;
height:30px; }
td { border:1px dotted; padding:2px; }
```

Try it Yourself: Updating Server Data from jQuery Using AJAX

In this exercise, you get a chance to implement a more complex AJAX web page with `.get()` and `.post()` requests, as well as some different AJAX event handlers. The resulting web page provides links to different vacation spots that you can rate. The data for the vacations is located in a data structure on the Node.js webserver. When you change the rating, a POST request is sent to the Node.js server and the server data is changed, making it permanent as long as the server is up. The purpose of the exercise is to solidify the jQuery AJAX concepts.

The code for the example is in Listings 15.17, 15.18, 15.19, and 15.20. Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson15/ajax_post.html, lesson15/js/ajax_post.js, and lesson15/css/ajax_post.css files.

2. Create a file in the root of the project named server_lesson15_ajax_post.js and place the contents of Listings 15.5, 15.6, 15.7, and 15.8 into it. This file will act as the Node.js webserver for this example only.

3. Stop the server.js webserver if it is already running for other examples. Then start the server_lesson15_ajax_post.js webserver using the node command or Run As, Node Application if you are using Eclipse.

Note

Don’t forget when you are done to stop the server_lesson15_ajax_post.js server and start the normal server.js before moving on to additional exercises.

---

4. Add the code shown in Listing 15.17 and Listing 15.19 to the HTML and CSS files. These files define the style and framework for the vacations page. There shouldn’t be anything new in these files that you haven’t already seen.

5. Open the ajax_post.js file and add the following `.ready()` function that will make a `.get()` request to the server to get the list of vacations from the server. Notice that a `.done()` function is added so that when the data has been returned and the links are populated, the first link is automatically clicked to set the vacation content. Notice that the `sendRating()` event handler is added to the start elements to handle rating changes via the mouse:

Click here to view code image

```
$(document).ready(function(){
  $.get("/getList", setList).done(function(){
      $("span:first").click(); return false; });
  $(".star").click(sendRating);
});
```

6. Add the following click handler `setList()` for the left navigation buttons. For each of the buttons, the `getTrip()` event handler is added:

Click here to view code image

```
function setList(data){
  var items = [];
  $.each(data, function(key, val) {
```

```
17     var item = $("<span></span>").html(val.title);
18     item.click(function(){getTrip(val.idx)});
19     $("#leftNav").append(item);
20   });
21 }
```

7. Add the following `getTrip()` event handler that will send a `.get()` AJAX request to get a specific trip's info by specifying /getTrip as the path and `idx:idx`, which is the index to the trip on the webserver. The `setTrip()` handler is called when the AJAX request is complete, so you need to add code shown in Listing 15.18 lines 1–10. This function takes the JSON data and populates the content elements:

Click here to view code image

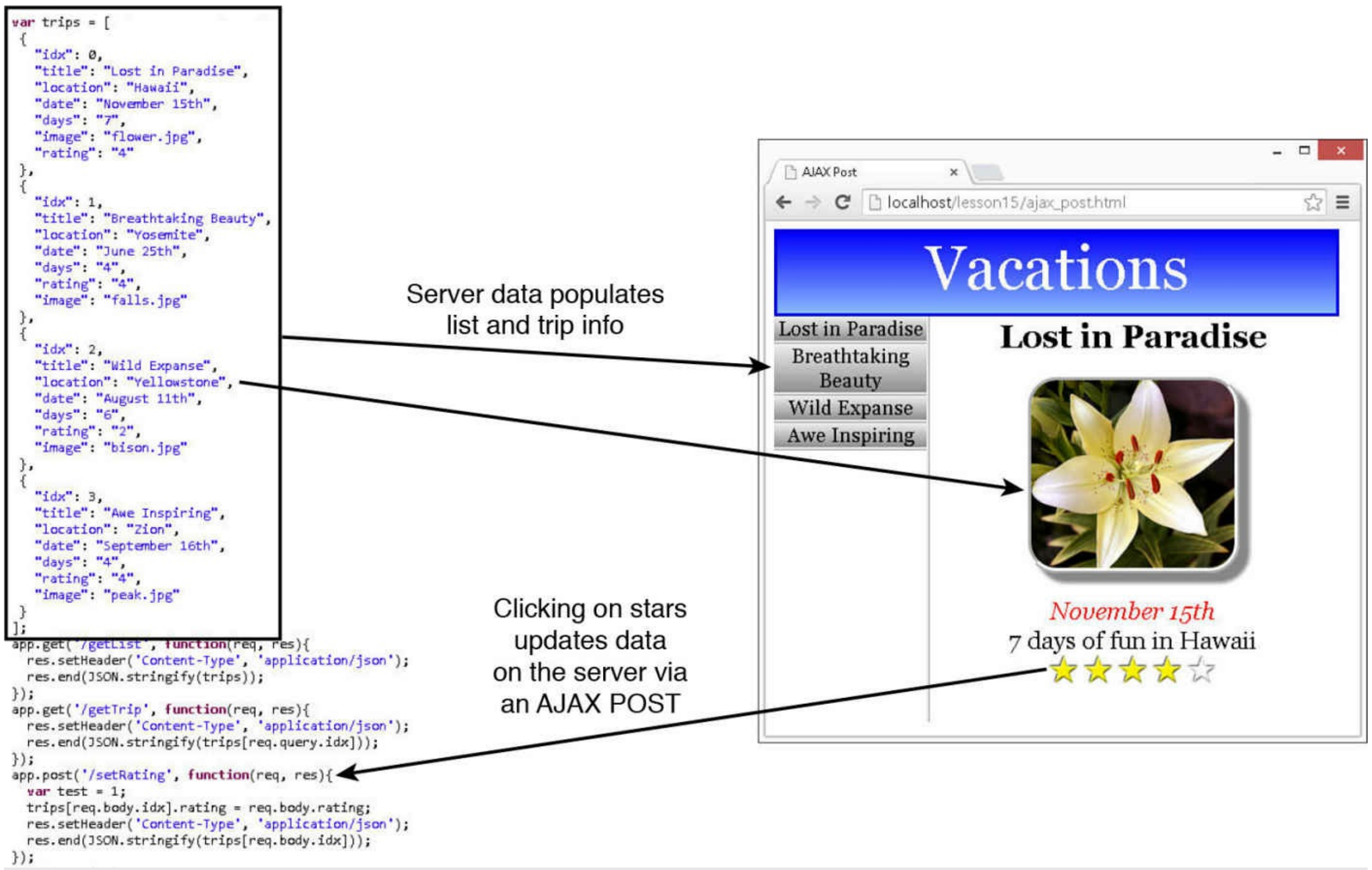
```
11 function getTrip(idx){
12   $.get("/getTrip", {idx: idx}, setTrip);
13 }
```

8. Add the following `sendRating()` function that gets called when the user clicks a star. The index of the `.star` element is sent to the server via a `.post()` method with parameters `idx:idx` and `rating: rating`. The resulting POST request updates the JSON file on the server, permanently storing the new rating value:

Click here to view code image

```
22 function sendRating(){
23   var idx = $("#idx").html();
24   var rating = $(".star").index($(this))+1;
25   $.post("/setRating", {idx: idx, rating: rating}, setTrip);
26 }
```

9. Save all three files and then open the HTML document in a web browser, as shown in Figure 15.6. You should be able to link to the different vacations, see the images, and set the ratings.



```
var trips = [
 {
   "idx": 0,
   "title": "Lost in Paradise",
   "location": "Hawaii",
   "date": "November 15th",
   "days": "7",
   "image": "flower.jpg",
   "rating": "4"
 },
 {
   "idx": 1,
   "title": "Breathtaking Beauty",
   "location": "Yosemite",
   "date": "June 25th",
   "days": "4",
   "rating": "4",
   "image": "falls.jpg"
 },
 {
   "idx": 2,
   "title": "Wild Expanse",
   "location": "Yellowstone",
   "date": "August 11th",
   "days": "6",
   "rating": "2",
   "image": "bison.jpg"
 },
 {
   "idx": 3,
   "title": "Awe Inspiring",
   "location": "Zion",
   "date": "September 16th",
   "days": "4",
   "rating": "4",
   "image": "peak.jpg"
 }
];
app.get('/getList', function(req, res){
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(trips));
});
app.get('/getTrip', function(req, res){
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(trips[req.query.idx]));
});
app.post('/setRating', function(req, res){
  var test = 1;
  trips[req.body.idx].rating = req.body.rating;
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(trips[req.body.idx]));
});
```

Server data populates list and trip info
Clicking on stars updates data on the server via an AJAX POST
AJAX Post
localhost/lesson15/ajax_post.html
Vacations
Lost in Paradise
Breathtaking Beauty
Wild Expanse
Awe Inspiring
Lost in Paradise
November 15th
7 days of fun in Hawaii

FIGURE 15.6 Simple vacation page with the capability to load data dynamically and update the rating via AJAX.

LISTING 15.17 ajax_post.html HTML Document That Loads the jQuery and JavaScript

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>AJAX Post</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript" src="js/ajax_post.js"></script>
    <link rel="stylesheet" type="text/css" href="css/ajax_post.css">
  </head>
  <body>
    <div><div id="banner">Vacations</div>
    <div>
      <div id="leftNav"></div>
      <div id="content">
        <span id="idx" class="hidden"></span>
        <p id="title">Title</p>
        <img id="photo" src="/images/falls.jpg" width="200px"/>
        <p id="date">date</p>
        <p id="info">
        <label>5</label> days of fun in <label>Location</label></p>
        <img class="star" src="/images/star.png" />
        <img class="star" src="/images/star.png" />
        <img class="star" src="/images/star.png" />
        <img class="star" src="/images/star.png" />
        <img class="star" src="/images/star.png" />
      </div>
    </div>
  </body>
</html>
```

LISTING 15.18 ajax_post.js jQuery and JavaScript Code That Implements the AJAX Request That Populates the Page and Updates the Server Data

Click here to view code image

```
function setTrip(data){
  $("#idx").html(data.idx);
  $("#title").html(data.title);
  $("#photo").attr("src", "/images/"+data.image);
  $("#date").html(data.date);
  $("label:first").html(data.days);
  $("label:last").html(data.location);
  $(".star:gt("+(parseInt(data.rating)-1)+")").attr("src", "/images/empty.png");
  $(".star:lt("+(parseInt(data.rating))+")").attr("src", "/images/star.png");
}
function getTrip(idx){
  $.get("/getTrip", {idx: idx}, setTrip);
}
function setList(data){
  var items = [];
  $.each(data, function(key, val) {
    var item = $("<span></span>").html(val.title);
    item.click(function(){getTrip(val.idx)});
    $("#leftNav").append(item);
  });
}
function sendRating(){
  var idx = $("#idx").html();
  var rating = $(".star").index($(this))+1;
  $.post("/setRating", {idx: idx, rating: rating}, setTrip);
}
$(document).ready(function(){
  $.get("/getList", setList).done(function(){
     $("span:first").click(); return false; });
  $(".star").click(sendRating);
});
```

LISTING 15.19 ajax_post.css CSS Code That Styles the Page Elements

Click here to view code image

```
* { font-family:Georgia; }
div { margin:0px; display:inline-block;
  float:left; text-align:center; }
span { background-image: -moz-linear-gradient(top , #f1f1f1, #8F8F8F);
  background-image: -webkit-linear-gradient(top , #f1f1f1, #8F8F8F);
  background-image: -ms-linear-gradient(top , #f1f1f1, #8F8F8F);
  color:black; font-size:20px; float:left; cursor:pointer;
  width:150px; text-align:center; border-bottom:3px ridge; }
p { margin:0px; }
#banner {
  background-image: -moz-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -webkit-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -ms-linear-gradient(top , #0000FF, #88BBFF);
  color:white; height:80px; width:550px; font-size:60px;
  border:3px ridge blue; }
#title { font-weight:bold; font-size:32px; }
#leftNav {width:150px; height:400px; font-size:20px;
border-right:3px ridge;}
#content { height:400px; width: 400px; }
#photo { margin:20px; border:5px ridge white;
  box-shadow: 10px 10px 5px #888888; border-radius:30px;}
#date { color:red; font-style:italic; font-size:24px; }
#info, label { font-size:24px; }
.star {width:30px;}
#idx { display: none; }
```

LISTING 15.20 server_lesson15_ajax_post.js Node.js Server That Will Handle the POST and GET Requests for This Exercise

Click here to view code image

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use('/', express.static('./'));
var trips = [
 {
   "idx": 0,
   "title": "Lost in Paradise",
   "location": "Hawaii",
   "date": "November 15th",
   "days": "7",
   "image": "flower.jpg",
   "rating": "4"
 },
 {
   "idx": 1,
   "title": "Breathtaking Beauty",
   "location": "Yosemite",
   "date": "June 25th",
   "days": "4",
   "rating": "4",
   "image": "falls.jpg"
 },
 {
   "idx": 2,
   "title": "Wild Expanse",
   "location": "Yellowstone",
   "date": "August 11th",
   "days": "6",
   "rating": "2",
   "image": "bison.jpg"
 },
 {
   "idx": 3,
   "title": "Awe Inspiring",
   "location": "Zion",
   "date": "September 16th",
   "days": "4",
   "rating": "4",
   "image": "peak.jpg"
 }
];
app.get('/getList', function(req, res){
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(trips));
});
app.get('/getTrip', function(req, res){
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(trips[req.query.idx]));
});
app.post('/setRating', function(req, res){
  var test = 1;
  trips[req.body.idx].rating = req.body.rating;
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(trips[req.body.idx]));
});
app.listen(80);
```

Using Advanced jQuery AJAX

The concepts already covered in this lesson should take care of most of your AJAX needs. However, they do not cover the full power of the jQuery AJAX interface. The following sections discuss some of the additional AJAX functionality built directly into jQuery.

Reviewing Global Setup

jQuery provides the `.ajaxSetup()` method that allows you to specify options that configure AJAX requests globally throughout the script. Table 15.3 lists some of the options that can be specified when calling `.ajaxSetup()`. For example, the following code sets the default global URL for requests:

Click here to view code image

```
$.ajaxSetup({url:"service.php", accepts:"json"})
```

Using Global Event Handlers

jQuery provides methods to create global event handlers that are called on events, such as initialization or completion for all AJAX requests. The global events are fired on each AJAX request. Table 15.2 lists the methods that you can use to register global event handlers.

Method	Description
`.ajaxComplete(function)`	Registers a handler to be called when AJAX requests are fully complete.
`.ajaxError(function)`	Registers a handler to be called when AJAX requests fail.
`.ajaxSend(function)`	Registers a function to be executed before an AJAX request is sent.
`.ajaxStart(function)`	Registers a handler to be called when the first AJAX request starts.
`.ajaxStop(function)`	Registers a handler to be called when all AJAX requests have completed.
`.ajaxSuccess(function)`	Registers a function to be executed whenever an AJAX request completes successfully.

TABLE 15.2 jQuery Global AJAX Event Handler Registration Methods

An example of using global event handlers to set the class of a form is shown next:

Click here to view code image

```
$(document).ajaxStart(function(){
  $("form").addClass("processing");
});
$(document).ajaxComplete(function(){
  $("form").removeClass("processing");
});
```

Note

Global events are never fired for cross-domain script or JSONP requests, regardless of the value of global.

Implementing Low-Level Ajax Requests

All AJAX request wrapper methods that you have been working with in this lesson are handled underneath through the `.ajax(url [, settings])` interface. This interface is a bit more difficult to use, but it gives you much more flexibility and control of the AJAX request.

The `.ajax()` method is called the same way that `.get()` and `.post()` are; however, the settings argument allows you to set a variety of settings used when making the request.

Table 15.3 lists the more common settings that can be applied to the `.ajax()` method.

Method/Attribute	Description
`accepts`	Specifies the content type(s) the server can send back in the response; for example, application/json.
`async`	Boolean, default is true, meaning the request will be sent and then received asynchronously.
`beforeSend`	Specifies a function that should be run before sending the request.
`complete`	Function to execute when the response is fully received.
`contentType`	Sets the content type to send with the request. The server uses the content type to determine how to parse the data.
`crossDomain`	Boolean. Enables you to send a cross-domain request, such as JSONP.
`data`	Specifies the data payload to send with the request.
`error`	Specifies a function to execute if the request fails.
`headers`	Object that contains the headers with values that should be sent to the server. For example: `{"Content-length": data.length}`
`success`	Function to execute if the response comes back with a 200 status, meaning the request was successful.
`timeout`	Specifies the milliseconds to wait before giving up on getting a response.
`type`	Set to `GET` or `PUT` to specify request type.
`url`	Specifies the URL on the web server to send the request.

TABLE 15.3 Common Settings Available in the .ajax() Request Method

A simple example of using the `.ajax()` interface is shown in the following code:

Click here to view code image

```
$.ajax({
  url:"setEmail",
  type:"get",
  accepts:"json",
  contentType: 'application/x-www-form-urlencoded; charset=UTF-8',
  data: {"first":"Brad", "last":"Dayley"}
}).fail(function(){ alert("request Failed"); });
```

The `.ajax()` method returns a `jqXHR` method that provides some additional functionality, especially when handling the response. Table 15.4 lists some of the methods and attributes attached to the `jqXHR` object.

Method/Attribute	Description
`abort()`	Aborts the request.
`always(function)`	Specifies a function to be called when the request completes.
`done(function)`	Specifies a function to be called when the request completes successfully.
`fail(function)`	Specifies a function to be called when the request fails.
`getAllResponseHeaders()`	Returns the headers included in the response.
`getResponseHeader(name)`	Returns the value of a specific response header.
`setRequestHeader(name, value)`	Sets the value of a HTTP header that will be sent with the AJAX request.
`readyState`	Values: 1—Has not started loading yet. 2—Is loading. 3—Has loaded enough and the user can interact with it. 4—Fully loaded.
`status`	Contains the response status code returned from the server; for example, 200 status means the request was successful.
`statusText`	Contains the status string returned from the server.

TABLE 15.4 Common Methods and Attributes of the `jqXHR` Object Returned by `.ajax()`

Summary

AJAX is the basic communication framework between jQuery/JavaScript and the web server. Using AJAX allows you to get additional data from the web server and use it to dynamically update the web page instead of completely reloading it. This enables you to provide a much better experience for the user.

In this lesson, you learned the ins and outs of AJAX. You were able to implement several examples that gave you experience with GET and POST requests, as well as handling different types of data, such as HTML, JSON, and XML.

Q&A

Q. Why is there a GET and a POST request and not just one request?

A. The answer is optimization. Although you could do everything with just POST requests, optimizations can be made at the web server as well as the browser by using GET requests that have the parameters directly in the URL. This is especially true with caching.

Q. Are there any other methods to send cross-domain requests besides JSONP?

A. Yes. Flash and Silverlight both have mechanisms that allow you to send cross-domain requests. HTML5 also introduces the `postMessage` interface designed to allow cross-domain requests.

Workshop

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

Quiz

1. Would you use a GET or a POST request to set a new password on a web server?
2. True or false: The `.done()` event handler function is called only if the AJAX request succeeds.
3. What data type is the response data converted to for XML data?
4. What data type is the response data converted to for JSON data?
5. Is it possible to specify the request headers sent with a GET request via jQuery?

Quiz Answers

1. POST.
2. True.
3. DOM object.

4. Basic JavaScript object.

5. Yes. You can use `.ajax()` with the headers parameter, or you can use `.ajaxSetup()` to set the headers parameter globally.

Exercises

1. Modify the example in Listings 15.9, 15.10, 15.11, and 15.12. Add an additional date value to each image in the JSON file. Then add the date value along with the image on the web page by including it in the image caption.

2. Modify the example in Listings 15.13, 15.14, 15.15, and 15.16. Add an additional column for rating. You will need to update the XML file, as well as the AJAX response handlers, to add the additional column to the rows. Also, you will need to fix up the CSS.

Part IV: Utilizing jQuery UI

Lesson 16. Introducing jQuery UI

What You'll Learn in This Lesson:

- How to download and add the jQuery UI libraries
- How to create custom jQuery UI themes
- New functionality jQuery UI provides over jQuery alone
- Using jQuery UI selectors
- How to dynamically position UI elements using jQuery UI

jQuery UI is an additional library built on top of jQuery. The purpose of jQuery UI is to provide a set of extensible interactions, effects, widgets, and themes that make it easier to incorporate professional UI elements in your web pages. In this lesson, you get a chance to download and implement jQuery UI in some web pages. The purpose of this lesson is to introduce you to how jQuery interacts with HTML, CSS, jQuery, and JavaScript.

Getting Started with jQuery UI

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

jQuery UI saves developers time by providing prebuilt UI elements, such as calendars and menus, with interactions, such as dragging and resizing, right out of the box. The following sections introduce you to the library; you learn how to download it and apply it to your projects.

The jQuery UI library is made up of several components that are used to build, manage, and interact with the UI elements on the web page. These elements can be categorized into the following:

- **JavaScript**—jQuery UI comes with its own .js file that hooks into the main jQuery library. This file must be loaded in your web pages along with jQuery to implement jQuery UI components.
- **CSS**—Several css files are included with jQueryUI. The main one is jquery-ui.css. These files contain all the CSS settings pertaining to jQuery UI and in particular to the theme that is represented.
- **Images**—Query also provides several images that are used to build some of the UI components.

Getting the jQuery UI Library

You can implement jQuery UI in your projects in one of two ways. If you want to use a specific theme already provided, you can select one of the themes in jQuery UI section of the jQuery CDN location at https://code.jquery.com.

For example, the following lines load the jQuery UI library and the Smoothness theme from the jQuery CDN locations:

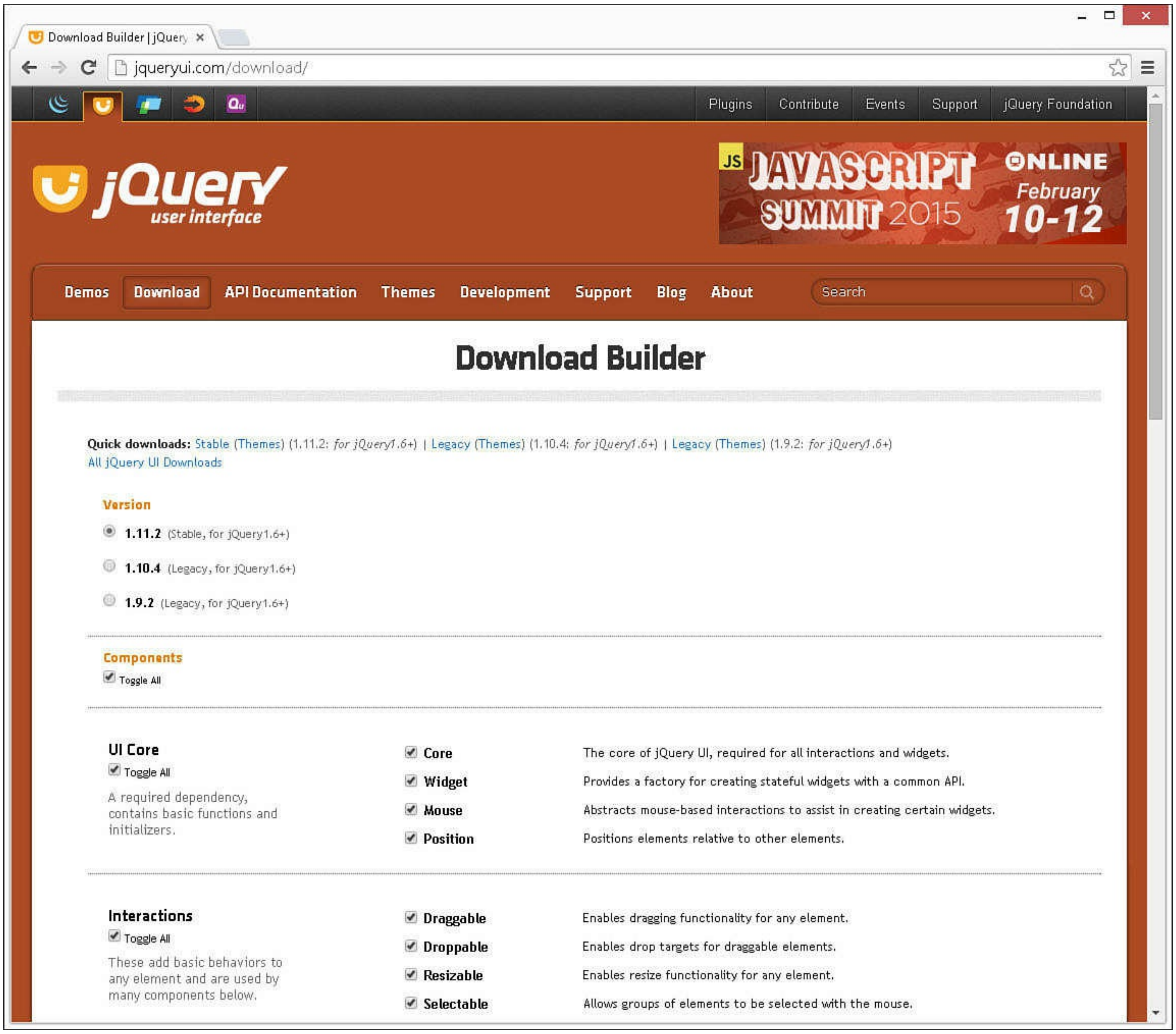
Click here to view code image

```
<script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
<script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
<link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
```

You can also download the library along with a customized theme and store them on your own web server. In that case, you would use your own location for the `src` and `href` attributes rather than the CDN locations for the jquery-ui.js and jquery-ui.css files.

The jQuery library can be downloaded from the following location by selecting the options that you want to include, shown in Figure 16.1, and clicking the Download button at the bottom. This downloads the jQuery UI files:

```
http://jqueryui.com/download
```



Download Builder

Quick downloads: Stable (Themes) (1.11.2: *for jQuery1.6+*) | Legacy (Themes) (1.10.4: *for jQuery1.6+*) | Legacy (Themes) (1.9.2: *for jQuery1.6+*)
All jQuery UI Downloads

Version

[✓ **1.11.2**] (Stable, for jQuery1.6+)

1.10.4 (Legacy, for jQuery1.6+)

1.9.2 (Legacy, for jQuery1.6+)

Components

[✓] Toggle All

UI Core [✓] Toggle All A required dependency, contains basic functions and initializers.	[✓] **Core**	The core of jQuery UI, required for all interactions and widgets.
	[✓] **Widget**	Provides a factory for creating stateful widgets with a common API.
	[✓] **Mouse**	Abstracts mouse-based interactions to assist in creating certain widgets.
	[✓] **Position**	Positions elements relative to other elements.
Interactions [✓] Toggle All These add basic behaviors to any element and are used by many components below.	[✓] **Draggable**	Enables dragging functionality for any element.
	[✓] **Droppable**	Enables drop targets for draggable elements.
	[✓] **Resizable**	Enables resize functionality for any element.
	[✓] **Selectable**	Allows groups of elements to be selected with the mouse.

FIGURE 16.1 Using jQuery UI download builder to build and download a custom version of jQuery UI.

Using ThemeRoller to Create a Custom Theme

In addition to the basic jQuery UI theme, you can also use the jQuery UI ThemeRoller, shown in Figure 16.2, to select some different custom themes or customize your own theme. A theme defines the colors, border radius, and multiple other styles applied to jQuery UI widgets and elements.

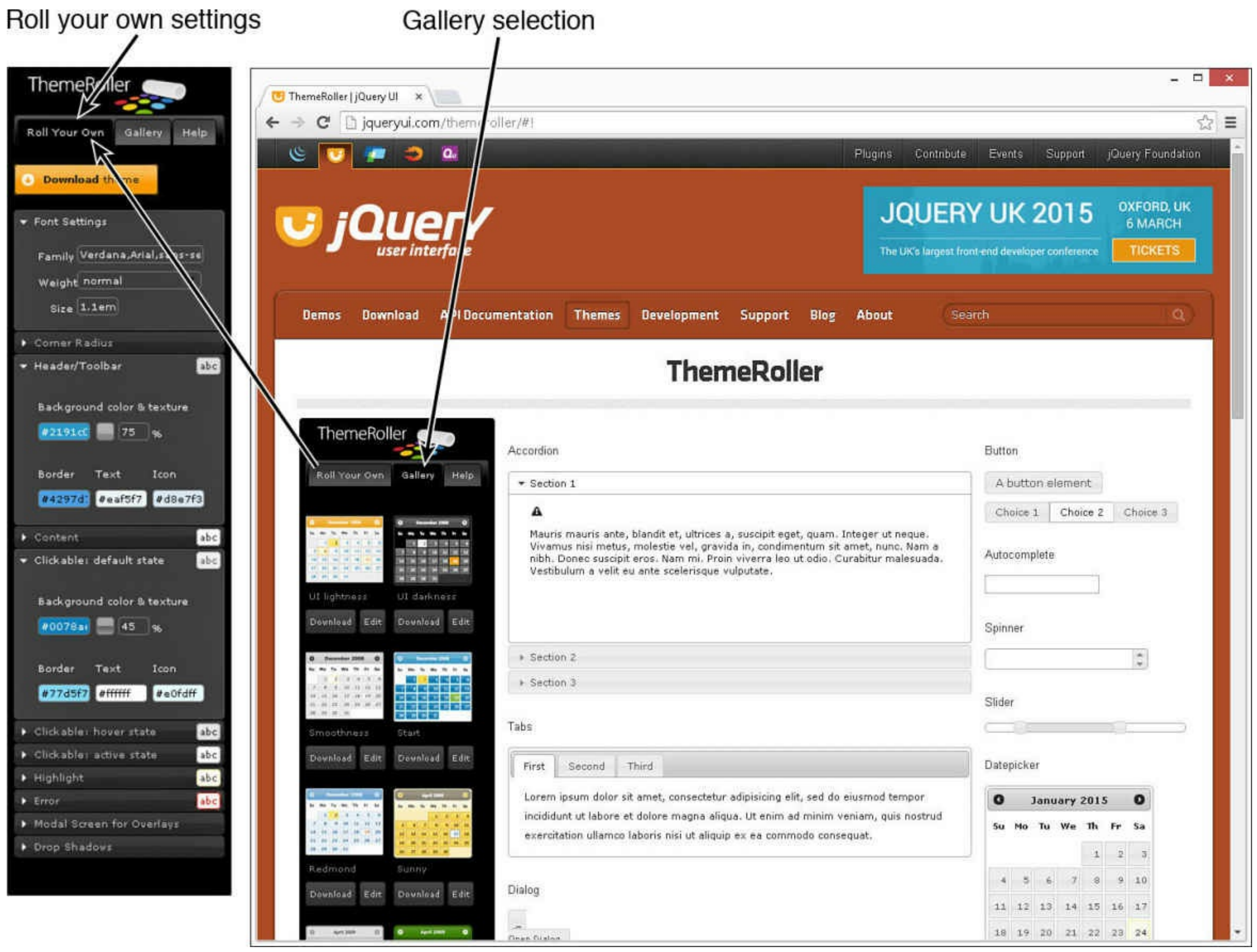


FIGURE 16.2 Using jQuery UI ThemeRoller to define a custom theme.

To access the jQuery UI ThemeRoller:

1. Open the following URL in your browser:
   http://jqueryui.com/download/
2. Scroll down to the bottom of the download page shown in Figure 16.1 and click the Design a Custom Theme link. This brings up the ThemeRoller shown in Figure 16.2.
3. Select the Gallery tab and view the gallery of prebuilt themes shown in Figure 16.2.
4. Select the gallery that most fits your needs.
5. Select the Roll Your Own tab, also shown in Figure 16.2, and specify as many of the specific settings as you want to define.
6. Click the Download Theme button. This takes you back to the main download page.
7. Click the Download button to download the jQuery UI files.

Try it Yourself: Adding jQuery UI to a Web Page

Use the following steps to download the jQuery UI library and add it to the project you are using for this book:

1. In Eclipse, create the lesson16, lesson16/js, and lesson16/css folders, and then add the lesson16/date_picker.html file.
2. Create the lesson16/js and lessong16/css folders.
3. Add the code shown in Listing 16.1. This is straightforward code to validate that the jQuery UI library is installed properly.
4. The following lines load the jQuery library, jQuery UI library, and the smoothness CSS theme for jQuery UI:

Click here to view code image

```
<script type="text/javascript"
src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
<script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
<link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-
ui.css">
```

5. The following line adds a `<div>` element and assigns the `datepicker id` to it in the HTML document:

Click here to view code image

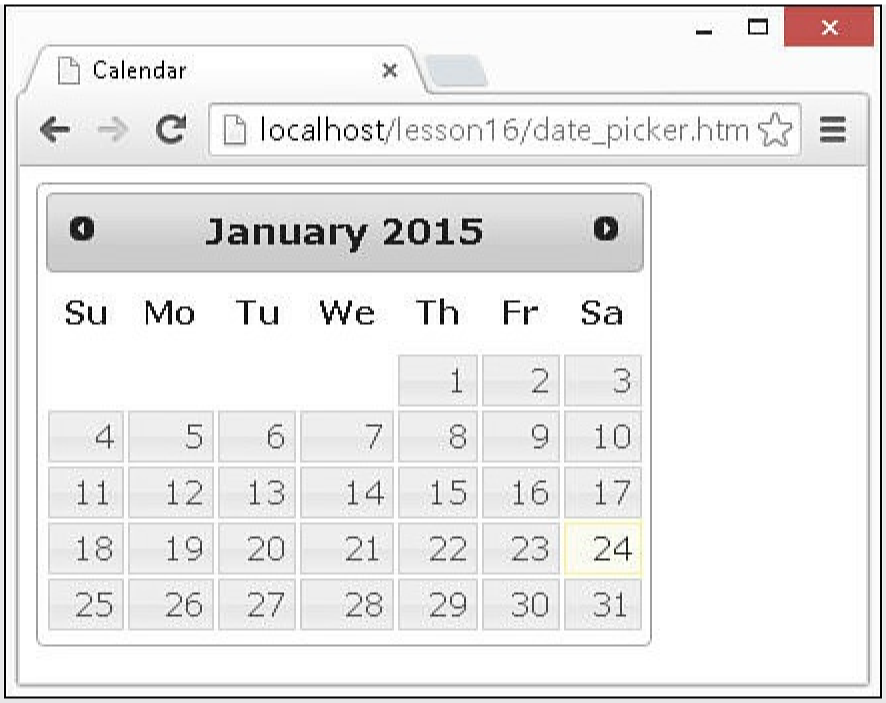
```
<div id="datepicker"></div>
```

6. The following `ready()` function adds the jQuery UI Datepicker element to the `<div>`:

Click here to view code image

```
$(document).ready(function(){
  $( "#datepicker" ).datepicker();
});
```

7. Save the file, and then open the HTML document in a web browser, as shown in Figure 16.3. You should see the date picker displayed correctly if the libraries are installed properly.



Su	Mo	Tu	We	Th	Fr	Sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

FIGURE 16.3 Simple jQuery UI date picker.

LISTING 16.1 date_picker.html HTML Document That Adds the jQuery UI Libraries and Renders a Date Picker

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Calendar</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <script>
      $(document).ready(function(){
        $( "#datepicker" ).datepicker();
      });
    </script>
  </head>
  <body>
    <div id="datepicker"></div>
  </body>
</html>
```

Applying jQuery UI in Your Scripts

Now that you've got the jQuery UI library loaded and ready to go, you can learn about some of the enhancements that jQuery UI adds over jQuery. This section covers some new functionality as well as some enhancements to the jQuery library. The meat of jQuery UI—namely, the effects, interactions, and widgets—are covered in upcoming lessons.

Understanding Enhanced jQuery UI Functionality

jQuery UI is in many ways an extension of the jQuery library. To get you up to speed so that you can begin implementing jQuery UI components, the following sections discuss some of the important upgrades from jQuery.

Adding and Removing Unique IDs

jQuery UI provides a couple of additional methods to jQuery objects that allow you to easily add and remove unique IDs to a set of elements. This is especially useful when you dynamically create a bunch of new elements that you must be able to access by ID later.

To add unique IDs, use the `.uniqueId()` method. This method checks each element in the set and adds an `id` attribute if one is not present. The new id will have a prefix of `"ui-id-"`. If the element already has an `id` attribute, it is not altered.

You can later delete the unique IDs using the `.removeUniqueId()` method on the set. Only elements that had IDs created by `.uniqueId()` are affected.

For example, the following code adds `id` values to all `<div>` elements and then later removes them:

Click here to view code image

```
var divs = $("div").uniqueId();
...
do something
...
$("div"). removeUniqueId ();
```

Getting the ScrollParent

Another helpful addition in jQuery UI is the `.scrollParent()` method. This method searches the ancestors of the element and returns the first parent element that is scrollable. This method works only on jQuery objects that have a single element in the set.

Getting the zIndex

Another helpful addition in jQuery UI is the `.zIndex()` method. This method returns a numeric `z-index` value of the element, if it has one, or the first ancestor that does. This enables you to quickly determine the stacking placement of any item on the page.

Async Focus

jQuery UI extends the `.focus(delay [. callback])` method of jQuery objects to allow for a `delay` before setting the focus and including a `callback` function that will be executed when the element gets the focus. The `delay` is specified in milliseconds.

This functionality has a wide range of uses, from using a timer, to automatically selecting a form element, to forcing the refocus of an element. For example, the following code adds a half-second delay before setting the focus to an element `#timedInput`:

Click here to view code image

```
$("#timedInput").focus(500, function(){
  $(this).val("Enter Text Now");
});
```

Using New Selectors in jQuery UI

One great feature of jQuery UI is the capability to extend the jQuery selectors that are already pretty extensive. These new selectors make it easier to narrow down selections specific to UI element needs. The following sections discuss each of the new selectors.

Using the :data() Selector

One of my favorite selectors in jQuery UI is the `:data()` selector. This selector enables you to filter elements based on a specific key that was added to elements using the `.data()` jQuery method. For example, the following code adds a `color` value to all `<span>`, `<div>`, and `<p>` elements, and then uses the `:data()` selector to set those colors on the elements:

Click here to view code image

```
$(p).data("color", "red");
$(span).data("color", "blue");
$(div).data("color", "green");
$(":data(color)").each(function(){
  $(this).css({color:$(this).data("color")});
});
```

:focusable

The `:focusable` selector allows you to limit elements to only those that can receive focus. For example, the following statement limits the changes to only those form elements that can receive focus:

Click here to view code image

```
$("form:focusable").each(function(){
  $(this).css({color:red});
});
```

:tabbable

The `:tabbable` selector is similar to the `:focusable` selector. It allows you to limit elements to only those that can be tabbed to. For example, the following statement limits the changes to only those form elements that can be tabbed to:

Click here to view code image

```
$("form:tabbable").each(function(){
  $(this).css({color:red});
});
```

This filter is very useful, especially when you're trying to exclude elements that are disabled.

Tip

Elements that have a negative tab index are `:focusable` but not `:tabbable`.

Try it Yourself: Applying jQuery UI Selectors Based on Data Values

In this example, you add several `<div>` elements to the web page. Then in jQuery, you add an image data value to some of them. Using the `:data()` selector, you then apply different changes to the elements with image data than those without image data.

The code for the example is in Listings 16.2, 16.3, and 16.4. Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson16/jquery_image_adder.html, lesson16/js/jquery_image_adder.js, and lesson16/css/jquery_image_adder.css files.
2. Add the code shown in Listing 16.2. The following lines are used to load the jQuery library, jQuery UI library, and CSS files:

Click here to view code image

```
<script type="text/javascript"
src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
<script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
<script type="text/javascript" src="js/jquery_image_adder.js">
```

```
</script>
    <link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-
ui.css">
    <link rel="stylesheet" type="text/css"
href="css/jquery_image_adder.css">
```

3. Open the jquery_image_adder.js file and add a basic `.ready()` function.

4. Add the following three lines. These lines add the `image` data value to some of the `<div>` elements in the document:

Click here to view code image

```
$("#arch").data("image", "images/arch.jpg");
$("#volcano").data("image", "images/volcano.jpg");
$("#pyramid").data("image", "images/pyramid2.jpg");
```

5. Add the following `click` handler function that will first get all the `<div>` elements that have an `image` value and use the `image` value to set the `src` of an `<img>` element that gets prepended. Then the function will get all `<div>` elements that do not have an `image` value and append the generic insert.png image:

Click here to view code image

```
$("#add").click(function(){
  $("div:data(image)").each(function(){
    $(this).prepend(
        $('<img></img>').attr("src", $(this).data("image")));
  });
  $("div:not(:data(image))").each(function(){
    $(this).prepend(
        $('<img></img>').attr("src", "/images/insert.png"));
  });
});
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 16.4. When you click the Add Images button, you should see the images pop up for the `<div>` elements that had an `image` value and the generic image for those that do not.

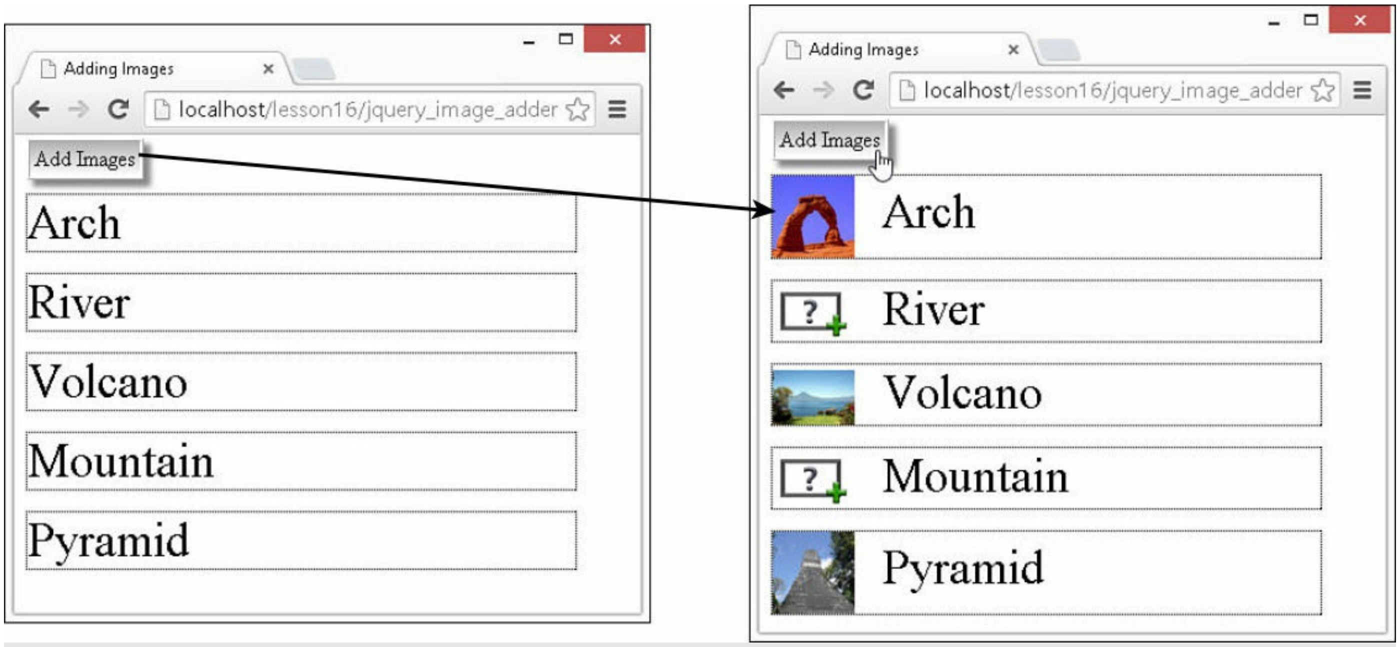


FIGURE 16.4 Using the jQuery UI selector to update the elements that have an image data value.

LISTINGS 16.2 jquery_image_adder.html HTML Document That Adds the Web Page

[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Adding Images</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/jquery_image_adder.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/jquery_image_adder.css">
  </head>
  <body>
    <span id="add">Add Images</span>
    <div id="arch">Arch</div>
    <div id="river">River</div>
    <div id="volcano">Volcano</div>
    <div id="mountain">Mountain</div>
    <div id="pyramid">Pyramid</div>
  </body>
</html>
```

LISTING 16.3 jquery_image_adder.js jQuery and jQuery UI Code That Uses the :data() Selector to Select Elements

Click here to view code image

```
$(document).ready(function(){
  $("#arch").data("image", "/images/arch.jpg");
  $("#volcano").data("image", "/images/volcano.jpg");
  $("#pyramid").data("image", "/images/pyramid2.jpg");
  $("#add").click(function(){
    $("div:data(image)").each(function(){
      $(this).prepend(
          $('<img></img>').attr("src", $(this).data("image")));
    });
    $("div:not(:data(image))").each(function(){
      $(this).prepend(
          $('<img></img>').attr("src", "/images/insert.png"));
    });
  });
});
```

LISTING 16.4 jquery_image_adder.css CSS Code That Styles the Page Elements

Click here to view code image

```
img {
    width:60px; margin-right:20px; vertical-align:middle; }
div {
    margin-top:15px; border:1px dotted;
    width:400px; font-size:35px;}
span {
    background-image: -moz-linear-gradient(top , #B1B1B1, #FFFFFF);
  background-image: -webkit-linear-gradient(top , #B1B1B1, #FFFFFF);
  background-image: -ms-linear-gradient(top , #B1B1B1, #FFFFFF);
  border:3px ridge white; box-shadow: 5px 5px 5px #888888;
  padding:3px; cursor:pointer; }
```

Positioning UI Elements with jQuery UI

A great advantage that jQuery UI provides is the capability to position elements relative to other elements and handle collisions. This is done by extending the `.position()` method to allow for an options object that defines the relative positions between the jQuery element and other elements or event locations.

For example, to position an element `#div1` to the right of `#div2`, you could use the following:

Click here to view code image

```
$("#div1").position("my:"left", at:"right", of:"#div2");
```

Pretty simple. Table 16.1 describes the options that jQuery UI provides to the `.position()` method.

Option	Description

`my`	Specifies the relative position of the current jQuery object to use for alignment. Acceptable values are as follows:
	`"right", "left", "top", "bottom", "center"`
	
	You can also combine these, for example:
	`"left top", "right bottom", "left center", "center center"`
	
	These positions can also be adjusted using numerical or percentage values. For example, the following places the item `-10` pixels to the left and `20%` of the height down:
	`"left-10 top+20%"`

`at`	Specifies the relative position in the target element to use for alignment. This option can be set to the same values as the `my` option.
`of`	Specifies a selector, a DOM element object, a jQuery object, or a JavaScript `Event` object. In the case of a jQuery object, the first element in the set is used. In the case of a JavaScript `Event` object, the `pageX` and `pageY` properties are used.
`collision`	Specifies how to handle instances where the element overflows the window in some direction. When this option is specified and the object overflows the current window, it is moved based on the collision value. Accepted values are as follows: ▸ **`"flip"`**—Flips the element to the opposite side of the specified target, then runs the collision detection again. Whichever side allows more of the element to be visible is then used. ▸ **`"fit"`**—Repositions the element away from the edge of the window. ▸ **`"flipfit"`**—Tries to apply the flip logic by placing the element on whichever side allows more of the element to be visible. Then it tries the fit logic to ensure as much of the element is visible as possible. ▸ **`"none"`**—Does not apply any collision detection.

`using`	Allows you to specify a callback function that handles the actual positioning. The first argument passed to the function is the position value that can be used to set the CSS properties using `.css()`/`.animate()`. The second parameter is an object that contains the dimensions and relative positions of both the source and target elements.
`within`	Allows you to specify the container object to use when determining if there is a collision. This defaults to the JavaScript `window` object, but can be set to a selector, a DOM element object, or a jQuery object.

TABLE 16.1 Option Settings Used When Positioning Elements with jQuery UI .position()

Try it Yourself: Using jQuery UI to Position Images on a Web Page

The best way to help you understand jQuery UI positioning is to give you some hands-on experience. In this example, you use jQuery UI position to position static image elements as well as a dynamic one that moves with the mouse. You add some collision protections to keep the image from leaving a <div> element.

The code for the example is in Listings 16.5, 16.6, and 16.7. Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson16/dynamic_positioning.html, lesson16/js/dynamic_positioning.js, and lesson16/css/dynamic_positioning.css files.
2. Add the code shown in Listing 16.5 and Listing 16.7 to the HTML and CSS files.
3. Open the dynamic_positioning.js file and a `.load()`.
4. Inside the `.load()` function, add the following lines to position `#img2` at the bottom-right corner of `#img1` and position `#img3` at the bottom-right corner of `#img2`:

Click here to view code image

```
$("#img2").position(
    {my:"left top", at:"right bottom", of:"#img1"});
$("#img3").position(
    {my:"left top", at:"right bottom", of:"#img2"});
```

5. Add the following `mousemove` event handler to reposition `#img3` with the mouse movement. Notice that `collision` is set to `"flip" within` the `<div>` element so that the image will not be repositioned outside:

Click here to view code image

```
$("div").mousemove(function(e) {
  $("#img4").position({ my:"left top", at:"center", of:e,
                              collision:"flip", within:"div" });
     })
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 16.5. You should see three images placed on the page, and the fourth image will track with the mouse cursor as it is moved.

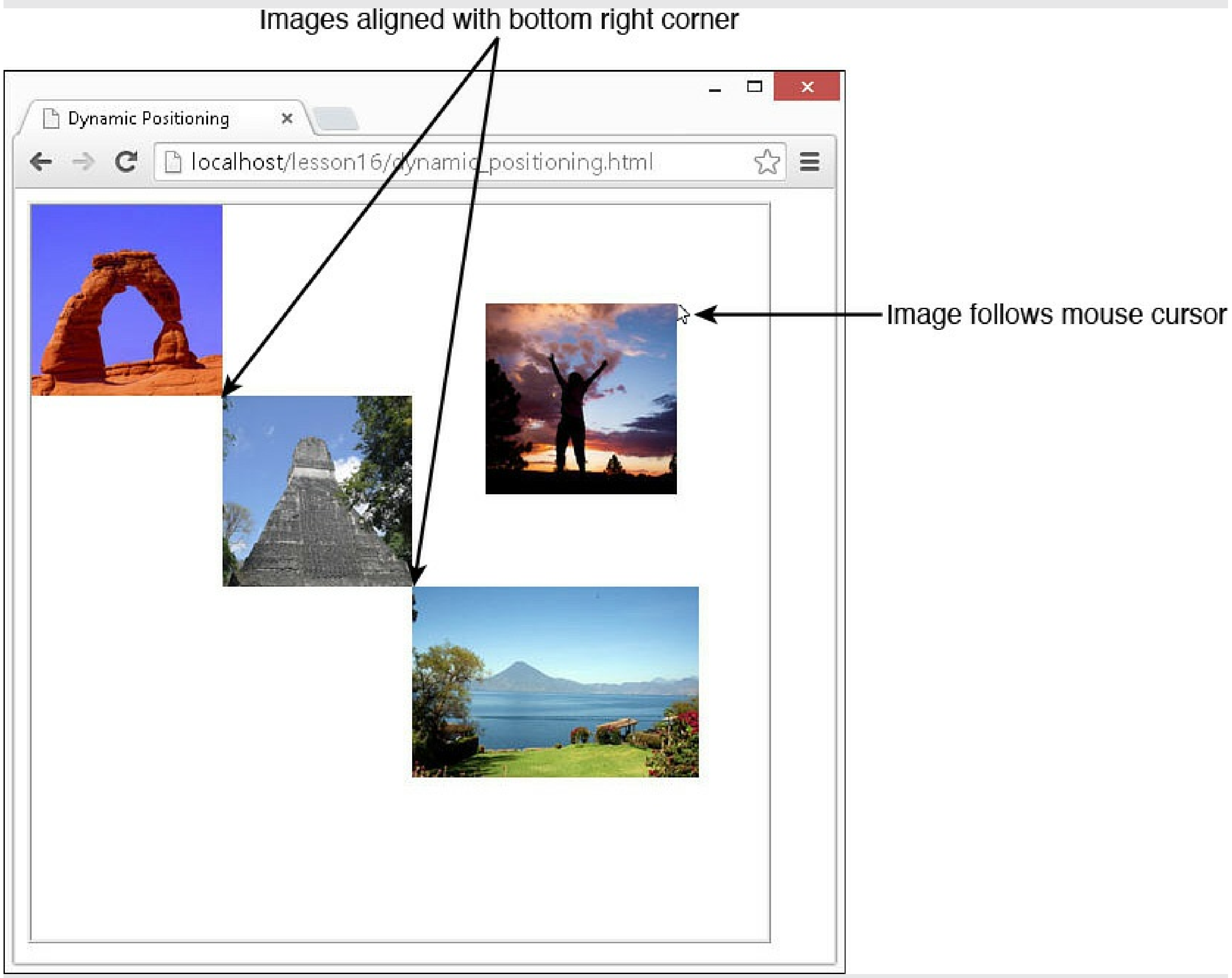

Images aligned with bottom right corner
Dynamic Positioning
localhost/lesson16/dynamic_positioning.html
Image follows mouse cursor

FIGURE 16.5 Positioning images using the `.position()` method in jQuery UI.

LISTING 16.5 dynamic_positioning.html HTML Document That Adds the Images to the Web Page

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dynamic Positioning</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/dynamic_positioning.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/dynamic_positioning.css">
  </head>
  <body>
    <div>
      <img id="img1" src="/images/arch.jpg">
      <img id="img2" src="/images/pyramid2.jpg">
      <img id="img3" src="/images/volcano.jpg">
      <img id="img4" src="/images/jump.jpg">
    </div>
  </body>
</html>
```

LISTING 16.6 dynamic_positioning.js jQuery and jQuery UI That Dynamically Positions the Images

Click here to view code image

```
$(window).load(function(){
  $("#img2").position(
      {my:"left top", at:"right bottom", of:"#img1"});
  $("#img3").position(
      {my:"left top", at:"right bottom", of:"#img2"});
  $("div").mousemove(function(e) {
    $("#img4").position({ my:"left top", at:"center", of:e,
                          collision:"flip", within:"div" });
  })
});
```

LISTING 16.7 dynamic_positioning.css CSS Code That Styles the Page

Click here to view code image

```
img {
    position: absolute; height: 130px; width:auto; }
div {
    height:500px; width:500px; border:3px ridge; }
```

Summary

In this lesson, you downloaded and implemented jQuery UI in a few examples. You learned that jQuery UI extends jQuery with some additional functionality, such as new selectors, as well as enhances existing jQuery functionality, such as element positioning. You implemented some examples to illustrate how to use jQuery in your web pages.

Q&A

Q. Is there anything that can be done in jQuery UI that I can't do myself in jQuery and JavaScript?

A. No, but that's not the point. The point is that jQuery and jQuery UI will save you a ton of time.

Q. Can I use more than one theme at a time?

A. No, the themes will conflict with each other. You can, however, have multiple themes in different locations on your website and then dynamically adjust which .css files get loaded. This allows some users to have one theme and other users to have another.

Workshop

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

Quiz

1. What jQuery UI selector would you use to isolate elements that happen to have a specific data value assigned to them?
2. True or false: There is no way to get the scroll parent for an element.
3. How would you delay setting the focus on an element?
4. How can you easily reposition an element directly to the right of another element using jQuery UI?

Quiz Answers

1. `:data()`
2. False. You can use the `.scrollParent()` method to get the nearest scrolling container.
3. Use the `.focus()` method in jQuery UI with a delay value.
4. Use the `.position()` method with one element at `"right"` and other

"left".

Exercises

1. Open the code in Listings 16.2, 16.3, and 16.4, and modify it to add an additional data value that specifies the color; for example, `.data("color","red")`. Then, when the user clicks the Add Images button, change the color of the text using `.css()`.
2. Open the code in Listings 16.5, 16.6, and 16.7, and modify it so that the images overlap each other as they cascade. The simplest way to do this is to add negative values to the `.position()` settings.

Lesson 17. Using jQuery UI Effects

What You'll Learn in This Lesson:

- Methods to apply effects using the jQuery UI library
- How easing effects make animation changes variable
- Ways to add cool effects when hiding/showing elements
- How to apply effects to class changes
- Applying animation effects when repositioning elements

jQuery UI provides a rich set of animation-type effects that can be applied to elements on your web pages. There are a couple of reasons why animating elements is a good thing. One is to leave the user with an impression that the website is interactive and fun to use. The second is to help users understand the visual changes that are taking place in your dynamic scripts.

In this lesson, you see the improvements that jQuery UI provides in animation effects. You will be able to apply effects directly to elements, or you can apply effects when making class, visibility, or position changes.

Applying jQuery UI Effects

The purpose of this section is to introduce you to the effects that jQuery UI provides. This section discusses each of the effects and how to apply them using the `.each()` method. You are also introduced to the multitude of easing functions that provide a variable aspect to how values are applied during the effect animation.

Understanding jQuery UI Effects

jQuery UI effects are just animations to CSS position, size, and visibility properties. The animated changes are implemented in such a way as to create visual effects that give users a better experience.

For example, suppose a user tries to log in with an invalid password. In addition to the form validation message, you can also use jQuery UI effects to make the login button shake, which will catch the user's attention better, letting the user know the login failed. These are subtle changes to the web page, but they can have a large impact on the user experience.

Table 17.1 lists the effects with values that can be applied to manipulate them. This should give you an idea of the effects possible with jQuery UI. You implement some of these effects later in this lesson.

Method	Options	Description
`blind`	`direction`	Provides the effect of "pulling the blinds" up by rolling the element up from the bottom. `direction` option can be set to `up`, `down`, `left`, `right`, `vertical`, or `horizontal`.
`bounce`	`distance` `times`	Provides a bouncing effect by repositioning the element up and down vertically. `distance` is specified in pixels. `times` is the number of times the element bounces.
`clip`	`direction`	The element slides down as the bottom is erased, simulating the bottom being clipped off. `direction` option can be set to `vertical` or `horizontal`.
`drop`	`direction`	Slides the element as it fades in or out. `direction` option can be set to `up`, `down`, `left`, or `right`.
`explode`	`pieces`	Slices the element into equal pieces that fade away in different directions. `pieces` should be a perfect square number (4,9,...).
`fade`		Slowly fades the element in or out.
`fold`	`size` `horizFirst`	Folds the element in one direction and then a second. `size` is the number of pixels to fold down to. `horizFirst` specifies `true` or `false` on which direction to fold first.

highlight	color	Adds a color highlight to the image. color specifies the hex color value; for example, #FF0000.
puff	percent	Scales an element up at the same time it hides it. percent value specifies the percentage to scale the element to; for example, 50=smaller, 150=bigger.
pulsate	times	Fades the element in and out quickly, simulating a pulsing effect. times value is the number of times to fade.
scale	direction origin percent scale	Shrinks or enlarges the element to a vanishing point. direction can be both, vertical, or horizontal. origin is an array specifying the vanishing point, which defaults to, for example: ["middle", "center"] percent specifies the percentage to scale to. scale specifies what part of the element to resize: box, content, both.
shake	direction distance times	Animates rapid position changes vertically or horizontally. direction can be left, right, up, or down. distance is the number of pixels to shake. times is the number of times to shake.
size	to origin scale	Animates resizing the element. to specifies the new height and width: {height:#, width:#} origin specifies the vanishing point array; for example: ["middle", "right"]

		`scale` specifies what part of the element to resize: `box`, `content`, `both`.
`slide`	`direction` `distance`	Animates the element to simulate a sliding effect. `direction` can be `left`, `right`, `up`, or `down`. `distance` to slide up to the height or width of the element.
`transfer`	`className` `to`	Animates a wire frame transitioning from one element to another. `className` specifies class the transfer element will receive. `to` specifies the element to transfer to.

TABLE 17.1 jQuery UI Effects

Setting the Effect Animation Easing

The easing function sets a value path that the effect uses when animating the effect. You have already seen the linear and swing easing in jQuery animations. jQuery UI adds a large number of new easing functions that can provide some fun effect animation.

The simplest way to illustrate how easing works is to show you the graphs published at the following location and shown in Figure 17.1. Think of the horizontal axis of the graphs as duration time, where left is 0 and the right is complete. Think of the vertical axis of the graph as how complete the transition of the effect is. For example, in a fade-out transition, the bottom would be fully opaque and the top would be fully transparent: http://api.jqueryui.com/easings/

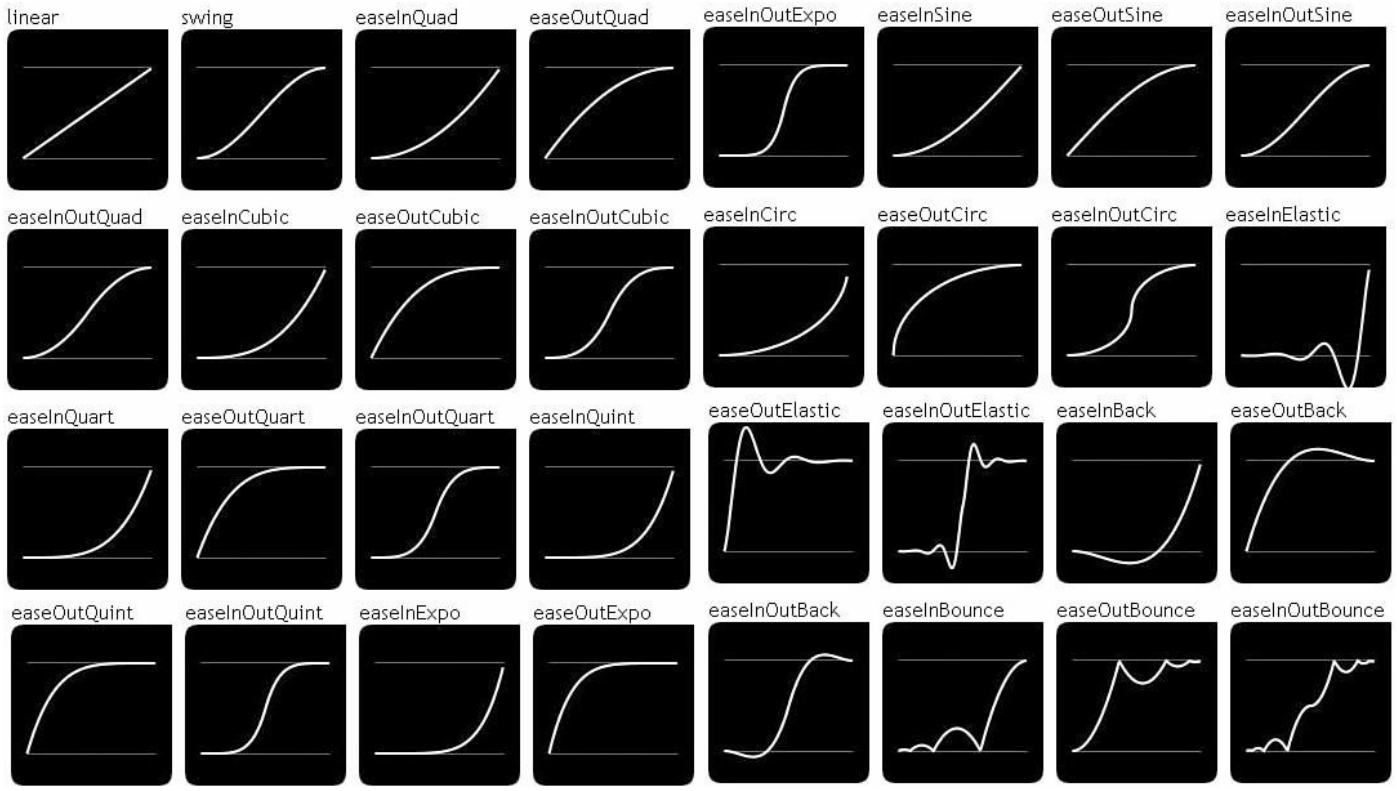


FIGURE 17.1 jQuery UI easing functions.

Adding Effects to jQuery Objects

There are multiple ways to apply effects to jQuery objects. Effects can be added as a part of another transition, such as a class change or visibility change. You can also apply effects to an element using the `.effect()` method. The `.effect()` method has the following syntax:

Click here to view code image

```
.effect( effect [, options ] [, duration ] [, complete ] )
```

In the `.effect()` method, effect is the name of the effect and `options` is an object containing the option values. Table 17.1 lists the effect names and options that you can apply to each effect. The `duration` is specified in milliseconds, and you can add an optional `complete` handler function that will be executed when the effect has been applied.

The following example illustrates the full syntax of applying a `size` effect to an element:

Click here to view code image

```
("img").effect("size",
               {to:{height:100, width:100}, origin:["right","top"],
scale:"box"},
               3000,
               function(){alert("effect complete");});
```

Try it Yourself: Adding jQuery UI Effects

In this example, you apply several effects to `<img>` elements. You add four images to the web pages and apply a different effect on each when the user clicks the image. The purpose of the example is to familiarize you with how to implement different effects, set the effect `options`, and apply a `complete` function.

The code for the example is in Listings 17.1, 17.2, and 17.3. Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson17, lesson17/js, and lesson17/css folders, and then add the lesson17/jquery_effects.html, lesson17/js/jquery_effects.js, and lesson17/css/jquery_effects.css files.
2. Add the code shown in Listing 17.1 and Listing 17.3 to the HTML and CSS files.
3. Open the jquery_effects.js file and add a `.ready()` function.
4. Add the following `click` handler to `#img1`. The `click` handler applies a basic shake effect; 20 pixels in the down direction first and shake 5 times. 3000 milliseconds means the effect will take 3 seconds:

Click here to view code image

```
$("#img1").click(function(e) {
  $(this).effect("shake",
      { direction:"down", distance:20, times:5}, 3000);
});
```

5. Add the following `click` handler to `#img2`. The `click` handler applies a scale effect in both directions to the middle, right vanishing point. The effect scales the image down to 40 percent. Also notice that the `easeInBounce easing` is added to adjust the flow of the animation:

Click here to view code image

```
$("#img2").click(function(e) {
  $(this).effect("scale",
    { direction:"both", origin:["middle", "right"],
      percent:40, scale:"box", easing:"easeInBounce"}, 3000);
});
```

6. Add the following `click` handler to `#img3`. The `click` handler applies a double effect. The first effect is a slide in the downward direction for 200 pixels and the second, which slides right, is placed in the callback handler for the first effect, so it will not occur until the first effect is finished:

Click here to view code image

```
$("#img3").click(function(e) {
  $(this).effect("slide",
    { direction:"down", distance:200}, 3000, function(){
        $(this).effect("slide",
            {direction:"right", distance:200}, 3000);
  });
});
```

7. Add the following `click` handler to `#img4`. The `click` handler applies an explode effect, breaking the image into 16 pieces and having them fade as they move apart:

Click here to view code image

```
$("#img4").click(function(e) {
  $(this).effect("explode", {pieces:16}, 3000);
});
```

8. Save all three files and then open the HTML document in a web browser, as shown in Figure 17.2. You should see the effects as you click the images.

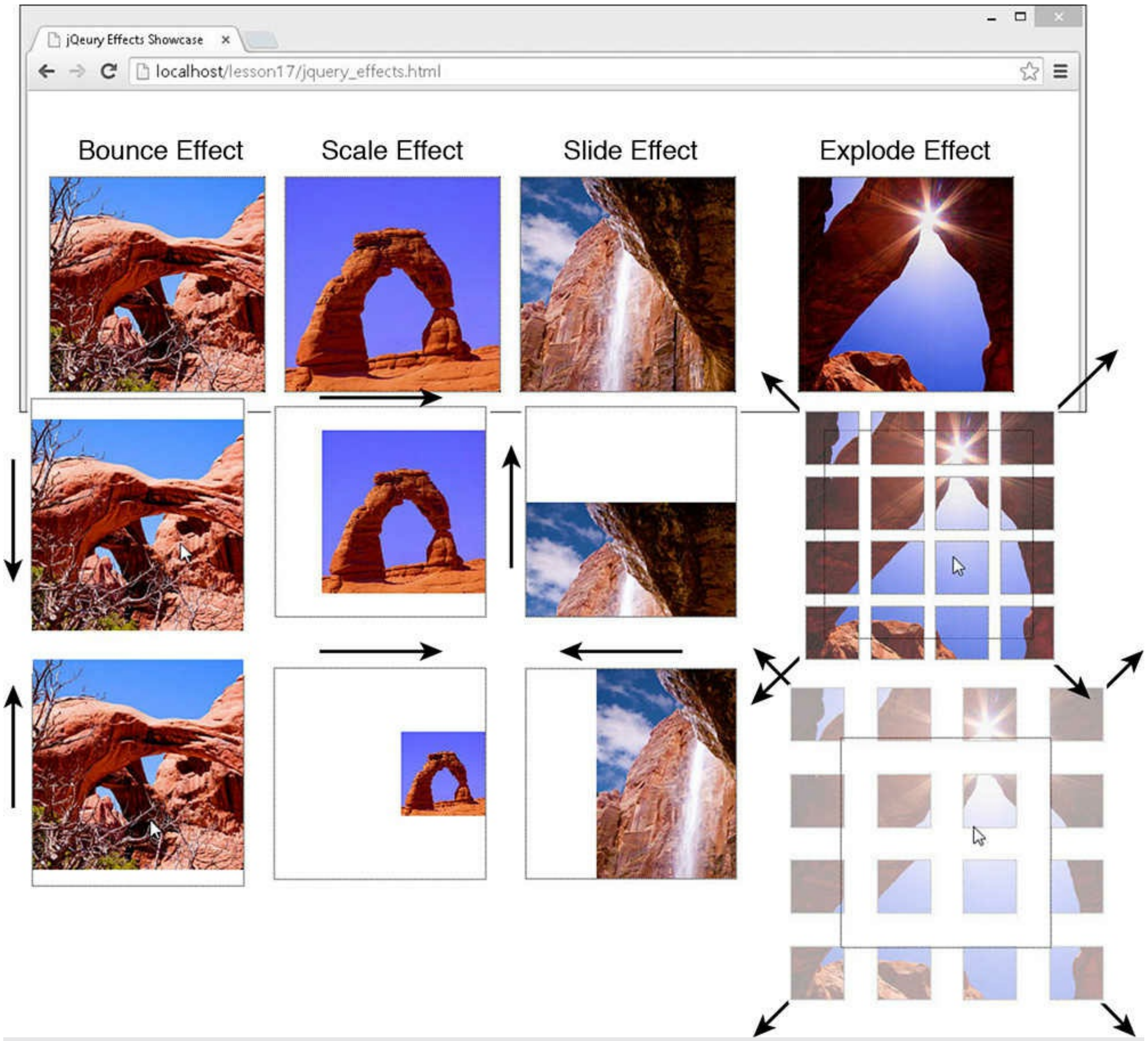


FIGURE 17.2 Applying jQuery UI effects to images.

LISTING 17.1 jquery_effects.html HTML Document That Adds the Web Page

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQeury Effects Showcase</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
```

```
    <script type="text/javascript" src="js/jquery_effects.js"></script>
    <link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css"
href="css/jquery_effects.css">
  </head>
  <body>
      <div id="frame1">
        <img id="img1" src="/images/double.jpg"></div>
      <div id="frame2">
        <img id="img2" src="/images/arch.jpg"></div>
      <div id="frame3">
        <img id="img3" src="/images/cliff.jpg"></div>
      <div id="frame4">
        <img id="img4" src="/images/sunstar.jpg"></div>
  </body>
</html>
```

LISTING 17.2 jquery_effects.js jQuery and jQuery UI That Apply Several Effects on Images

[Click here to view code image]

```
$(document).ready(function(){
  $("#img1").click(function(e) {
    $(this).effect("shake",
        { direction:"down", distance:20, times:5}, 3000);
  });
  $("#img2").click(function(e) {
    $(this).effect("scale",
      { direction:"both", origin:["middle", "right"],
        percent:40, scale:"box", easing:"easeInBounce"}, 3000);
  });
  $("#img3").click(function(e) {
    $(this).effect("slide",
      { direction:"down", distance:200}, 3000, function(){
          $(this).effect("slide",
              {direction:"right", distance:200}, 3000);
    });
  });
  $("#img4").click(function(e) {
    $(this).effect("explode", {pieces:16}, 3000);
  });
});
```

LISTING 17.3 jquery_effects.js CSS Code That Styles the Page

[Click here to view code image]

```
img {
    height:200px; }
div {
    height:200px; width:200px; border:1px dotted;
  display:inline-block; position:fixed; }
#frame1 {
    top:80px; left:20px; }
#frame2 {
    top:80px; left:240px; }
#frame3 {
    top:80px; left:460px; }
#frame4 {
    top:80px; left:720px; }
```

Adding Effects to Class Transitions

A very important part of jQuery UI effects is the capability to animate transitions when applying classes to elements. This is done by adding a duration to the class transition function and specifying the easing function to control the animation effect. Any numerical class values that are changing will be animated each step of the class transition.

Note

Colors can be tricky; jQuery UI is not able to animate the transition from red to blue, but it can animate the transition from #FF0000 to #0000FF. If you want to animate color transitions, use the hex numerical value for them.

The following is a list of the class transition methods that you can use to apply effects on jQuery objects by setting `duration` and `easing` values:

- **.addClass(className [, duration] [, easing] [, complete])**—Adds the class and animates the changes to numerical class properties.
- **.removeClass(className [, duration] [, easing] [, complete]))**—Removes the class and animates the changes to numerical class properties.
- **.switchClass(removeClassName, addClassName [, duration] [, easing] [, complete]))**—First removes the `removeClassName` and animates the changes to numerical class properties, and then adds the `addClassName` animating the numerical class property changes.
- **.toggleClass(className [, switch] [, duration] [, easing] [, complete])**—Adds the class if the object(s) do not already have it or removes it if they do. Any changes to numerical class properties will be animated based on

the easing function.

Try it Yourself: Applying Easing to Class Transitions

It's time to jump in and add some effects to class changes. In this example, you apply animation effects by applying different easing functions to `<span>` elements dressed up as buttons. The purpose of this example is to show each of the class transitions applying easing functions.

The code for the example is in Listings 17.4, 17.5, and 17.6. Use the following steps to create the dynamic web page:

1. In Eclipse, add the lesson17/class_transitions.html, lesson17/js/class_transitions.js, and lesson17/css/class_transitions.css files.
2. Add the code shown in Listing 17.4 and Listing 17.6 to the HTML and CSS files. Notice the different class styles defined in the CSS file. These will be used to animate the class changes.
3. Now open the class_transitions.js file and add the following `.ready()` function that implements a `click` handler for each of the `<span>` elements in the HTML document. The `click` handlers call the `.addClass()`, `.removeClass()`, `.switchClass()`, and `.toggleClass()` methods for the different button elements:

Click here to view code image

```
$(document).ready(function(){
  $("#btn1").click(function(e) {
    $(this).addClass( "round", 2000, "easeInElastic"); });
  $("#btn2").click(function(e) {
    $(this).switchClass( "active", "inactive", 2000,
                         "easeInOutElastic"); });
  $("#btn3").click(function(e) {
    $(this).toggleClass( "round", 2000, "easeOutQuart"); });
  $("#btn4").click(function(e) {
    $(this).removeClass( "round", 2000, "easeInCirc"); });
});
```

4. Save all three files and then open the HTML document in a web browser, as shown in Figure 17.3. You should see the animated class transitions as you click each of the buttons.

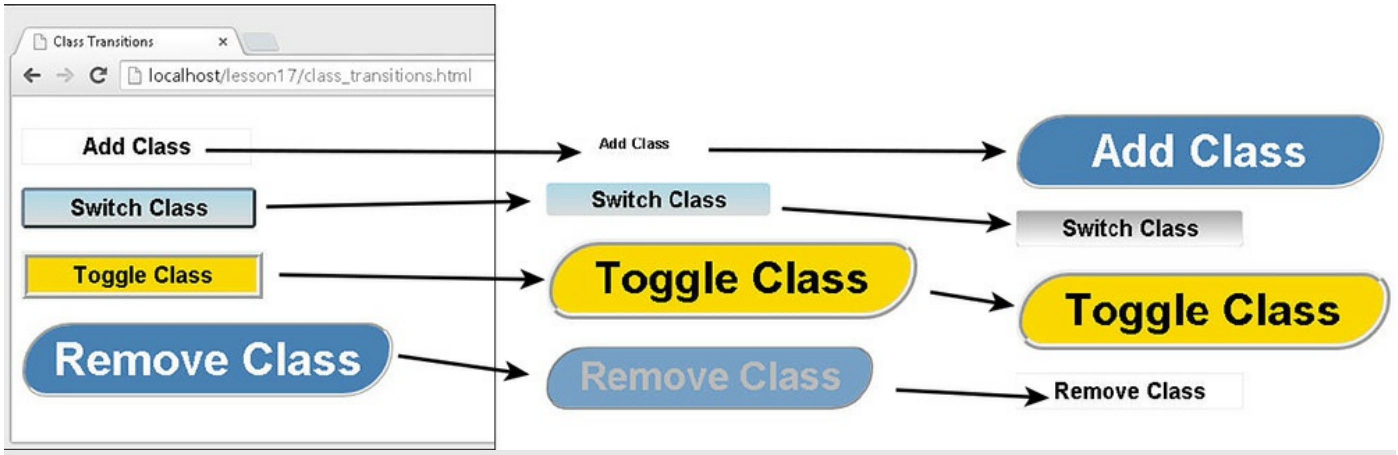


FIGURE 17.3 Adding animated effects to class transitions.

LISTING 17.4 class_transitions.html HTML Document That Adds the Web Page

[Click here to view code image]

```
<!DOCTYPE html>
<html>
  <head>
    <title>Class Transitions</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/class_transitions.js">
</script>
    <link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css"
href="css/class_transitions.css">
  </head>
  <body>
    <span id="btn1">Add Class</span><br>
    <span id="btn2" class="active">Switch Class</span><br>
    <span id="btn3" class="square">Toggle Class</span><br>
    <span id="btn4" class="round">Remove Class</span><br>
  </body>
</html>
```

LISTING 17.5 class_transitions.js jQuery and jQuery UI Code That Implements the Class Transitions with Animation Effects

[Click here to view code image]

```
$(document).ready(function(){
  $("#btn1").click(function(e) {
    $(this).addClass( "round", 2000, "easeInElastic"); });
  $("#btn2").click(function(e) {
    $(this).switchClass( "active", "inactive", 2000,
                         "easeInOutElastic"); });
  $("#btn3").click(function(e) {
    $(this).toggleClass( "round", 2000, "easeOutQuart"); });
  $("#btn4").click(function(e) {
    $(this).removeClass( "round", 2000, "easeInCirc"); });
});
```

LISTING 17.6 class_transitions.css CSS Code That Styles the Page

Click here to view code image

```
span {
    display:inline-block; height:30px; width:200px;
    margin-top:20px; border:1px ridge;
  text-align:center; font:bold 20px/30px arial; }
.round {
    border-width:3px; border-radius:60px 30px;
    height:60px; width:320px; line-height:60px;
  background-color:steelblue; color:white; font-size:40px; }
.square {
    border-width:6px; background-color:gold; color:black; }
.active {
    background-image: -moz-linear-gradient(top , lightblue, gainsboro);
  background-image: -webkit-linear-gradient(top , lightblue,
gainsboro);
  background-image: -ms-linear-gradient(top , lightblue, gainsboro);
  border-radius:4px 4px; border:3px outset slategrey; }
.inactive {
    background-image: -moz-linear-gradient(top , darkgrey, white);
  background-image: -webkit-linear-gradient(top , darkgrey, white);
  background-image: -ms-linear-gradient(top , darkgrey, white);
  border-radius:4px; color:#steelblue; }
```

Adding Effects to Element Visibility Transitions

Another very cool effect that you can add to your web pages with jQuery UI is visibility changes. This can be one of the most useful in allowing users to visualize what is happening, and it provides them with a chance to follow the page flow better.

Visibility effects are applied in the same manner as the `.effect()` function you learned earlier in this lesson. You specify an effect from Table 17.1 and then set the desired options, including an `easing` function if you want to control the animation.

The following is a list of the different element visibility transition methods that you can add effects to using jQuery UI:

- **.hide(effect [, options] [, duration] [, complete])**—Applies the effect with options while hiding the element.
- **.show(effect [, options] [, duration] [, complete])**—Applies the effect with options while showing the element.
- **.toggle(effect [, options] [, duration] [, complete])**—Either shows or hides the object based on its current visibility and applies the specified effect while doing so.

Try it Yourself: Applying Effects to jQuery Visibility Transitions

In this example, you apply effects to the visibility of menu items. The purpose of the example is to familiarize you with how to implement jQuery UI effects in jQuery’s visibility methods.

The code for the example is in Listings 17.7, 17.8, and 17.9. Use the following steps to create the dynamic web page:

1. In Eclipse, add the lesson17/visibility_transitions.html, lesson17/js/visibility_transitions.js, and lesson17/css/visibility_transitions.css files.
2. Add the code shown in Listing 17.7 and Listing 17.9 to the HTML and CSS files.
3. Open the visibility_transitions.js file and add the following `.ready()` function that will hide the secondary menus initially:

Click here to view code image

```
$(document).ready(function(){
  $("#showMenu, #showMenu2, #toggleMenu").hide();
...
})
```

4. Add the following click handlers for the different menu items. Notice that you use several, including `fold`, `scale`, `explode`, and `blind`. The reason is so you can see how the effects work. On the `blind` effect, you set `easing` to `easeOutBounce`; this provides a simple `bounce` effect, as if the menu bounces at the bottom:

Click here to view code image

```
  $("#show").click(function(e) {
      $("#showMenu").show("fold", {size:22}, 2000); });
  $("#show2").click(function(e) {
      $("#showMenu2").show("scale",
          {origin:["top","left"]}, 2000); });
  $("#showMenu").click(function(e) {
      $("#showMenu").hide("fold", {size:22}, 2000); });
```

```
$("#showMenu2").click(function(e) {
    $("#showMenu2").hide("explode", {pieces:9}, 2000); });
$("#toggle, #toggleMenu").click(function(e) {
    $("#toggleMenu").toggle("blind",
        {direction:"up", easing:"easeOutBounce"}, 1000); });
```

5. Save all three files and then open the HTML document in a web browser, as shown in Figure 17.4. You should be able to select the menus and see the `.show()`, `.hide()`, and `.toggle()` effects working.

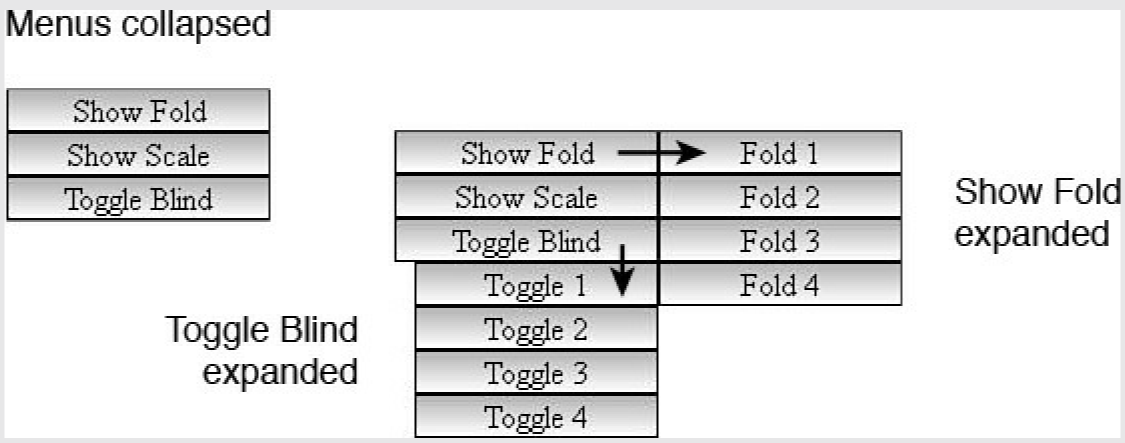


FIGURE 17.4 Using jQuery UI effects to improve showing and hiding menu options.

LISTING 17.7 visibility_transitions.html HTML Document That Adds the Web Page

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Visibility Transitions</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/visibility_transitions.js">
</script>
    <link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css"
href="css/visibility_transitions.css">
  </head>
  <body>
    <span id="show">Show Fold</span><br>
    <span id="show2">Show Scale</span><br>
    <span id="toggle">Toggle Blind</span><br>
    <div id="showMenu">
      <span>Fold 1</span><br><span>Fold 2</span><br>
      <span>Fold 3</span><br><span>Fold 4</span><br>
    </div>
    <div id="showMenu2">
      <span>Explode 1</span><br><span>Explode 2</span><br>
      <span>Explode 3</span><br><span>Explode 4</span><br>
    </div>
    <div id="toggleMenu">
      <span>Toggle 1</span><br><span>Toggle 2</span><br>
      <span>Toggle 3</span><br><span>Toggle 4</span><br>
    </div>
  </body>
</html>
```

LISTING 17.8 visibility_transitions.js jQuery and jQuery UI That Implements the Visibility and Effects

Click here to view code image

```
$(document).ready(function(){
  $("#showMenu, #showMenu2, #toggleMenu").hide();
  $("#show").click(function(e) {
      $("#showMenu").show("fold", {size:22}, 2000); });
  $("#show2").click(function(e) {
      $("#showMenu2").show("scale",
          {origin:["top","left"]}, 2000); });
  $("#showMenu").click(function(e) {
      $("#showMenu").hide("fold", {size:22}, 2000); });
  $("#showMenu2").click(function(e) {
      $("#showMenu2").hide("explode", {pieces:9}, 2000); });
  $("#toggle, #toggleMenu").click(function(e) {
      $("#toggleMenu").toggle("blind",
          {direction:"up", easing:"easeOutBounce"}, 1000); });
});
```

LISTING 17.9 visibility_transitions.css CSS Code That Styles the Page

Click here to view code image

```
span {
  display:inline-block; width:130px; border:1px ridge;
  text-align:center; cursor:pointer;
  background-image: -moz-linear-gradient(top , lightblue, white);
  background-image: -webkit-linear-gradient(top , lightgray, white);
  background-image: -ms-linear-gradient(top , lightgray, white); }
div span{
  width:120px; margin-left:10px; }
#showMenu {
  position:fixed; left:130px; top:8px; }
#showMenu2 {
```

```
position:fixed; left:130px; top:30px; }
```

Try it Yourself: Adding Effects to Animations

The new easing functionality in jQuery UI can also be applied to the jQuery `.animation()` method. Using the different `easing`, you can alter the effect of the animation through varying the rate that the transition occurs. One of the best examples of this is to apply a bounce transition to an animation that alters the position of an element.

The best way to illustrate this is through an example. In this example, you apply several animation effects to the movement of an image. The image is a simple ball that moves around the screen when clicked. At the final position, the ball hits a needle and "pops" using an `explode` effect. The purpose of the example is for you to see how the effects apply to the animation process.

The code for the example is in Listings 17.10, 17.11, and 17.12. Use the following steps to create the dynamic web page:

1. In Eclipse, add the lesson17/animation_effects.html, lesson17/js/animation_effects.js, and lesson17/css/animation_effects.css files.
2. Add the code shown in Listing 17.10 and Listing 17.12 to the HTML and CSS files. The `<img>` elements are set to `fixed` positioning so their movement can be animated by changing the CSS position.
3. Open the animation_effects.js file and add the `coords` array shown at the top of Listing 17.11. This array provides positioning coordinates and easing function names that will be used by the `click` handler.
4. The following `.ready()` function adds the `click` handler to the ball:

Click here to view code image

```
$(document).ready(function(){
  $("#ball").click(reposition);
});
```

5. Add the following `click` handler `reposition()` code. This function will pop off a coordinate and use the values in an `.animate()` call that will animate moving the ball. The easing value is set using the `easing` attribute of the `coord` object. Notice that the `callback` handler loops back to the `reposition()` function. When there are no more coordinates left, an explode effect is applied to the ball:

Click here to view code image

```
function reposition(){
  if (coords.length){
    coord = coords.pop();
    $(this).animate(coord, 1000, coord.easing, reposition);
  } else{
    $("#ball").effect("explode", {pieces:100}, 2000); }
}
```

6. Save all three files and then open the HTML document in a web browser, as shown in Figure 17.5. When you click the ball, it should move around the web page with varying speed, illustrating the easing functions. At the end, it should hit the needle and disappear.

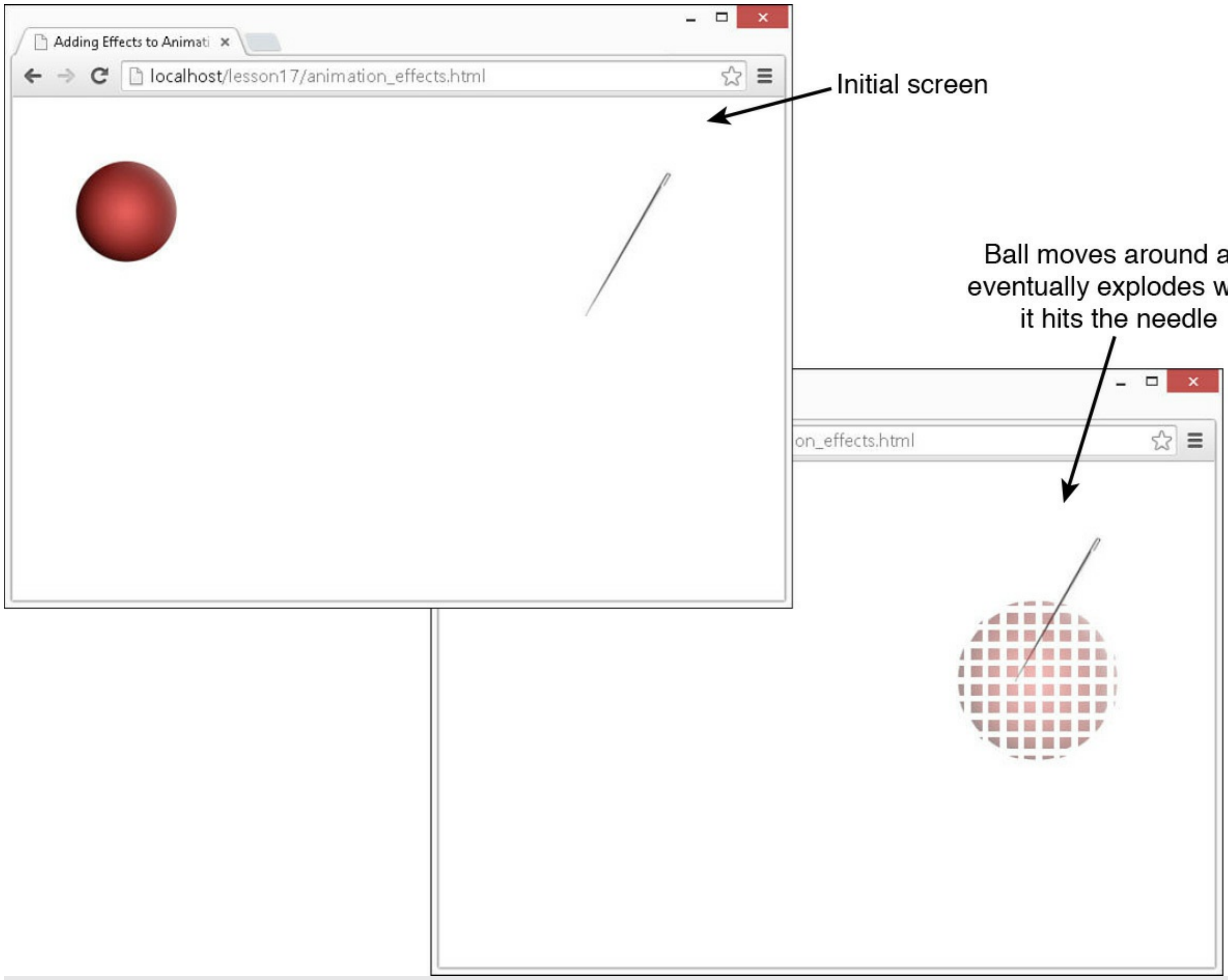

Adding Effects to Animati
localhost/lesson17/animation_effects.html
Initial screen
Ball moves around and eventually explodes when it hits the needle
on_effects.html

FIGURE 17.5 Using jQuery UI effects to adjust the timing of position animation.

LISTING 17.10 animation_effects.html HTML Document That Adds the Web Page

[Click here to view code image]()

```
01 <!DOCTYPE html>
02 <html>
03   <head>
04     <title>Adding Effects to Animations</title>
05     <meta charset="utf-8" />
06     <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
07     <script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
08     <script type="text/javascript" src="js/animation_effects.js">
</script>
09     <link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
10     <link rel="stylesheet" type="text/css"
```

```
href="css/animation_effects.css">
  </head>
  <body>
      <img id="needle" src="/images/needle.jpg" />
    <img id="ball" src="/images/ball.png" />
  </body>
</html>
```

LISTING 17.11 animation_effects.js jQuery and jQuery UI That Implements the Reposition Effects

[Click here to view code image]()

```
var coords = [{top:140, left:470, easing:"easeInBounce"},
              {top:100, left:200, easing:"easeOutElastic"},
              {top:300, left:200, easing:"easeInOutCirc"},
              {top:20, left:300, easing:"easeInBounce"},
              {top:10, left:10, easing:"easeOutExpo"},
              {top:200, left:100, easing:"easeInSine"}]
function reposition(){
  if (coords.length){
    coord = coords.pop();
    $(this).animate(coord, 1000, coord.easing, reposition);
  } else{
    $("#ball").effect("explode", {pieces:100}, 2000); }
}
$(document).ready(function(){
  $("#ball").click(reposition);
});
```

LISTING 17.12 animation_effects.css CSS Code That Styles the Page

[Click here to view code image]()

```
img {
  position:fixed; z-index: -1;}
#needle {
  left:450px; top:50px; width: 150px; }
#ball {
  left:50px; top:50px; width: 100px; }
```

Summary

jQuery UI effects are basically animations to the CSS properties of page elements. The benefit that they provide is that rather than having the effect happen instantaneously, you can have it happen gradually. Using easing functions, you can adjust the rate that the changes occur in the animation to give elements more of an interactive feel.

Q&A

Q. Is there a way to animate changing an `<img>` element from one source to another so that part of both elements are visible at the same time?

A. Not directly, but there is a trick you can employ. Use two `<img>` elements and animate the `opacity` property changes at the same time. As one disappears, the other one will become visible.

Q. Is there a way to create custom easing functions?

A. Yes, you can create a custom easing function and attach it to `$.easing`. The function needs to accept the following parameters and return a new value based on those parameters:

- **tPercent**—Percentage of time passed in the animation from 0.0 to 1.0.
- **tMS**—Milliseconds since animation started.
- **startValue**—Starting value of the property.
- **endValue**—Ending value of the property.
- **tTotal**—Duration of the animation.

Click here to view code image

```
    $.easing.myCustom = function(tPercent, tMS, startValue, endValue,
tTotal) {
        var newValue= <your code here>>...
       return newValue;
    }
```

Workshop

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

Quiz

1. How do you control the amount of time the effect will take?
2. How do you define the number of pieces the explode effect will generate to 25?
3. True or false: You cannot animate changes to the `border-style`.
4. What effect would you use to simulate an element shrinking?

Quiz Answers

1. Setting the `duration` value.
2. Set the options value to `{pieces:25}`.
3. True. You can animate only numerical changes.
4. Scale or Size.

Exercises

1. Open the code in Listings 17.1, 17.2, and 17.3 and change which effects are applied to the images. Try applying the `pulsate`, `drop`, and `puff` effects.
2. Modify the code in Listings 17.10, 17.11, and 17.12. Add a duration attribute to each of the coordinates so that you can also adjust the `duration` time for each point in the ball's animation. Add a few new points as well. The coordinate values should look something like this:

Click here to view code image

```
{top:20, left:300, easing:"easeInBounce", duration:1500},
```

Lesson 18. Advanced Interactions Using jQuery UI Interaction Widgets

What You'll Learn in This Lesson:

- Implementing drag-and-drop functionality
- Making elements and children resizable
- How to select multiple pages elements using a bounding box
- Creating sortable tables, lists, and containers

A special set of jQuery UI widgets are intended to provide generalized interactions for various elements. These widgets allow you to make elements draggable and droppable and provide sorting, box selection, and resize functionality to elements.

The jQuery UI interaction widgets can be attached to elements to provide a rich set of predefined interactions. The following sections cover the different interaction widgets and how to implement them in your web pages.

Introducing jQuery UI Interactions

All jQuery UI interactions are based on two main components—the `jQuery.widget` factory and the mouse widget. The `jQuery.widget` factory provides the base functionality for all widgets, including creation, disabling, enabling, and option settings. The mouse widget provides the base mouse interactions with the widget that captures mouse events and allows the widgets to interact with them.

Reviewing the jQuery.widget Factory

The `jQuery.widget` factory defines an interface that is used by all jQuery UI widgets. The options, methods, and events of the factory are available to all widgets. Table 18.1 lists the methods and events defined in the factory and available in all widgets.

Method	Description
`create`	Event triggered each time the widget is created.
`destroy()`	Removes the widget functionality completely.
`disable()`	Keeps the widget functionality, but disables it.
`enable()`	Enables the widget functionality.
`option([optionName] [,value])`	`option()` returns an object with all option keys/values. `option(optionName)` returns the specific option value. `option(optionName, value)` sets an option value.
`widget(name, [,base], prototype)`	Used to create custom widgets. Provides three parameters: `name` is the string used name to access the widget, `base` is the existing widget to inherit functionality from, and `prototype` is an object defining the widget.

TABLE 18.1 Methods and Events Available on All jQuery UI Widgets

You can get more information about the `jQuery.widget` factory at http://api.jqueryui.com/jQuery.widget/.

Understanding the Mouse Interaction Widget

The mouse interaction widget is automatically applied to all widgets. Typically, you will not need to interact with it much. However, it does expose a few options that are very useful at times. Those options are the following:

- **cancel**—Cancels interaction for specific elements. For example, to cancel mouse interactions for elements with `class="label"` in the `#item1`, you use the following:

Click here to view code image

```
$( "#item1" ).mouse( "option", "cancel", ".label" );
```

- **delay**—Delays the time after the `mousedown` event occurs before the interaction takes place. For example, to add a 1-second delay for mouse interactions on `#item2`, use the following:

Click here to view code image

```
$( "#item2" ).mouse( "option", "delay", 1000 );
```

- **distance**—Specifies the distance in pixels the mouse must travel after the mousedown event occurs before the interaction should start. For example, to set the distance to 10 pixels for mouse interactions on `#item3`, use the following:

Click here to view code image

```
    $( "#item3" ).mouse( "option", "distance", 10 );
```

Using the Drag-and-Drop Widgets

Now that you have reviewed the widget interface and the mouse interaction widget, you are ready to look at some of most common jQuery UI widgets—the draggable and droppable widgets. These widgets are designed to work in tandem.

You can define one element to be draggable and then another to be droppable. When draggable elements are dropped on droppable widgets, you can apply JavaScript and jQuery code to provide whatever interaction for the user you would like.

Dragging Elements with the Draggable Widget

The draggable widget defines an element as draggable by holding down the mouse and moving it. This allows you to move the element to whatever position on the screen you would like.

The draggable widget will handle scrolling elements and provides several options to control the look and feel while dragging. Table 18.2 describes the more common draggable options. The following shows an example of attaching the `draggable` widget to an element with the `cursor` and `opacity` options:

Click here to view code image

```
  $("#img1").draggable({cursor:"move", opacity:.5});
```

Option	Description
`axis`	Can be set to `x` or `y`, or `false`. `x` drags horizontally only, `y` drags vertically only, and `false` drags freely.
`containment`	Specifies a container to limit dragging within. Possible values are `"parent"`, `"document"`, or `"window"`.
`cursor`	Specifies the cursor to display while dragging.
`helper`	Defines what element is displayed when dragging. Values can be `"original"`, `"clone"`, or a function that returns a DOM object.
`opacity`	Sets the opacity while dragging.
`revert`	Boolean. Specifies if the `"original"` object should return to its original position when dragging stops.
	String. If set to `"valid"`, revert occurs only if the object has been dropped successfully. `"invalid"` reverts only if the object hasn't been dropped successfully.
`stack`	Is set to `false` or a selector. If a selector is specified, the item is brought to the top `z-index` of the element specified by the selector.
`zIndex`	`z-index` value to use while dragging.

TABLE 18.2 Common Draggable Widget Options

The draggable widget also provides the additional events so handlers can be attached to the element when dragging starts, is in progress, and stops. Table 18.3 lists the events that you can access on draggable items. The following shows an example of adding a `dragstop` event to apply a `bounce` effect when the item is dropped:

Click here to view code image

```
$("#drag1").draggable({cursor:"move", opacity:.5});
$("#drag1").on("dragstop", function(){$(this).effect("bounce", 1000); });
```

Event	Description
`drag(event, ui)`	Triggered while dragging. `event` is the JavaScript event object. `ui` is an object with the following values: ▶ **`helper`**—jQuery object representing the helper for the draggable item. ▶ **`position`**—`{top, left}` object for the current CSS position. ▶ **`offset`**—`{top, left}` object for the current CSS offset.
`dragstart(event, ui)`	Triggered when dragging starts.
`dragstop(event, ui)`	Triggered when dragging stops.

TABLE 18.3 Draggable Widget Events

Try it Yourself: Adding Draggable Images to a Web Page

In this example, you implement draggable multiple image elements. Each image behaves a bit differently, as described in the following steps. The purpose of the example is to help you see how easy it is to make web elements draggable.

The code for the example is in Listings 18.1, 18.2, and 18.3. Use the following steps to create the dynamic web page:

1. In Eclipse, create the lesson18, lesson18/js, and lesson18/css folders, and then add the lesson18/draggable_images.html, lesson18/js/draggable_images.js, and lesson18/css/draggable_images.css files.

2. Add the code shown in Listing 18.1 and Listing 18.3 to the HTML and CSS files.

3. Open the draggable_images.js file and add a `.load()` function.

4. Add the following lines. Line 2 adds the `draggable` widget to `#drag1` and sets the cursor to move while dragging; also, the `opacity` is at 50% while dragging. Line 3 adds a `dragstop` hander function that applies the `bounce` effect to the image when it is dropped:

Click here to view code image

```
02   $("#drag1").draggable({cursor:"move", opacity:.5});
03   $("#drag1").on("dragstop", function(){
04     $(this).effect("bounce", 1000); });
```

5. Add the following lines that implement `draggable` on the `#drag2` element.

The helper option is set to "clone" so that the object stays in place while dragging; a dragstop event handler is added to animate changing the position from the original to the location of the helper clone. Notice that the offset is collected using the ui parameter:

Click here to view code image

```
$("#drag2").draggable({helper:"clone"});
$("#drag2").on("dragstop", function(e, ui){
  $("#drag2").animate(ui.offset); });
```

6. Add the following lines that implement draggable on the #drag3 element. This time, you implement a drag handler that updates a paragraph element with the current mouse coordinates while dragging. The dragstop handler will clear out the position text:

Click here to view code image

```
$("#drag3").draggable();
$("#drag3").on("drag", function(e){
  $(this).children("p").html(e.pageX+", "+e.pageY); });
$("#drag3").on("dragstop", function(e){
  $(this).children("p").html(""); });
```

7. Save all three files and then open the HTML document in a web browser, as shown in Figure 18.1. You should be able to drag the images around and test the interactions.

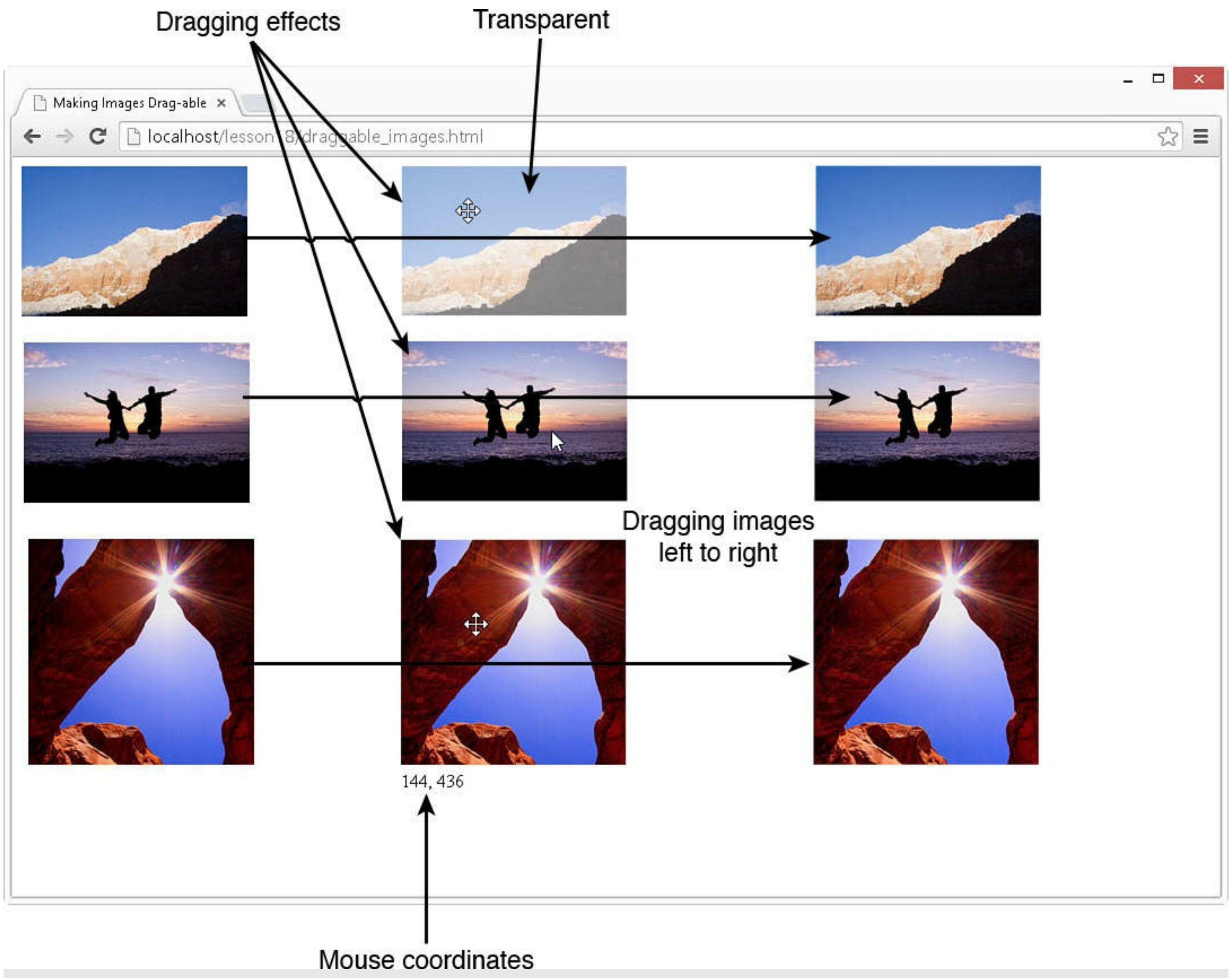


FIGURE 18.1 Applying jQuery UI dragging widgets to move images on the screen.

LISTING 18.1 draggable_images.html HTML Document That Adds the Web Page

[Click here to view code image](#)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Making Images Drag-able</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/draggable_images.js">
</script>
    <link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css"
```

```
href="css/draggable_images.css">
  </head>
  <body>
      <div id="drag1">
      <img id="img1" src="/images/snowy_cliff.jpg"></div>
      <div id="drag2">
      <img id="img1" src="/images/shadow_jump.jpg"></div>
      <div id="drag3">
      <img id="img1" src="/images/sunstar.jpg"></div>
  </body>
</html>
```

LISTING 18.2 draggable_images.js jQuery and jQuery UI Implements Draggable Images

Click here to view code image

```
$(window).load(function(){
  $("#drag1").draggable({cursor:"move", opacity:.5});
  $("#drag1").on("dragstop", function(){
    $(this).effect("bounce", 1000); });
  $("#drag2").draggable({helper:"clone"});
  $("#drag2").on("dragstop", function(e, ui){
    $("#drag2").animate(ui.offset); });
  $("#drag3").draggable();
  $("#drag3").on("drag", function(e){
    $(this).children("p").html(e.pageX+", "+e.pageY); });
  $("#drag3").on("dragstop", function(e){
    $(this).children("p").html(""); });
});
```

LISTING 18.3 draggable_images.css CSS Code That Styles the Page

Click here to view code image

```
p {
    margin:0px; }
div {
    height:80px; width:100px; position:fixed; }
#drag2 {
    top:100px; }
#drag3 {
    top:200px; }
img {
    width:200px; }
img:hover{
    cursor:move; }
```

Creating Drop Targets with the Droppable Widget

The droppable widget defines an element as a valid drop container usable by draggable items. This enables you to provide interactions between elements using simple mouse controls.

The droppable widget allows you to specify an accept function that can process the information about the event, such as mouse coordinates as well as the draggable item involved. Table 18.4 describes the more common droppable options. The following shows an example of attaching the `droppable` widget to an element and specifying the `tolerance` level:

Click here to view code image

```
$("#div1"). droppable ({tolerance:"touch"});
```

Option	Description
`accept`	Specifies a selector used to filter the elements that will be accepted by the droppable item.
`activeClass`	Specifies a class that will be applied to the droppable item while a valid draggable item is being dragged.
`greedy`	Boolean. The default is `false`, meaning that all valid parent droppable items will receive the draggable item as well. When `true`, only the first droppable item will receive the draggable item.
`hoverClass`	Specifies a class that will be applied to the droppable item while a valid draggable item is hovering over it.
`tolerance`	Specifies the method used to determine if a draggable item is valid. Acceptable values are the following: ▶ **`fit`**—Draggable overlaps droppable entirely. ▶ **`intersect`**—Draggable overlaps droppable at least 50% in both directions. ▶ **`pointer`**—Mouse hovers over droppable. ▶ **`touch`**—Draggable overlaps droppable in any location.

TABLE 18.4 Common Droppable Widget Options

The droppable widget also provides the additional events so handlers can be attached to the element when dragging and dropping. Table 18.5 lists the events that you can access on droppable items. The following shows an example of adding a `dropactivate` event to apply a `shake` effect when a droppable item is activated by a drag start:

Click here to view code image

```
$("#drop1").droppable({tollerance:"pointer"});
```

```
$("# drop1").on("dropactivate", function(){$(this).effect("shake", 1000);
});
```

Event	Description
`dropactivate(event, ui)`	Triggered when a valid draggable item begins dragging. `event` is the JavaScript event object. `ui` is an object with the following values: ▶ **`draggable`**—jQuery object representing the actual draggable item. ▶ **`helper`**—jQuery object representing the helper for the draggable item. ▶ **`position`**—`{top, left}` object for the current draggable CSS position. ▶ **`offset`**—`{top, left}` object for the current draggable CSS offset.
`drop(event, ui)`	Triggered when a draggable item is dropped on droppable.
`dropout(event, ui)`	Trigger when draggable leaves droppable based on tolerance.
`dropover(event, ui)`	Trigger when draggable enters droppable based on tolerance.

TABLE 18.5 Droppable Widget Events

Try it Yourself: Applying Drag and Drop to a Web Page

In this example, you implement draggable and droppable on page elements. The first droppable element displays an image, and the second adds the image and `src` text to a list. The purpose of the example is to help you see how easy it is to make web elements droppable.

The code for the example is in Listings 18.4, 18.5, and 18.6. Use the following steps to create the dynamic web page:

1. In Eclipse, add the lesson18/drag_n_drop.html, lesson18/js/drag_n_drop.js, and lesson18/css/drag_n_drop.css files.
2. Add the code shown in Listing 18.4 and Listing 18.6 to the HTML and CSS files.
3. Open the drag_n_drop.js file and add a `.ready()` function.
4. Add the following lines that add the `draggable` widget to `#drag1`, `#drag2`, and `#drag3`. Use `clone` for the `helper` setting to keep the images in place, set the `cursor` and the `opacity`. Also, you set up `zIndex` so that the images will show on top of other page elements, even while dragging over them:

Click here to view code image

```
02   $("#drag1, #drag2, #drag3").draggable(
03     {helper:"clone", cursor:"move", opacity:.7, zIndex:99});
```

5. Add the following lines that implement droppable on `#drop1`. `accept` is set to `"img"` so that only `<img>` elements will be accepted. In line 6, a `dropover` event handler is added that applies a `pulsate` effect to the droppable box when the draggable item is hovering over it. Also, a `drop` event handler is added in line 8 that will add an `<img>` element to `#drop1` with the same `src` attribute as the draggable element. A `bounce` effect is also added to show the user the content changed:

Click here to view code image

```
04   $("#drop1").droppable(
05       { accept:"img", tolerance:"fit"});
06   $("#drop1").on("dropover", function(e,ui){
07     $(this).effect("pulsate"); });
08   $("#drop1").on("drop", function(e,ui){
09     $(this).html($("<img></img>").attr("src",
10         ui.draggable.attr("src")));
11     $(this).effect("bounce");
12   });
```

6. Add the following lines that implement droppable on `#drop2`. Notice that in line 15, a `hoverClass` is added. The class will cause the background to turn light blue when the box is hovered over by a droppable item. Also in line 16, the `drop` handler function is implemented that adds a `<div>` element with the `<img>` and `src` text to `#drop2`:

Click here to view code image

```
13   $("#drop2").droppable(
14       { accept:"img", tolerance:"intersect",
15         hoverClass:"drop-hover"});
16   $("#drop2").on("drop", function(e,ui){
17     var item = $("<div></div>");
18     item.append($("<img></img>").attr("src",
19         ui.draggable.attr("src")));
20     item.append($("<span></span>").html(
21         ui.draggable.attr("src")));
22     $(this).append(item);
23   });
```

7. Save all three files and then open the HTML document in a web browser, as shown in Figure 18.2. You should be able to drag and drop the images around and test the interactions.

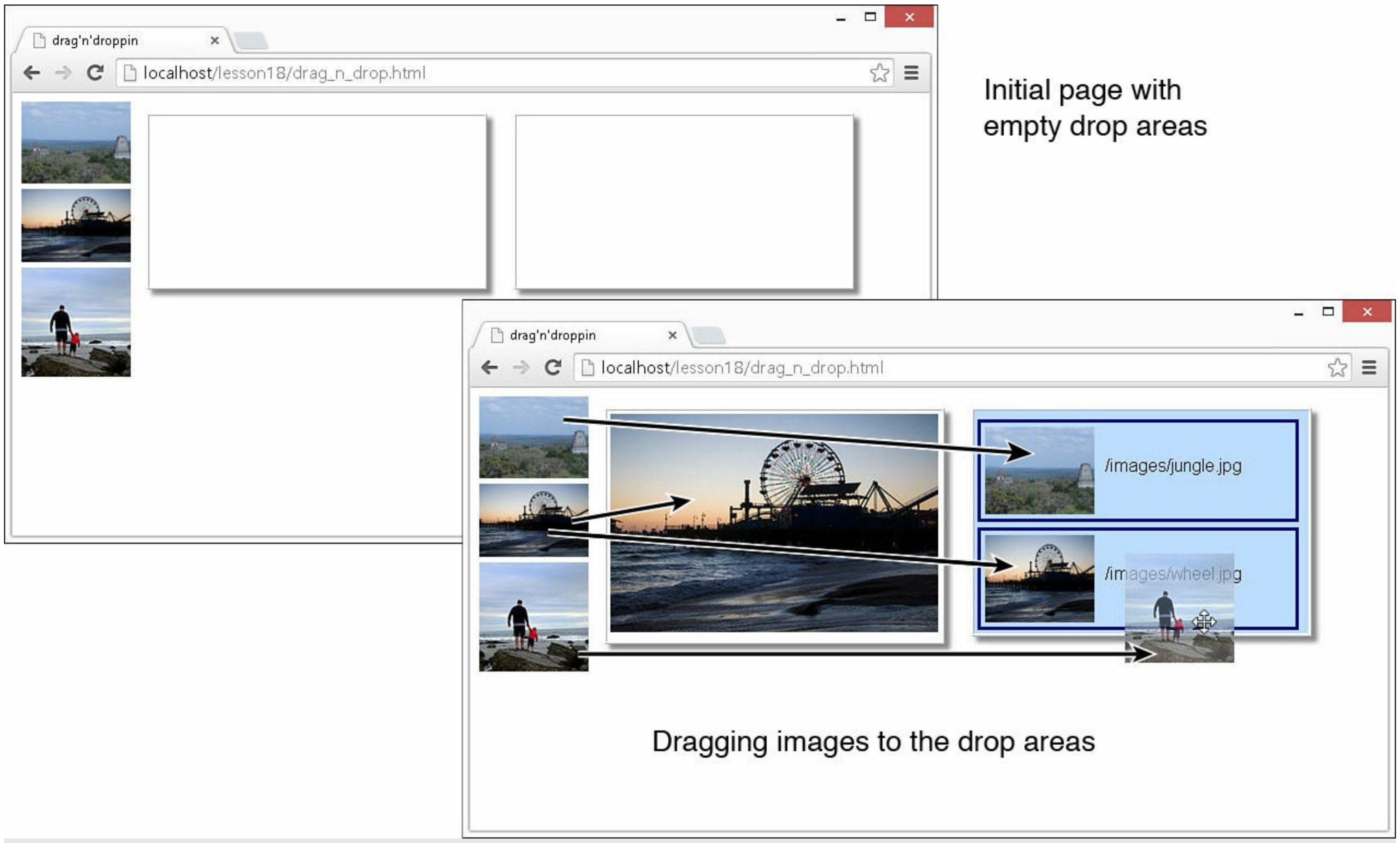

drag'n'droppin
localhost/lesson18/drag_n_drop.html
Initial page with empty drop areas
/images/jungle.jpg
/images/wheel.jpg
Dragging images to the drop areas

FIGURE 18.2 Applying jQuery UI drag and drop.

LISTING 18.4 drag_n_drop.html HTML Document That Adds the Web Page

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>drag'n'droppin</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/drag_n_drop.js"></script>
    <link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/drag_n_drop.css">
  </head>
  <body>
    <div id="images">
      <img id="drag1" src="/images/jungle.jpg" />
      <img id="drag2" src="/images/wheel.jpg" />
      <img id="drag3" src="/images/img7.jpg" />
    </div>
    <div id="drop1"></div>
    <div id="drop2"></div>
  </body>
</html>
```

LISTING 18.5 drag_n_drop.js jQuery and jQuery UI Implements Draggable and Droppable Elements

Click here to view code image

```
$(document).ready(function(){
  $("#drag1, #drag2, #drag3").draggable(
    { helper:"clone", cursor:"move", opacity:.7, zIndex:99 });
  $("#drop1").droppable(
      { accept:"img", tolerance:"fit"});
  $("#drop1").on("dropover", function(e,ui){
    $(this).effect("pulsate"); });
  $("#drop1").on("drop", function(e,ui){
    $(this).html($("<img></img>").attr("src",
        ui.draggable.attr("src")));
    $(this).effect("bounce");
  });
  $("#drop2").droppable(
      { accept:"img", tolerance:"intersect",
        hoverClass:"drop-hover"});
  $("#drop2").on("drop", function(e,ui){
    var item = $("<div></div>");
    item.append($("<img></img>").attr("src",
        ui.draggable.attr("src")));
    item.append($("<span></span>").html(
        ui.draggable.attr("src")));
    $(this).append(item);
  });
});
```

LISTING 18.6 drag_n_drop.js CSS Code That Styles the Page

Click here to view code image

```
div {
    display:inline-block; vertical-align:top; }
img {
    width:100px; margin:0px; }
#images {
    width:100px; height:300px; }
#drop1, #drop2 {
    width:300px; min-height:150px; padding:3px; margin:10px;
  border:3px ridge white; box-shadow: 5px 5px 5px #888888; }
#drop1 img {
    width:300px; }
#drop2 div{
    height:80px; width:280px; padding:4px;
    border:3px ridge darkblue; margin-top:5px; }
#drop2 div img {
    height:80px; margin-right:10px; }
#drop2 div span {
    display:inline-block; vertical-align:top;
    font:16px/70px arial; }
.drop-hover {
    background-color:#BBDDFF; }
```

Resizing Elements Using the Resizable Widget

A frequent request for users is the capability to define the size and shape of images, lists, tables, and so on. The resizable widget provides the capability to easily resize an image with mouse controls. This allows users to resize page elements as they desire.

The resizable widget attaches several handle controls to the page elements that interact with the mouse to resize the elements. You can also resize other elements at the same time.

Table 18.6 describes the more common resizable options. The following shows an example of attaching the `resizable` widget to an element and specifying the `aspectRatio` as `true`:

Click here to view code image

```
$("#div1"). resizable ({aspectRatio:true});
```

Option	Description
`alsoResize`	Specifies a selector, jQuery object, or DOM object to resize synchronously with the currently sizable element.
`aspectRatio`	Can be set to true to maintain the current aspect ratio or a number to force a specific aspect ratio.
`autoHide`	Boolean. When set to true, the handles will disappear when not hovered over.
`containment`	Limits the resizing bounds to an object. Values can be a selector, DOM object, or a string containing `"parent"` or `"document"`.
`ghost`	Boolean. When `true`, a semitransparent element is shown during resizing.
`handles`	Specifies a comma-separated list of compass style locations to place the image. Values can be the following, for example: {handles:"n,e,s,w,ne,se,sw,nw"}
`helper`	Same as for draggable.

TABLE 18.6 Common Resizable Widget Options

The resizable widget also provides the additional events so handlers can be attached to the element when resizing. Table 18.7 lists the events that you can access on resizable items. The following shows an example of adding a `resizestop` event to apply a `pulsate` effect when a resizable item has finished being resized:

Click here to view code image

```
$("#resize1"). resizable ({aspectRatio:true });
$("#resize1").on("dropactivate", function(){$(this).effect("pulsate"); });
```

Event	Description
`resize(event, ui)`	Triggered while resizing. `event` is the JavaScript event object. `ui` is an object with the following values: ▶ **`element`**—jQuery object representing the element to be resized. ▶ **`originalElement`**—jQuery object representing the original element before being wrapped. ▶ **`helper`**—jQuery object representing the helper for the draggable item. ▶ **`originalPosition`**—`{top, left}` object for the original position. ▶ **`originalSize`**—`{width, height}` object for original size. ▶ **`position`**—`{top, left}` object for the current position. ▶ **`size`**—`{width, height}` object for the current offset.
`resizestart(event, ui)`	Triggered when a resizing starts.
`resizestop(event, ui)`	Triggered when a resizing stops (mouse up).

TABLE 18.7 Resizable Widget Events

Try it Yourself: Creating Resizable Elements

In this example, you implement draggable and resizable elements to allow users to customize the position and size of items on the web page. The purpose of the example is to illustrate the interaction between draggable and resizable.

The code for the example is in Listings 18.7, 18.8, and 18.9. Use the following steps to create the dynamic web page:

1. In Eclipse, add the lesson18/resizable_elements.html, lesson18/js/resizable_elements.js, and lesson18/css/resizable_elements.css files.

2. Add the code shown in Listing 18.7 and Listing 18.9 to the HTML and CSS files.

3. Open the resizable_elements.js file and add a `.ready()` function.

4. Add the following line that will add `draggable` to the main `<div>` elements:

Click here to view code image

```
$("#resize1, #resize2, #resize3").draggable();
```

5. Add the following line that implements the `resizable` widget on `#resize1`. Notice that the `aspectRatio` is true, so it will force the aspect ratio to be constant when resizing. The `alsoResize` option is set to the image contained in `#resize1` so that the `<img>` element will be resized:

Click here to view code image

```
$("#resize1").resizable(
    {aspectRatio:true, alsoResize:"#resize1 img" });
```

6. Add the following line that implements `resizable` on `#resize2`. This time, `aspectRatio` is not set, so you can adjust the box freely, which distorts the image:

Click here to view code image

```
$("#resize2").resizable(
    {alsoResize:"#resize2 img"});
```

7. Add the following lines that append a series of `<p>` elements to `#resize3` and then add the resizable widget. You also resize the `#list <div>` when resizing the widget to shrink the scrollable list with the box. Also, you changed the `handles` option to `"n,s,e,w"`, which allows you to resize the elements from the top, bottom, or sides, but not the corners:

Click here to view code image

```
for(var i=0; i<100; i++){
  $("#list").append($("<p></p>").html("Item "+i)); }
$("#resize3").resizable(
    {alsoResize:"#resize3 #list", handles:"n,s,e,w"});
```

8. Save all three files and then open the HTML document in a web browser, as shown in Figure 18.3. You should be able to drag and resize the elements and test the interactions.

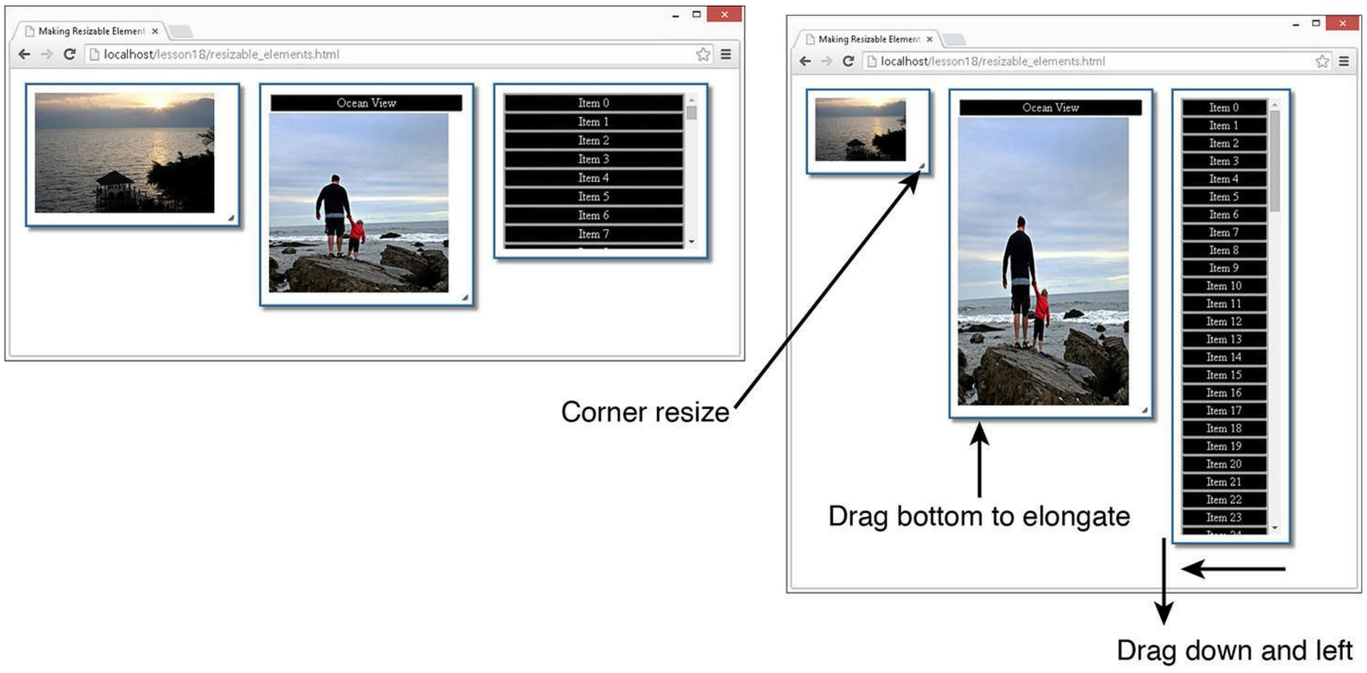


FIGURE 18.3 Applying jQuery UI resizing.

LISTING 18.7 resizable_elements.html HTML Document That Adds the Web Page

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Making Resizable Elements</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/resizable_elements.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/resizable_elements.css">
  </head>
  <body>
    <div id="resize1"><img src="/images/beachhouse.jpg" /></div>
    <div id="resize2">
      <p>Ocean View</p>
      <img src="/images/img7.jpg" />
    </div>
    <div id="resize3"><div id="list"></div></div>
  </body>
</html>
```

LISTING 18.8 resizable_elements.js jQuery and jQuery UI Implements Resizing and Moving the Page Elements

Click here to view code image

```
$(document).ready(function(){
  $("#resize1, #resize2, #resize3").draggable();
  $("#resize1").resizable(
      {aspectRatio:true, alsoResize:"#resize1 img" });
  $("#resize2").resizable(
      {alsoResize:"#resize2 img"});
  for(var i=0; i<100; i++){
    $("#list").append($("<p></p>").html("Item "+i)); }
  $("#resize3").resizable(
      {alsoResize:"#resize3 #list", handles:"n,s,e,w"});
});
```

LISTING 18.9 resizable_elements.css CSS Code That Styles the Page

Click here to view code image

```
img {
  width:230px; }
#resize1, #resize2, #resize3 {
  width:250px; padding:10px; display:inline-block;
  margin:10px; vertical-align:top;
  border:3px ridge white; box-shadow: 5px 5px 5px #888888; }
p {
  margin:2px; border:1px dotted; text-align:center; }
#list {
  height:200px; overflow-y:auto; }
```

Applying the Selectable Widget

Another frequent request for users is the capability to easily select multiple items on a page using a bounding box. The selectable widget provides that functionality by allowing the user to draw a box, or "lasso," around selectable children inside the selectable element using the mouse. Items inside the box are selected in the list.

Table 18.8 describes the more common selectable options. The following shows an example of attaching the `selectable` widget to an element and specifying the `tolerance` as `fit`:

Click here to view code image

```
$("#ul1"). selectable ({tolerance:"fit"});
```

Option	Description
appendTo	Specifies a selector that defines what element to attach the lasso to when dragging. This defaults to "body".
filter	Specifies a selector to use to define which child element can be selected in the bounding box.
tolerance	Can be set to "fit", meaning that the lasso overlaps the child element entirely, or "touch", meaning that any part of the lasso overlaps the child element.

TABLE 18.8 Common Selectable Widget Options

The selectable widget also provides the additional events so handlers can be attached to the selectable element or its children when changing the selection. Table 18.9 lists the events that you can access on selectable items.

Event	Description
selectableselected	Triggered at the end on the new elements selected.
selectableselecting	Triggered during select on each element selected.
selectablestart	Triggered on selectable when selecting starts.
selectablestop	Triggered on selectable when selecting stops.
selectableunselect	Triggered at the end on the elements unselected.
selectableunselected	Triggered during select on each element unselected.

TABLE 18.9 Selectable Widget Events

Each of the selectable events will pass the event object along with a `ui` object that will have a value for each of the events representing the selectable element. For example, the following code adds a `selectableselected` event to an element and then accesses the `selected` attribute:

Click here to view code image

```
$("#list1").selectable();
$("#list1").on("selectableselected", function(e, ui){
ui.selected.effect("shake");
});
```

Note

The `.ui-selecting` class is appended to child elements that are currently being selected. After a child element is selected, the `.ui-selected` class will be appended. This allows you to define some basic styles in CSS without having to add/remove classes in the selectable event handlers.

Try it Yourself: Creating Selectable Sets

In this example, you implement `selectable` on a `<list>` and a group of images. The purpose of the example is to give you a look at a couple of ways to implement the `selectable` widget. You will get a chance to apply some of the options and use different event handlers to interact with the selection process.

The code for the example is in Listings 18.10, 18.11, and 18.12. Use the following steps to create the dynamic web page:

1. In Eclipse, add the lesson18/selectable_sets.html, lesson18/js/selectable_sets.js, and lesson18/css/selectable_sets.css files.
2. Add the code shown in Listings 18.10 and Listings 18.12 to the HTML and CSS files. Notice in Listings 18.12 that `.ui-selected` and `.ui-selecting` classes are added for both `#set1` and `#set2`.
3. Open the selectable_sets.js file and add a `.ready()` function.
4. Add the following line that will populate the `#set1` list:

Click here to view code image

```
02   for(var i=0; i<100; i++){
03     $("#set1").append($("<p></p>").html("Item "+i)); }
```

5. Add the following lines to implement `selectable` on `#set1` along with `selectablestart`, `selectableselecting`, and `selectablestop` event handlers. Notice in the `selectable selecting` event handler that the `ui.selecting.innerHTML` value is used to update the list of items being selected:

Click here to view code image

```
04   $("#set1").selectable({ filter:"p" });
05   $("#set1").on("selectablestart", function(e, ui){
06     $("span").html("Selecting "); });
07   $("#set1").on("selectableselecting", function(e, ui){
08     $("span").append(ui.selecting.innerHTML+", "); });
09   $("#set1").on("selectablestop", function(e, ui){
10     $("span").html("Selection Complete"); });
```

6. Add the following lines to implement selectable on `#set2`, which is a `<div>` full of images. Inside the `selectablestop` event handler, you use the `.ui-selected` class attribute to find the selected images. In line 12, a `highlight` effect is applied to the selected images, and the `<span>` element is updated with the count of selected images:

Click here to view code image

```
$("#set2").selectable();
$("#set2").on("selectablestop", function(e, ui){
  var selection = $("#set2 .ui-selected");
  selection.effect("highlight");
  $("span").html("Selected "+ selection.length +
                 " Photos"); });
```

7. Save all three files and then open the HTML document in a web browser, as shown in Figure 18.4. You should be able to drag to select items in the list as well as images.

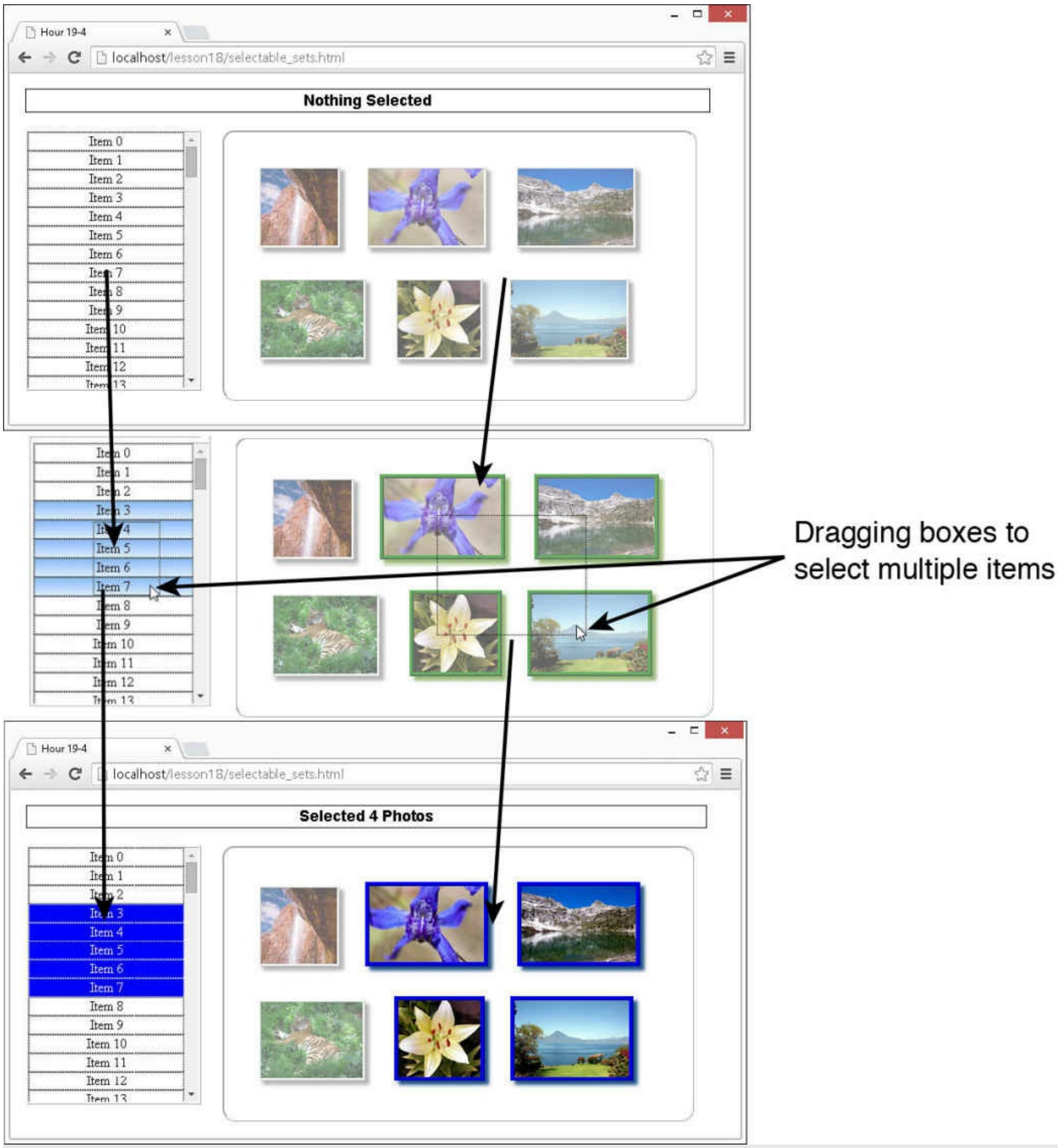


FIGURE 18.4 Applying jQuery UI selecting items in a list using a bounding box.

LISTING 18.10 selectable_sets.html HTML Document That Adds the Web Page

Click here to view code image

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

```
  <head>
    <title>Sorting Elements</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
2.1.3.min.js"></script>
    <script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/selectable_sets.js">
</script>
    <link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css"
href="css/selectable_sets.css">
  </head>
  <body>
    <span>Nothing Selected</span><br>
    <div id="set1"></div>
    <div id="set2">
      <img src="/images/cliff.jpg" /><img src="/images/flower2.jpg" />
      <img src="/images/lake.jpg" /><img src="/images/tiger.jpg" />
      <img src="/images/flower.jpg" /><img src="/images/volcano.jpg" />
    </div>
  </body>
</html>
```

LISTING 18.11 selectable_sets.js jQuery and jQuery UI Implements Item Selection

Click here to view code image

```
$(document).ready(function(){
  for(var i=0; i<100; i++){
    $("#set1").append($("<p></p>").html("Item "+i)); }
  $("#set1").selectable({ filter:"p" });
  $("#set1").on("selectablestart", function(e, ui){
    $("span").html("Selecting "); });
  $("#set1").on("selectableselecting", function(e, ui){
    $("span").append(ui.selecting.innerHTML+", "); });
  $("#set1").on("selectablestop", function(e, ui){
    $("span").html("Selection Complete"); });
  $("#set2").selectable();
  $("#set2").on("selectablestop", function(e, ui){
    var selection = $("#set2 .ui-selected");
    selection.effect("highlight");
    $("span").html("Selected "+ selection.length +
                   " Photos"); });
});
```

LISTING 18.12 selectable_sets.js CSS Code That Styles the Page

Click here to view code image

```
span {
  display:inline-block; border:1px solid;
  font:bold 18px/26px arial;
  width:800px; text-align:center; margin:10px;}
div {
  display:inline-block; border:3px ridge white;
  vertical-align:top; margin:10px; }
p {
  border:1px dotted; margin:0px; }
#set1 {
  height:300px; width:200px; overflow-y:auto;
  text-align:center; }
#set1 .ui-selecting {
  background-image: -moz-linear-gradient(top , #88BBFF, #DDEEFF);
  background-image: -webkit-linear-gradient(top , #88BBFF, #DDEEFF);
  background-image: -ms-linear-gradient(top , #88BBFF, #DDEEFF); }
#set1 .ui-selected {
  background-color:blue; color:white; }
img {
  height:90px; border:3px ridge white; margin:15px;
  box-shadow: 5px 5px 5px #888888; opacity:.6; }
#set2 {
  width:500px; padding:25px; border-radius:15px; }
#set2 .ui-selecting{
  border:5px ridge green; box-shadow: 5px 5px 5px #558822; }
#set2 .ui-selected{
  border:5px ridge blue; box-shadow: 5px 5px 5px #225588;
  opacity:1; }
```

Sorting Elements with the Sortable Widget

One of the coolest interactions provided by jQuery UI is the `sortable` widget. The sortable widget allows you to drag and reposition the order of HTML elements that are flowing together in a list, table, or just inside a container.

The `sortable` widget repositions the other elements as you drag an item. You can also link sortable containers together so that you can drag an item from one sortable container to another.

Table 18.10 describes the more common sortable options. The following shows an example of attaching a `sortable` widget to an element and specifying the `tolerance` as `fit`:

Click here to view code image

```
$("#ul1").sortable({tolerance:"fit"});
```

Option	Description
`axis`	Specifies the direction that items can be dragged. Values are `x` for horizontal, `y` for vertical, or `false` for both.
`connectTo`	Specifies the selector of another selectable container that you want to be able to drag items from this container to.
`cursor`	Specifies cursor shown while sorting.
`helper`	Defines what element is displayed when sorting. Values can be `"original"`, `"clone"`, or a function that returns a DOM object.
`items`	Specifies a selector used to define what child elements are sortable inside the list.
`opacity`	Specifies the opacity of the helper while sorting.
`placeholder`	Specifies a class name that is applied to the placeholder while dragging.
`scroll`	Boolean. When true, the page will scroll when the sorting element comes to the edge.
`tolerance`	Specifies the method used to determine if a sorting item is in a new position. Acceptable values are as follows: ▶ **`intersect`**—Draggable overlaps droppable at least 50% in both directions. ▶ **`pointer`**—Mouse is over droppable.
`zIndex`	Specifies the `z-index` used when sorting.

TABLE 18.10 Common Sortable Widget Options

Note

To sort table rows using `sortable`, you need to make `<tbody>` sortable, not `<table>`.

The `sortable` widget also provides the additional events so handlers can be attached to the sortable element or its children when sorting. Table 18.11 lists the events that you can access on selectable items.

Event	Description
`sortactivate`	Triggered when a connected list starts sorting.
`sortbeforeStop`	Triggered when sorting stops, but the placeholder is still available.
`sortchange`	Triggered when the DOM position changes during sort.
`sortout`	Triggered when a sorting item leaves the sortable list.
`sortover`	Triggered when a sorting item enters a sortable list.
`sortreceive`	Triggered on receiving container when a sortable item is moved from one sortable to another.
`sortremove`	Triggered on sending container when a sortable item is moved from one sortable to another.
`sort`	Triggered during sorting.
`sortstart`	Triggered when sorting starts.
`sortstop`	Triggered when sorting stops.
`sortupdate`	Triggered when a DOM position changes at the end of sorting.

TABLE 18.11 Sortable Widget Events

Tip

jQuery UI will add the `.ui-sortable-helper` class to the helper element being sorted. You can define your own settings in the CSS for the helper class to control the look while moving the element in the `sortable`.

Each of the `sortable` events will pass the `event` object along with a `ui` object that will have the following values attached to it:

- **helper**—jQuery object representing the helper object being dragged.
- **item**—jQuery object representing the actual object being sorted.
- **offset**—`{top, left}` object for the current offset.
- **originalPosition**—`{top, left}` object for the original position.
- **position**—`{top, left}` object for the current position.
- **sender**—Sortable object that the item is being dragged from when dragging from one sortable to another.

For example, the following code connects `#list1` to `#list2` and then adds a `sortreceived` event that will add a `pulsate` effect on both the sender and recipient:

Click here to view code image

```
$("#list1").sortable({connectWith:"#list2"});
$("#list1").on("sortreceived", function(e, ui){
  ui.sender.effect("pulsate");
  $(this).effect("pulsate "); });
```

Try it Yourself: Implementing Sortable Elements

In this example, you implement `sortable` on a list and a table. There are two lists that are connected together so you can sort from one list to the other. The purpose of the example is to provide some practical examples of using sortable elements along with handling sort events.

The code for the example is in Listings 18.13, 18.14, and 18.15. Use the following steps to create the dynamic web page:

1. In Eclipse, add the lesson18/sortable_elements.html, lesson18/js/sortable_elements.js, and lesson18/css/sortable_elements.css files.

2. Add the code shown in Listing 18.13 and Listing 18.15 to the HTML and CSS files. Notice on line 13 of Listing 18.15 that `.ui-sortable-helper` classes are added to style elements while sorting.

3. Open the sortable_elements.js file and add a `.ready()` function.

4. Add the `images` array with image `name` and `src` locations and the `buildLists()` function shown in lines 1–21 of Listing 18.14. The `buildLists()` function uses the array to populate the elements in `#sorter1` and `#sortTable`.

5. Add the following line that makes `#sorter1` sortable and connects it to `#sorter2`:

Click here to view code image

```
24   $("#sorter1").sortable(
25       {cursor:"move", connectWith:"#sorter2"});
```

6. Add the following lines that implement `sortable` on `#sorter2` and adds the `sort- receive` event handler. The event handler adds a `pulsate` effect on both the sender and receiver when an item is added to `#sorter2` from the other list:

Click here to view code image

```
26   $("#sorter2").sortable(
27       {cursor:"move", connectWith:"#sorter1"});
28   $("#sorter2").on("sortreceive", function(e, ui){
29     ui.sender.effect("pulsate");
```

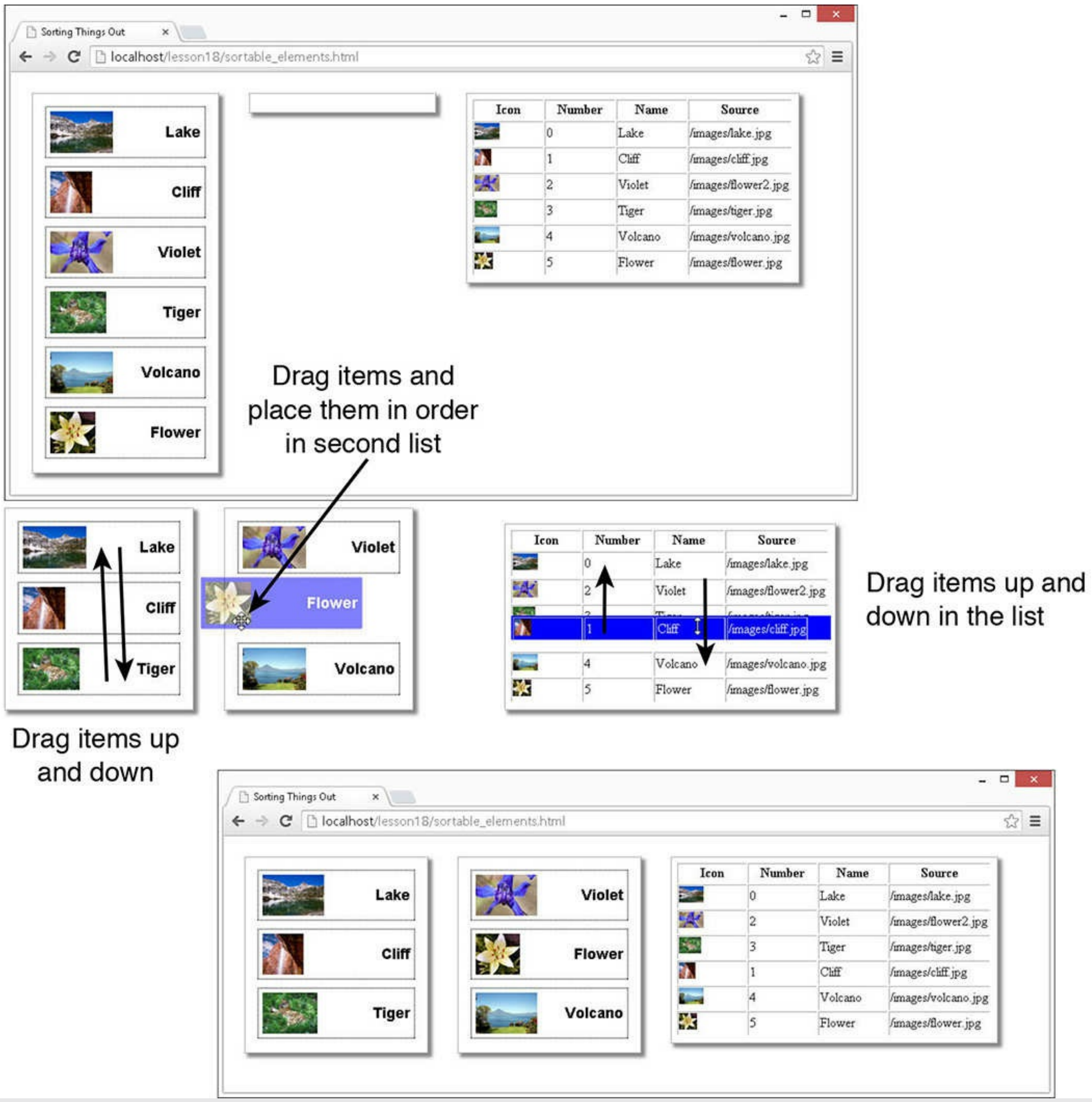
```
      $(this).effect("pulsate"); });
```

7. Add the following code to implement `sortable` on the `#sortTable` `<tbody>` element. Notice that the sort `axis` is restricted to the `y` direction. Also the `sortupdate` event handler is added that makes the sorting item `pulsate` when it has been moved to a new position:

Click here to view code image

```
$("#sortTable").sortable(
    {axis:"y", cursor:"n-resize", });
$("#sortTable").on("sortupdate", function(e, ui){
  ui.item.effect("pulsate"); });
```

8. Save all three files and then open the HTML document in a web browser, as shown in Figure 18.5. You should be able to drag elements from the div on the left to the one in the middle and back. You should also be able to reorder elements in the table.


Sorting Things Out
localhost/lesson18/sortable_elements.html
Lake
Cliff
Violet
Tiger
Volcano
Flower
Drag items and place them in order in second list
Drag items up and down
Drag items up and down in the list

Icon	Number	Name	Source
	0	Lake	/images/lake.jpg
	1	Cliff	/images/cliff.jpg
	2	Violet	/images/flower2.jpg
	3	Tiger	/images/tiger.jpg
	4	Volcano	/images/volcano.jpg
	5	Flower	/images/flower.jpg

Icon	Number	Name	Source
	0	Lake	/images/lake.jpg
	2	Violet	/images/flower2.jpg
	1	Cliff	/images/cliff.jpg
	4	Volcano	/images/volcano.jpg
	5	Flower	/images/flower.jpg

Icon	Number	Name	Source
	0	Lake	/images/lake.jpg
	2	Violet	/images/flower2.jpg
	3	Tiger	/images/tiger.jpg
	1	Cliff	/images/cliff.jpg
	4	Volcano	/images/volcano.jpg
	5	Flower	/images/flower.jpg

FIGURE 18.5 Applying jQuery UI selecting items in a list using a bounding box.

LISTING 18.13 sortable_elements.html HTML Document That Adds the Web Page

Click here to view code image

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sorting Things Out</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
```

```
2.1.3.min.js"></script>
    <script type="text/javascript"
src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/sortable_elements.js">
</script>
    <link rel="stylesheet" type="text/css"
href="http://code.jquery.com/ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css"
href="css/sortable_elements.css">
  </head>
  <body>
    <div id="sorter1"></div>
    <div id="sorter2"></div>
    <table border=1>
      <tbody>
        <tr>
          <th>Icon</th>
          <th>Number</th>
          <th>Name</th>
          <th>Source</th>
        </tr>
      </tbody>
      <tbody id="sortTable"></tbody>
    </table>
  </body>
</html>
```

LISTING 18.14 sortable_elements.js jQuery and jQuery UI Implements Sorting

Click here to view code image

```
var images = [
  {src:"/images/lake.jpg",name:"Lake"},
  {src:"/images/cliff.jpg",name:"Cliff"},
  {src:"/images/flower2.jpg",name:"Violet"},
  {src:"/images/tiger.jpg",name:"Tiger"},
  {src:"/images/volcano.jpg",name:"Volcano"},
  {src:"/images/flower.jpg",name:"Flower"}];
function buildLists(){
  $.each(images, function(i,item){
    var img = $("<img></img>").attr("src", item.src);
    var name = $("<p></p>").html(item.name);
    $("#sorter1").append($("<div></div>").append(img, name));
    var tr = $("<tr></tr>");
    tr.append($("<td></td>").append(
        $("<img></img>").attr("src", item.src)));
    tr.append($("<td></td>").html(i));
    tr.append($("<td></td>").html(item.name));
    tr.append($("<td></td>").html(item.src));
    $("#sortTable").append(tr);
  });
}
```

```
$(document).ready(function(){
  buildLists();
  $("#sorter1").sortable(
      {cursor:"move", connectWith:"#sorter2"});
  $("#sorter2").sortable(
      {cursor:"move", connectWith:"#sorter1"});
  $("#sorter2").on("sortreceive", function(e, ui){
    ui.sender.effect("pulsate");
    $(this).effect("pulsate"); });
  $("#sortTable").sortable(
      {axis:"y", cursor:"n-resize", });
  $("#sortTable").on("sortupdate", function(e, ui){
    ui.item.effect("pulsate"); });
});
```

LISTING 18.15 sortable_elements.css CSS Code That Styles the Page

Click here to view code image

```
#sorter1, #sorter2, table {
    display:inline-block; cursor:move;
  width:200px; padding:10px; vertical-align:top;
  margin:15px; height:auto; box-shadow: 5px 5px 5px #888888;
  border:3px ridge white;  }
#sorter1 div, #sorter2 div {
    width:180px; display:inline-block; height:50px;
  padding:5px; margin:5px; border:1px dotted;
  vertical-align:middle; }
p {
    float:right; margin:0px; display:inline-block; height:50px;
  font:bold 18px/50px arial; vertical-align:top;
  text-align:center; }
img {
    height:50px; }
table {
    width:auto; padding:5px; }
tr {
    background-color:white; }
td {
    min-width:80px; }
#sortTable img {
    height:20px; }
.ui-sortable-helper {
    background-color:blue; color:white; opacity:.5; }
table:hover{
    cursor:move}
```

Summary

Using interaction widgets, you can easily provide some advanced features to your web pages. In this lesson, you created some drag-and-drop elements by making some elements draggable and others droppable using jQuery UI draggable and droppable widgets.

Adding the selectable widget allowed you to draw a bounding box or lasso around multiple items. You used the resizable widget to make a container such as a `<div>` resizable. You also resized the content inside.

Finally, you learned how to implement the sortable widget to sort items in a `<div>` and `<tbody>`. Elements from sortable containers can also be dragged from one container to another.

Q&A

Q. Is it possible to create a custom interaction widget?

A. Yes. Using the jQuery UI `jquery.widget` factory, you can create a custom widget and provide whatever functionality in the prototype that you need.

Q. Is there a way to prevent mouse events from occurring on elements inside an item extended with a jQuery UI widget?

A. Yes. The following code cancels mouse events for items in an element `#myList` that have `class="notSelectable"`:

Click here to view code image

```
$("#myList").mouse({ cancel:".notSelectable"});
```

Workshop

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

Quiz

1. When making an item draggable, what option should you use to keep the original in place while dragging?
2. What droppable event will be triggered when a draggable item is ready to be dropped in it?
3. True or false: It is not possible to keep a fixed ratio when using the resizeable widget on an image.
4. Is there a way to limit what items are selected by the selectable widget?

Quiz Answers

1. Set `helper` to `"clone"` or another DOM object.
2. `dropover`
3. False. You can set the `aspectRatio` option to set a fixed ratio.
4. Yes. Use the `filter` option.

Exercises

1. Open the code in Listings 18.7, 18.8, and 18.9 and add a `resize` handler for the `#resize2` handler. Change the `font-size` attribute based on the width and height the element is resized to.
2. Open the code in Listings 18.13, 18.14, and 18.15 and modify the `#sortTable` element so that it connects with `#sorter2`. You will need to remove the `axis` restriction and change the `width` of `#sort2` to handle the additional width.

Lesson 19. Using jQuery UI Widgets to Add Rich Interactions to Web Pages

What You'll Learn in This Lesson:

- Adding autocomplete, slider, and spinner elements
- How to add a `datepicker` calendar to your forms
- Ways to stylize dialogs and buttons
- Implementing tooltips
- Organizing your web pages with custom menus and tabbed panels
- How to apply status bars to your work flow

jQuery UI provides a wide array of prebuilt widgets that provide extended functionality to HTML elements. These widgets provide functionality that makes forms and other input controls much more intuitive and easy to use, such as a calendar view for choosing dates and expandable menus.

This lesson introduces you to each of those built-in widgets to get you started with them. The purpose of this lesson is an introduction so that you understand what widgets are available and how to implement them into your web pages. The examples are pretty basic, so you should also look at the docs at the jQuery UI site for more information about specific widgets that you might like to use: http://api.jqueryui.com/category/widgets/.

No "Try It Yourself" sections are included this lesson because the examples are basic and easy to follow. The code for each of the listings and the supporting images can be found at the book's website at code/lesson19.

Reviewing Widgets

Before you get started, this section reviews how options and attribute values are accessed on widget elements, because it is a little different than on normal jQuery objects.

When you are creating the widget, options are specified in the object passed to the constructor. For example, the following code sets the `min` and `max` options for a slider widget:

Click here to view code image

```
$("#mySlider").slider({min:2, max:10});
```

To access those options later, you need to specify `"options"` as a parameter to the `.slider()` call. For example, the following code sets and then gets the `max` option:

Click here to view code image

```
var slider = $("#mySlider").slider({min:2, max:10});
slider.slider("option", "max", 100);
var currentMax = slider.slider("option", "max");
```

Attribute values for the widgets are accessed in a similar way. For example, the following code gets and then sets the value of the `value` attribute on a slider:

Click here to view code image

```
var value = $("#mySlider").slider("value");
$("#mySlider").slider("value", value+5 );
```

In fact, `value` and `options` are methods on the slider widget that get called when passed in as the first argument.

Adding an Expandable Accordion Element

The accordion widget combines pairs of headers and content into an expandable accordion view. The accordion widget is applied to a container element, and all the headers are identified. The `header` will become a tab and the content in between the headers will be attached to a separate `<div>` element that is expandable/collapsible.

The code in Listing 19.1 creates an accordion view with an image as the content in each tab. The visual can be seen in Figure 19.1. The following options are part of the example:

- **header**—Specifies a selector to use when determining the header elements.
- **collapsible**—Boolean. If `true`, all content in the accordion can be collapsed.

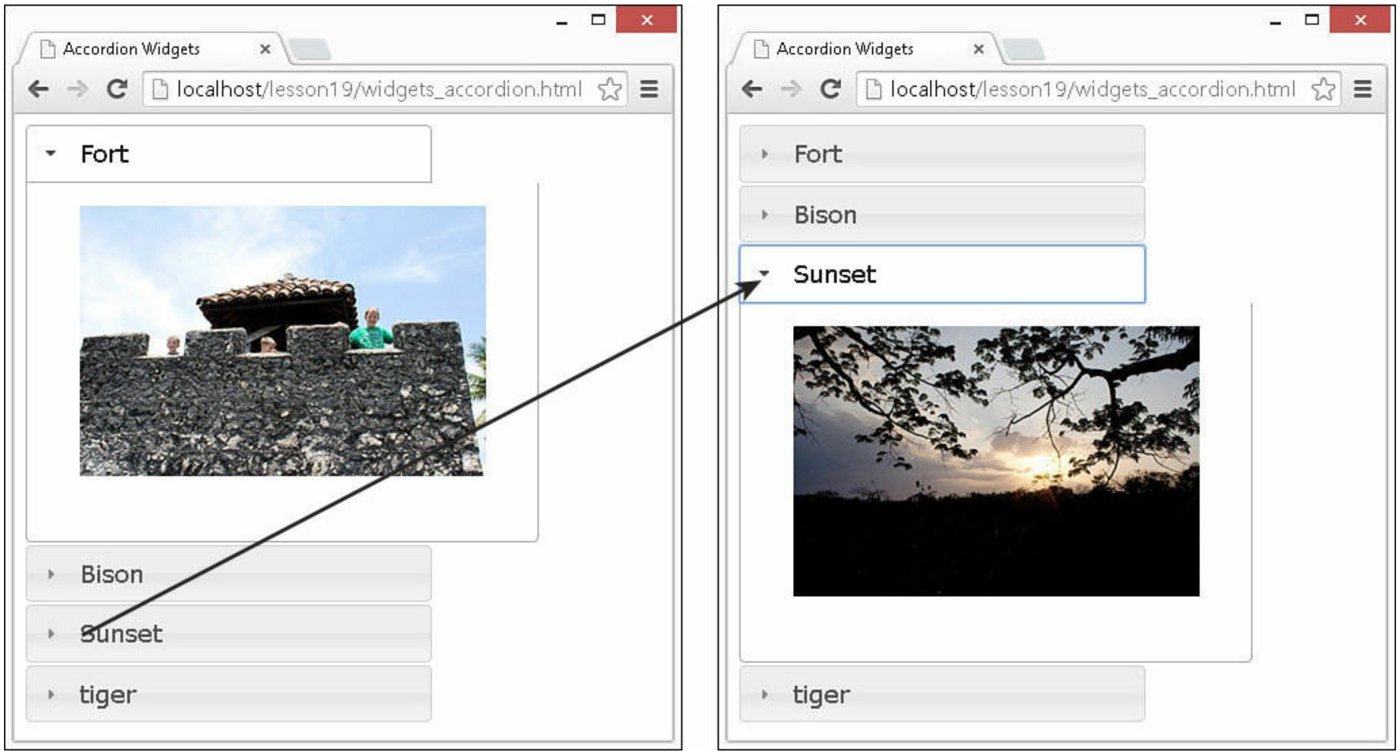

Accordion Widgets
localhost/lesson19/widgets_accordion.html
Fort
Bison
Sunset
tiger
Accordion Widgets
localhost/lesson19/widgets_accordion.html
Fort
Bison
Sunset
tiger

FIGURE 19.1 Simple accordion view.

LISTING 19.1 widgets_accordian.html jQuery, CSS, and HTML to Implement the Accordion

[Click here to view code image](#)

```
...jQuery...
  $(window).load(function(){
    $( "#accordion" ).accordion(
        {header:"p", collapsable:true});
  });
...CSS...
  div { width:300px; }
  img { width:300px; }
...HTML...
  <div id="accordion">
    <p>Fort</p><div>
    <img src="/images/fort.jpg" /></div>
    <p>Bison</p><div>
    <img src="/images/bison.jpg" /></div>
    <p>Sunset</p><div>
    <img src="/images/sunset3.jpg" /></div>
    <p>tiger</p><div>
    <img src="/images/tiger.jpg" /></div>
  </div>
```

Implementing Autocomplete in Form Elements

The autocomplete widget gets attached to text input elements. As the user types into the text input, suggestions from a list are displayed. This is especially helpful in circumstances where you have a finite set of possibilities that can be typed in and you want to make sure the correct spelling is used.

The code in Listing 19.2 creates a basic autocomplete input to specify the day of the week. The visual can be seen in Figure 19.2. The set of days to autocomplete are added by setting the `source` attribute to an array of day names in line 15.

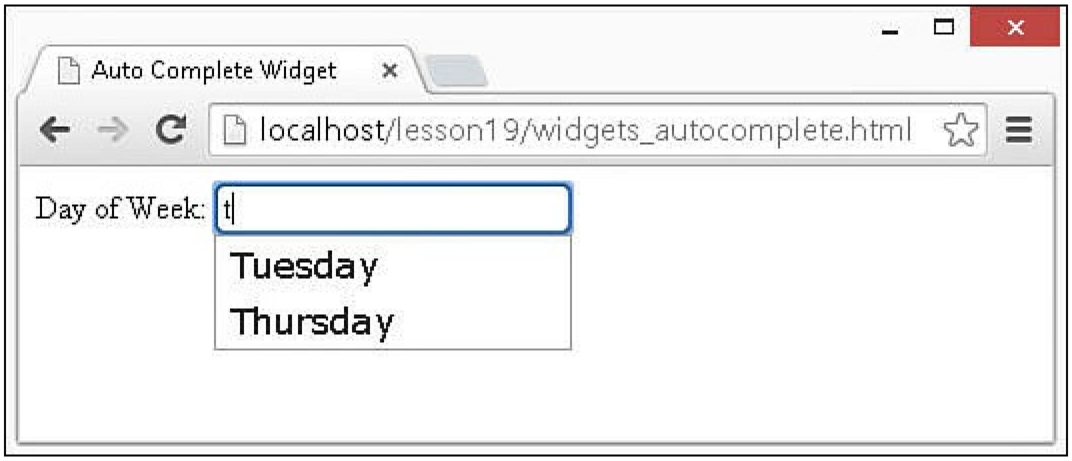


FIGURE 19.2 Autocompleting typing in a day of the week.

LISTING 19.2 widgets_autocomplete.html jQuery, CSS, and HTML to Implement the Autocomplete Field

Click here to view code image

```
...jQuery...
11     function dateChanged(dateStr, Object){
12       $("span").html("Updated to" + dateStr); }
13     $(document).ready(function(){
14       $( "#autocomplete" ).autocomplete({
15         source: ["Monday", "Tuesday", "Wednesday",
16                 "Thursday", "Friday"]
17       });
18     });
...CSS...
21     input {
22       border:2px ridge steelblue; border-radius:5px;
23       padding:3px; }
...HTML...
26     <label for="autocomplete">Day of Week: </label>
27     <input id="autocomplete">
```

Applying jQuery UI Buttons to Form Controls

A great feature of jQuery UI is the work that has been done to stylize buttons. Let's face it, the HTML button styles are outdated and ugly. Using jQueryUI, you can quickly stylize button, check box, and radio elements.

To stylize a single item, all you need to do is call `.button(options)` on the jQuery object that represents it. To stylize a set of radios or check boxes, you call `.buttonset()`.

The code in Listing 19.3 illustrates styling HTML elements as jQuery UI buttons. Notice in line 12, the `icons` option is set to `"ui-icon-gear"`, which is one of the icons included with jQuery UI download. Also, the `text` option is set to `true`. If this option is `false`, only the icon would be displayed. The visual can be seen in Figure 19.3.

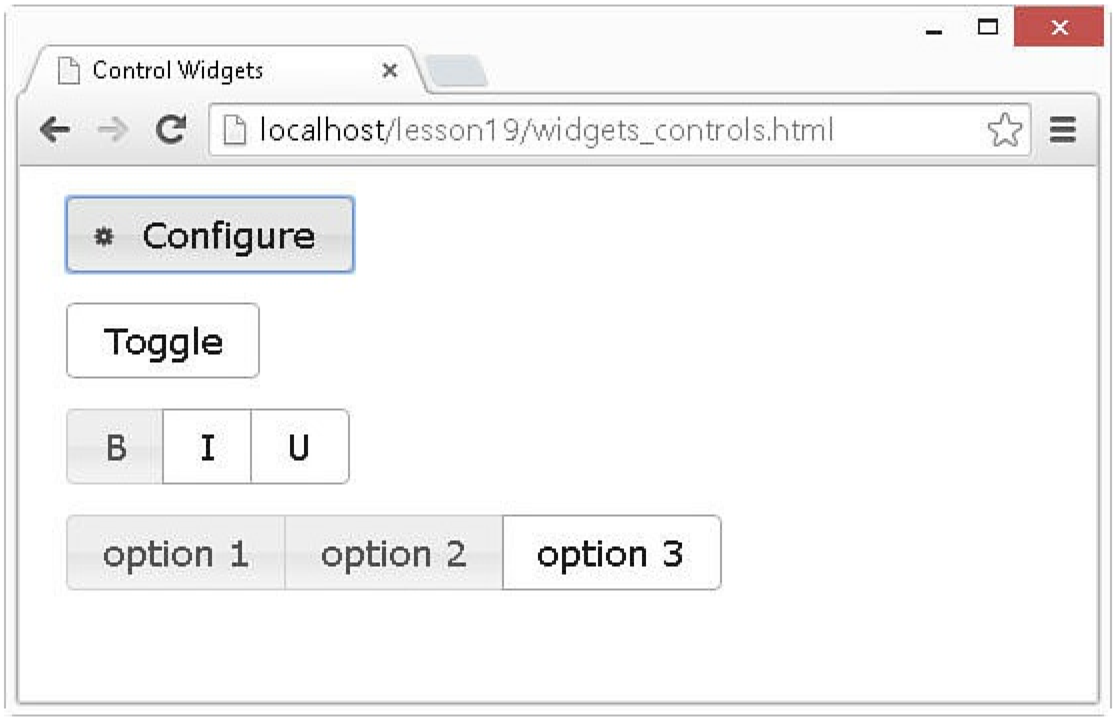


FIGURE 19.3 Styling form elements as jQuery UI buttons.

LISTING 19.3 widgets_controls.html jQuery, CSS, and HTML to Implement the Autocomplete Field

Click here to view code image

```
...jQuery...
11   $(window).ready(function(){
12       $( "#button1" ).button(
13           {icons: {primary: "ui-icon-gear"},text: true});
14       $( "#check" ).button();
15       $( "#format" ).buttonset();
16       $( "#radio" ).buttonset();
17     });
...CSS...
20     div { margin:15px; }
...HTML...
```

```
    <div>
      <button id="button1">Configure</button>
    </div>
    <div>
      <input type="checkbox" id="check" />
      <label for="check">Toggle</label>
    </div>
    <div id="format">
      <input type="checkbox" id="check1" />
      <label for="check1">B</label>
      <input type="checkbox" id="check2" />
      <label for="check2">I</label>
      <input type="checkbox" id="check3" />
      <label for="check3">U</label>
    </div>
    <div id="radio">
      <input type="radio" id="radio1" name="radio" />
      <label for="radio1">option 1</label>
      <input type="radio" id="radio2" name="radio" />
      <label for="radio2">option 2</label>
      <input type="radio" id="radio3" name="radio" />
      <label for="radio3">option 3</label>
    </div>
```

Creating a Calendar Input

The datepicker widget provided with jQuery enables you to implement a calendar interface that allows users to select a specific day using a simple click of the mouse. This can save a lot of problems when users input dates incorrectly because they are typing them by hand.

The datepicker widget is attached to a `text`, `date`, or `datetime <input>` element. When the user clicks the `<input>`, the calendar is displayed. You also add an icon image to launch the datepicker.

The code in Listing 19.4 creates a date with an image icon. The visual can be seen in Figure 19.4. Settings for the following illustrate some of the available options:

- **onSelect**—Specifies a function that will be called each time a new date is selected.
- **showOn**—This is set to “button” so that the datepicker will be launched when the button icon is clicked.
- **buttonImage**—Specifies the location of the image file to use.
- **buttonImageOnly**—When true, the datepicker is launched only when the button icon is clicked and not the `<input>`.
- **numberOfMonths**—Specifies the number of months to display.

- **showButtonPanel**—When true, the Today and Done buttons are displayed on the bottom of the datepicker.
- **dateFormat**—String that describes the format that will be placed in the `<input>` field.

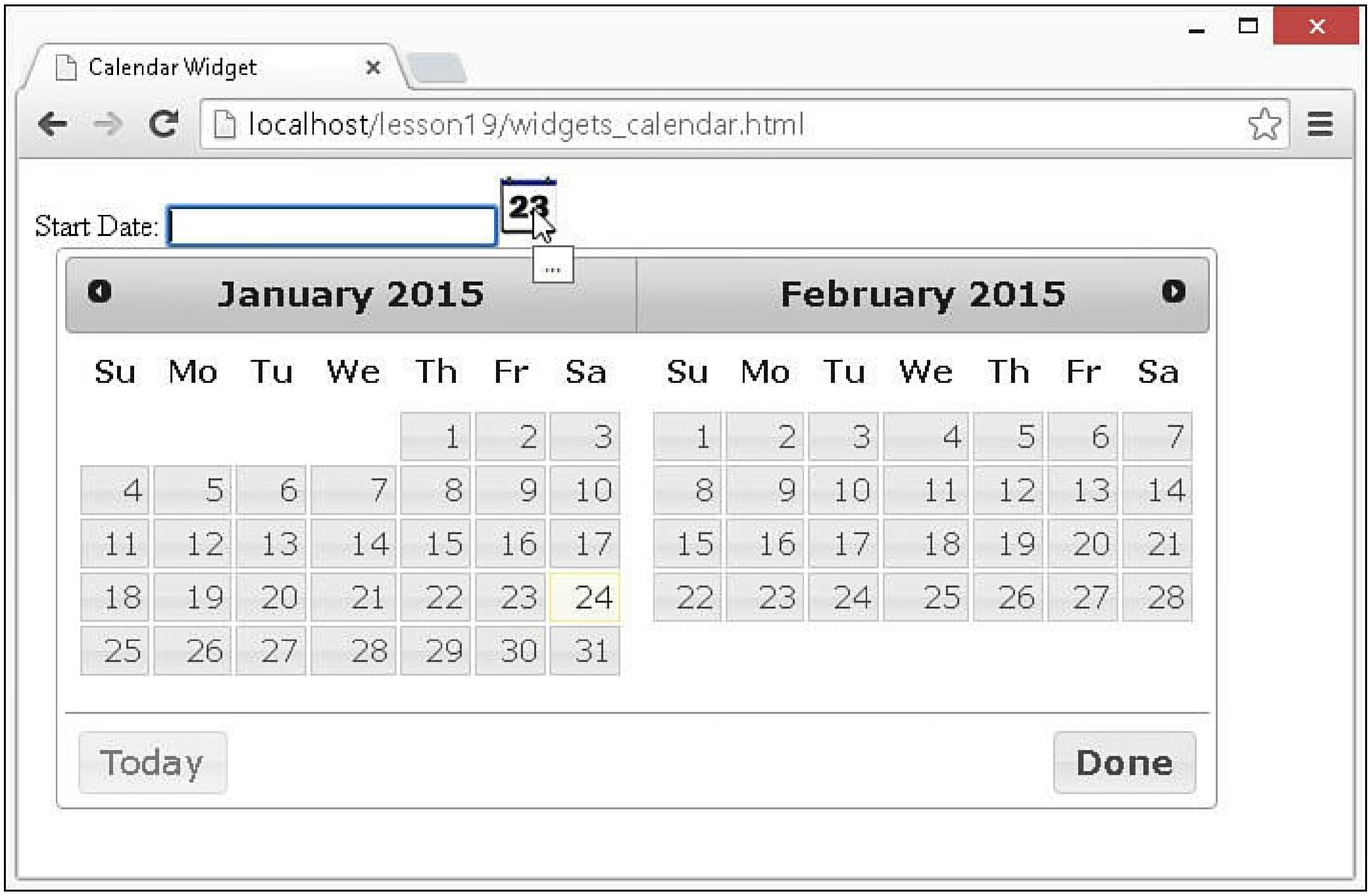


FIGURE 19.4 Adding a datepicker to a date input.

LISTING 19.4 widgets_calendar.html jQuery, CSS, and HTML to Implement the Datepicker Widget

[Click here to view code image]

```
...jQuery...
    function dateChanged(dateStr, Object){
     $("span").html("Updated to" + dateStr); }
    $(document).ready(function(){
     $( "#month" ).datepicker({
       onSelect:dateChanged,
       showOn: "button",
       buttonImage: "/images/calendar32.png",
       buttonImageOnly: true,
       numberOfMonths:2,
       showButtonPanel:true,
       dateFormat: "yy-mm-dd"
```

```
          });
      });
...CSS...
      input { border:2px ridge steelblue; border-radius:3px; }
...HTML...
      <label>Start Date: </label>
      <input type="text" id="month"></input>
      <span></span>
```

Generating Stylized Dialogs with jQuery UI

The dialog widget is a very useful inclusion to jQuery UI. You can easily get rid of the plain dialogs provided in JavaScript and replace them with dialogs that have styled attributes and even forms.

The code in Listing 19.5 creates a jQuery UI dialog that includes an image, icon, button, and some stylized text. The visual can be seen in Figure 19.5. The following options are part of the example:

- **modal**—Boolean. When `true`, other items on the page are disabled until the dialog returns.
- **buttons**—This is an object where the property key specifies the button name as well as the text displayed in the button. The property value specifies a function that will be called when that button is clicked. Notice that in line 13, the `buttons` property is used to specify a button named `Sweet` that executes a function that closes the dialog.

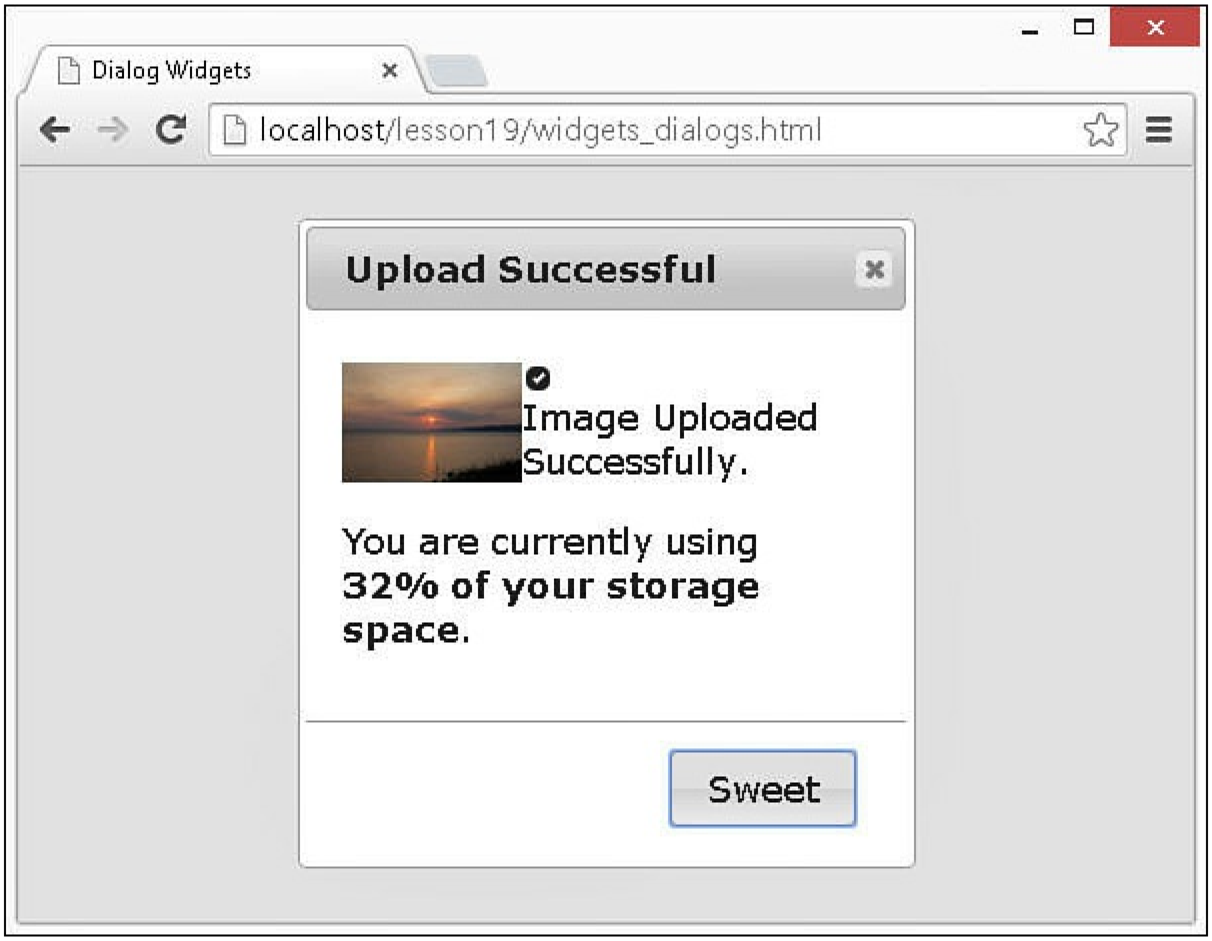


Image Uploaded Successfully.

You are currently using **32% of your storage space.**

FIGURE 19.5 Stylized jQuery UI dialog.

LISTING 19.5 widgets_dialogs.html jQuery, CSS, and HTML to Implement the Dialog

[Click here to view code image](#)

```
...jQuery...
      $(document).ready(function(){
         $( "#dialog" ).dialog({ modal: true,
           buttons: { Sweet: function() {
             $( this ).dialog( "close" ); }}});
      });
...CSS...
      img { height:60px; float:left; }
...HTML...
      <div id="dialog" title="Upload Successful">
        <p>
          <img src="/images/sunset2.jpg" />
          <span class="ui-icon ui-icon-circle-check"></span>
          Image Uploaded Successfully.
        </p>
        <p>You are currently using <br>
          <b>32% of your storage space</b>.</p>
```

```
</div>
```

Implementing Stylized Menus

One of the most used jQuery UI widgets is the menu widget. The menu widget enables you to turn an element tree into an expanding menu. Typically, menus are created by using cascading sets of `<ul>`/`<li>` elements with an `<a>` element that defines the link behavior and menu text.

> **Tip**
>
> You can customize the element tags that are used to build the element using the `menus` option; for example, `menus:"div.menuItem"`.

The code in Listing 19.6 creates a jQuery UI menu from a set of list items. Notice in the HTML that some of the `<li>` fields include a `<span>` that has `class="ui-icon ui-icon-{type}"`. These items include the jQuery UI icon specified along with the menu text.

The selected item is displayed in the `<p>` element to show how the selection handler works using the `menuselect` event handler defined in line 14. Also, the width of the menu is defined in the CSS code on line 20 by setting the `width` value in the `.ui-menu` class. The visual can be seen in Figure 19.6.

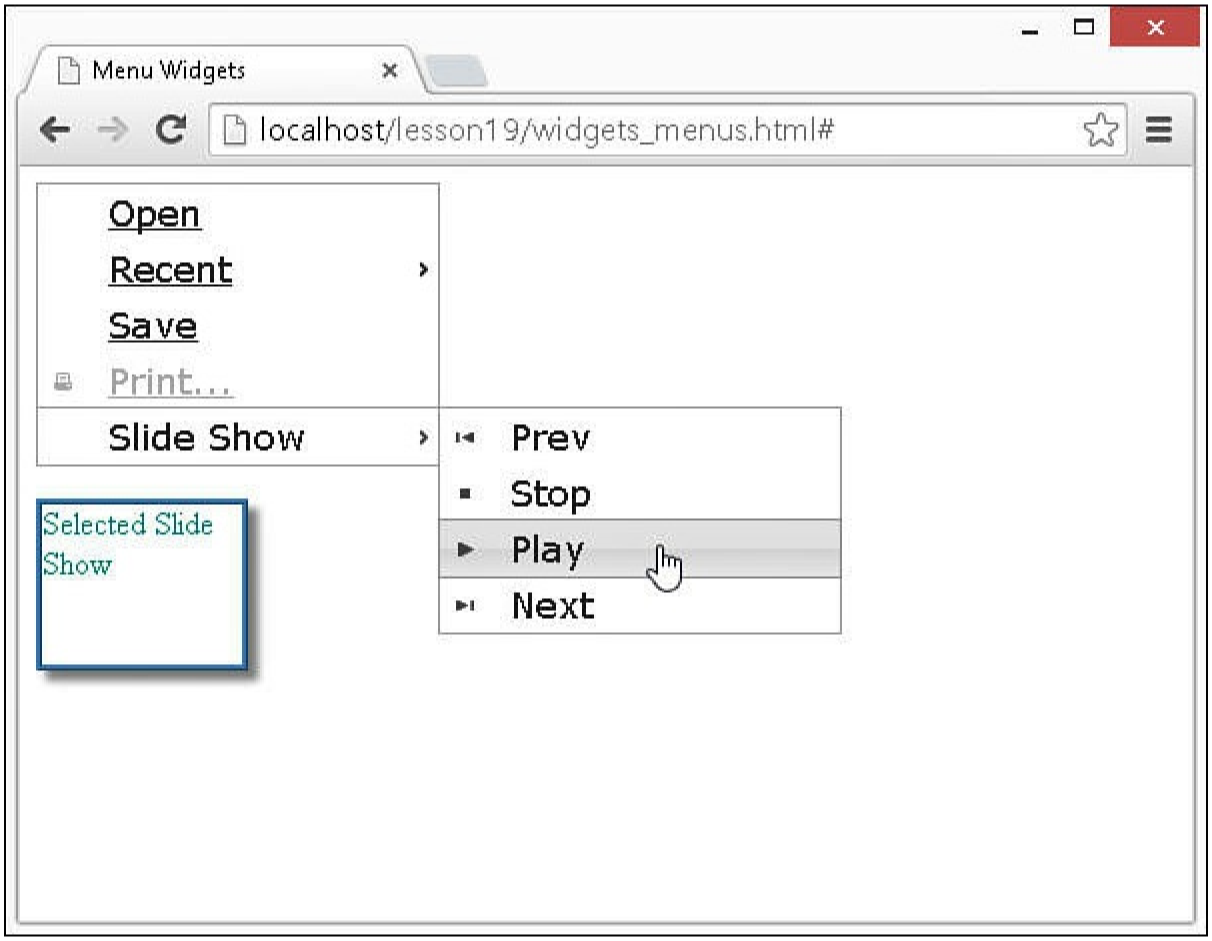


FIGURE 19.6 Stylized jQuery menu.

LISTING 19.6 widgets_menus.html jQuery, CSS, and HTML to Implement the Menus

[Click here to view code image]

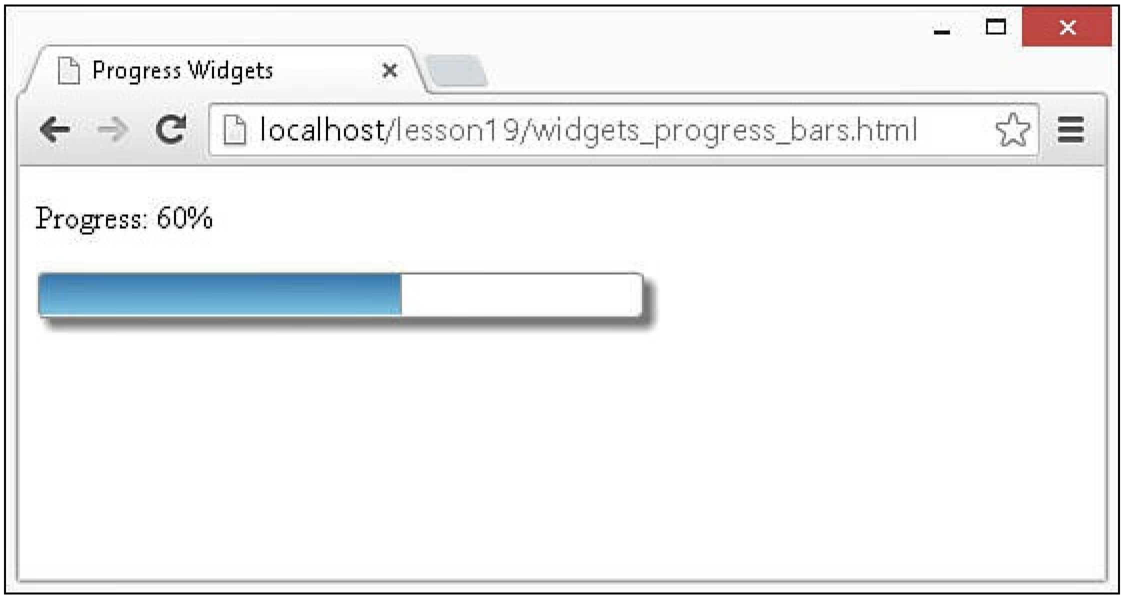
```
...jQuery...
     $(document).ready(function(){
       $( "#menu" ).menu();
       $( "#menu" ).on("menuselect", function(e, ui){
         $("p").html("Selected " +
             ui.item.children("a:first").html());
       });
     });
...CSS...
    .ui-menu { width: 200px; }
    p { box-shadow: 5px 5px 5px #888888;
        border:3px ridge steelblue; color:teal;
        display:inline-block; height:80px; width:100px; }
...HTML...
     <ul id="menu">
       <li><a href="#">Open</a></li>
       <li><a href="#">Recent</a><ul>
```

```
        <li><a href="#">Some File</a></li>
        <li><a href="#">Another File</a></li>
        </ul></li>
      <li><a href="#">Save</a></li>
      <li class="ui-state-disabled">
        <a href="#"><span class="ui-icon ui-icon-print">
        </span>Print...</a></li>
      <li><a href="#">Slide Show</a><ul>
        <li>
          <a href="#">
            <span class="ui-icon ui-icon-seek-start"></span>Prev</a>
        </li>
        <li>
          <a href="#">
            <span class="ui-icon ui-icon-stop"></span>Stop</a>
        </li>
        <li>
          <a href="#">
            <span class="ui-icon ui-icon-play"></span>Play</a>
        </li>
        <li>
          <a href="#">
            <span class="ui-icon ui-icon-seek-end"></span>Next</a>
        </li>
        </ul></li>
    </ul>
    <p></p>
```

Creating Progress Bars

The progress bar widget allows you to create some very simple-to-implement progress bars. The progress bar is controlled by changing the `value` property that ranges from 0 to 100. The progress is represented by an element with a class `.ui-progressbar-value`.

The code in Listing 19.7 provides an example of implementing a progress bar. The bar is updated in the `inc()` function, which illustrates getting and setting the value of the progress bar. `setTimeout()` is used for time delay. The visual can be seen in Figure 19.7.



Progress: 60%

FIGURE 19.7 Progress bar being updated by `setTimeout()`.

LISTING 19.7 widgets_progress_bars.html jQuery, CSS, and HTML to Implement the Progress Bar

[Click here to view code image]

```
...jQuery...
function inc(){
  var value = $("#progressbar").progressbar("value") + 5;
  if (value <= 100){
    $("p").html("Progress: " + value + "%");
    $("#progressbar").progressbar("value", value);
    setTimeout(inc, 100);
   }
}
$(document).ready(function(){
  $("#progressbar").progressbar({ value: 0});
  inc();
});
...CSS...
#progressbar {
  box-shadow: 5px 5px 5px #888888; border:2px ridge;
  display:inline-block; height:20px; width:300px; }
#progressbar .ui-progressbar-value{
  background-image: -moz-linear-gradient(top, steelblue, skyblue);
  background-image: -webkit-linear-gradient(top, steelblue, skyblue);
  background-image: -ms-linear-gradient(top , steelblue, skyblue);
}
...HTML...
<p></p>
<div id="progressbar"></div>
```

Note

The `progressbar` widget is often used in conjunction with the jQuery Ajax `beforeSend()` function to display the progress of a request.

Implementing Slider Bars

The slider widget allows you to create slider controls that adjust a value by dragging the mouse. The slider has two components: the slide and the handle. The slide is styled by the `.ui-slider-range` class, and the handle is styled by the `.ui-slider-handle` class.

Tip

You can define both a min and max handle that allows you to use a single slider control to define a range instead of a single value.

The code in Listing 19.8 provides an example of implementing a set of sliders that are used to adjust the background color of another element. The slider is applied to the `<div>` elements in lines 22–29 and sets the following options:

- **orientation**—Can be set to `"horizontal"` or `"vertical"`.
- **range**—Can be set to `true`, `"min"`, or `"max"`. Used to define the range. `"min"` goes from the slider min to one handle on the slider, and `"max"` goes from one handle on the slider to the slider max.
- **max**—Specifies the maximum value.
- **value**—Specifies the current value.
- **slide**—Event handler to call when the slide moves.
- **change**—Event handler to call when the slide value changes.

Also pay attention to the class settings in lines 41–46 of the CSS. Those alter the appearance of the slider and handler. The visual can be seen in Figure 19.8.

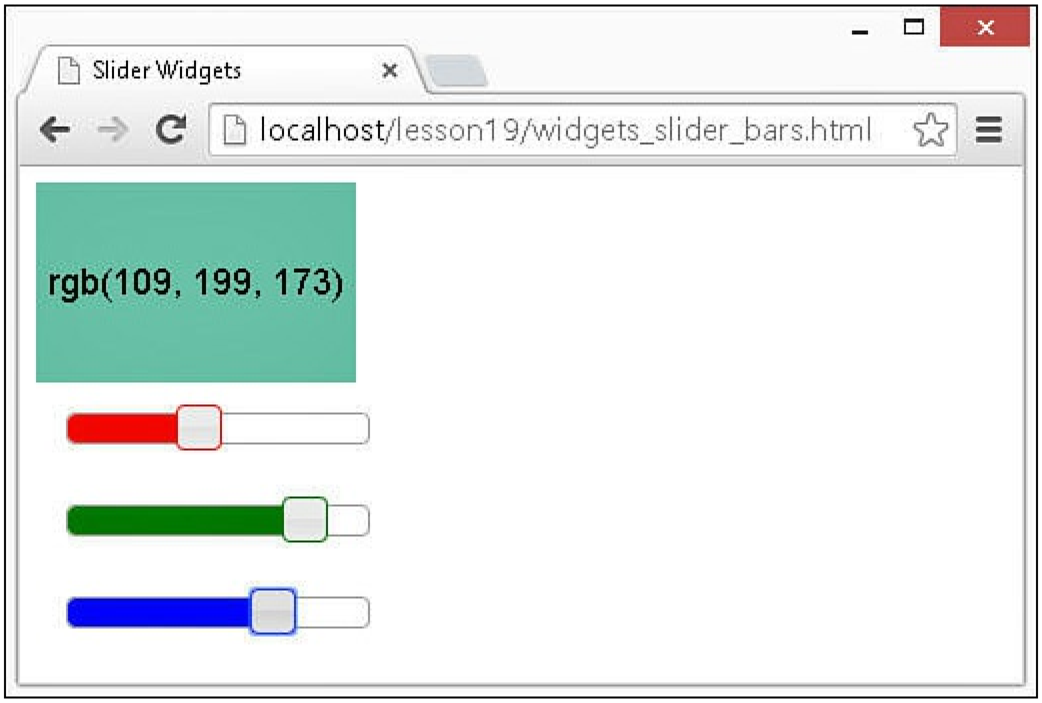


FIGURE 19.8 Sliders used to choose a color based on the RGB value.

LISTING 19.8 widgets_slider_bars.html jQuery, CSS, and HTML to Implement the Sliders

[Click here to view code image]

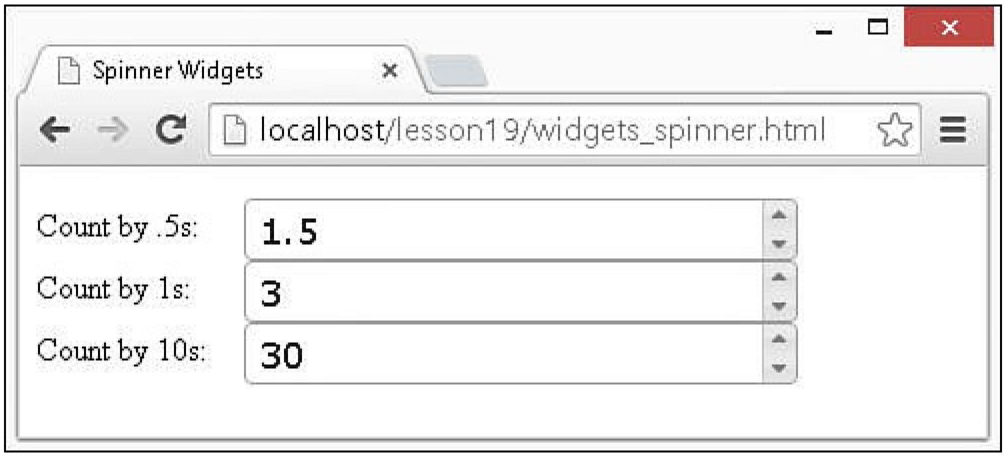
```
...jQuery...
function cValue(selector){
  var v = $(selector).slider("value").toString( 16 );
  if (v.length ===1) { v = "0" + v;}
  return v;
}
function refreshSwatch() {
  $("#mix").css("background-color", "#" + cValue("#red") +
    cValue("#green") +  cValue("#blue"));
  $("#mix").html($("#mix").css("background-color"));
}
$(document).ready(function(){
   $( "#red, #green, #blue" ).slider({
    orientation: "horizontal",
    range: "min",
    max: 255,
    value: 127,
    slide: refreshSwatch,
    change: refreshSwatch
  });
  $("#red").slider("value", 128);
  $("#green").slider("value", 128);
  $("#blue").slider("value", 128);
});
```

```
...CSS...
#mix {
  width:160px; height:100px; text-align:center;
  font:18px/100px arial; }
#red, #green, #blue {
  float: left; clear: left; width: 150px; margin: 15px; }
#red .ui-slider-range { background:red; }
#red .ui-slider-handle { border-color:red; }
#green .ui-slider-range { background:green; }
#green .ui-slider-handle { border-color:green; }
#blue .ui-slider-range { background:blue; }
#blue .ui-slider-handle { border-color:blue; }
...HTML...
<div id="mix"></div>
<div id="red"></div>
<div id="green"></div>
<div id="blue"></div>
```

Adding a Value Spinner Element

The spinner widget allows you to create a value input that has up and down arrows that enable you to increment/decrement the value with the mouse instead of typing in numbers.

The code in Listing 19.9 provides an example of implementing basic spinner input with 3 inputs: one that counts by .5, one that counts by 1s, and one that counts by 10s. The step value determines the amount each click of the mouse increments or decrements the input value. The visual can be seen in Figure 19.9.



Count by .5s: 1.5
Count by 1s: 3
Count by 10s: 30

FIGURE 19.9 Simple spinner input.

LISTING 19.9 widgets_spinner.html jQuery, CSS, and HTML to Implement the Spinner

Click here to view code image

```
...jQuery...
```

```
$(document).ready(function(){
  $( "#spin1" ).spinner({step: 0.5});
  $( "#spin2" ).spinner({step: 1});
  $( "#spin3" ).spinner({step: 10});
});
...CSS...
label { display:inline-block; width:100px; }
...HTML...
<p>
  <label for="spin1">Count by .5s:</label>
  <input id="spin1" name="value" /><br>
  <label for="spin2">Count by 1s:</label>
  <input id="spin2" name="value" /><br>
  <label for="spin3">Count by 10s:</label>
  <input id="spin3" name="value" /><br>
</p>
```

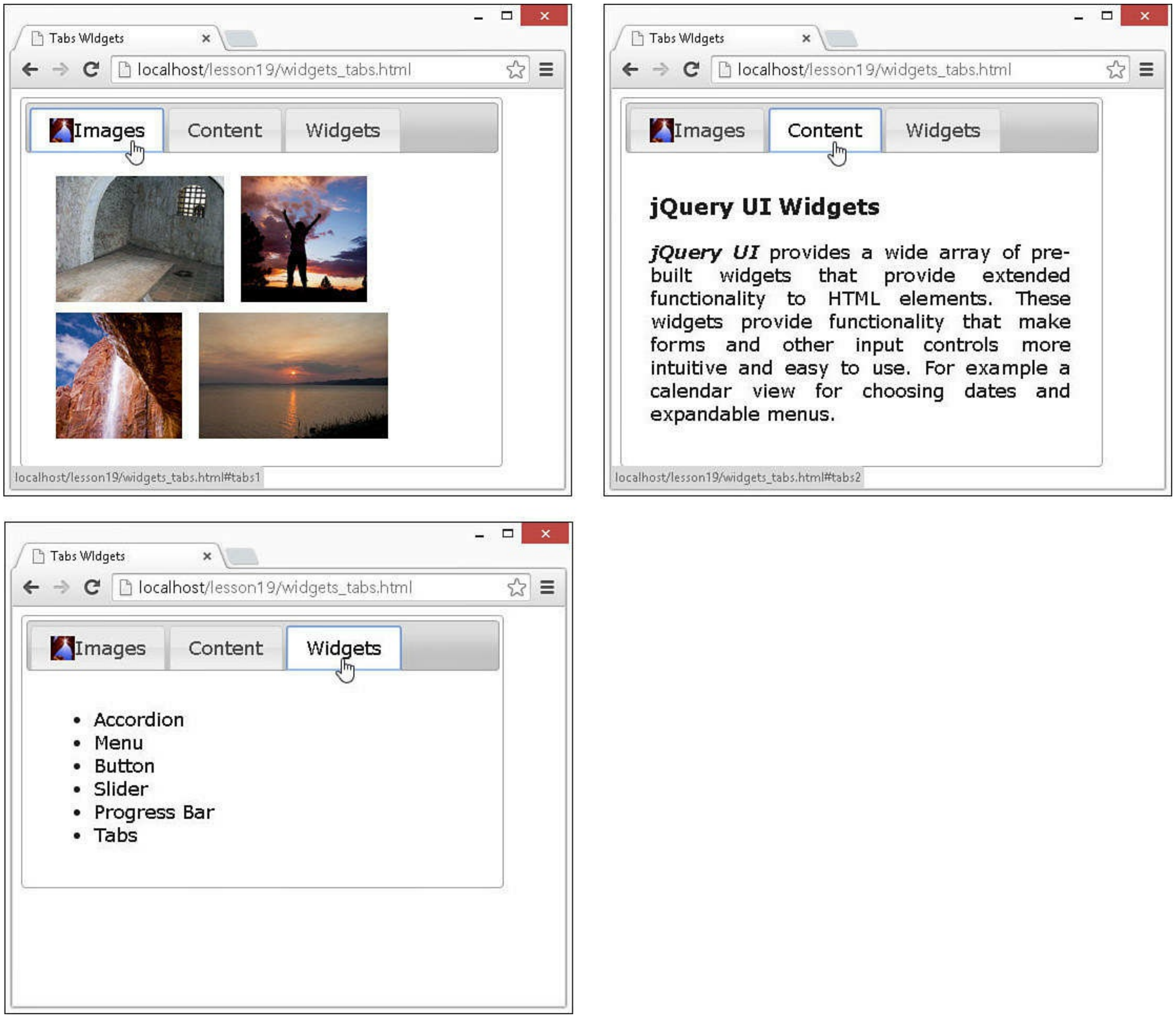
Creating Tabbed Panels

The tabs widget allows you to create a series of tabbed panels. This provides the capability to easily break chunks of content up and yet have it easily accessible. Each tab represents a panel that contains content that can be revealed by activating the tab.

Tabs are a list of elements containing an `<a>`. The tabs widget links the tab with content using the `href` value of an `<a>` element. Tabs can be activated by clicking them, or you can set options to have the tabs enabled on `mouseover` or some other event.

The code in Listing 19.10 provides an example of three tabs: one with image content, one with text, and another with a list. The tabs are defined with the following options. The visual can be seen in Figure 19.10:

- **event**—Specifies what event on the tab element will cause the tab to become active; for example, `click` or `mouseover`.
- **collapsible**—Boolean. Specifies whether the active tab can be collapsed by clicking on it or by setting `active` to `false`.
- **active**—When set to `false`, the active tab is collapsed. Requires `collapsible` to be `true`.


Tabs Widgets
localhost/lesson19/widgets_tabs.html
Images
Content
Widgets
localhost/lesson19/widgets_tabs.html#tabs1
localhost/lesson19/widgets_tabs.html#tabs2

jQuery UI Widgets

jQuery UI provides a wide array of pre-built widgets that provide extended functionality to HTML elements. These widgets provide functionality that make forms and other input controls more intuitive and easy to use. For example a calendar view for choosing dates and expandable menus.

- Accordion
- Menu
- Button
- Slider
- Progress Bar
- Tabs

FIGURE 19.10 Three tabs with different content.

Note

In the example, the tabs link to locations on the page using id values. You can also specify links external to the web page or even retrieve data from the server via AJAX to populate the panel content.

LISTING 19.10 widgets_tabs.html jQuery, CSS, and HTML to Implement the Tabbed Panel

[Click here to view code image](#)

```
...jQuery...
```

```
  $(document).ready(function(){
    $( "#tabs" ).tabs(
        { event: "mouseover", collapsible: true,
          active:"false"});
  });
...CSS...
  * { vertical-align:top; }
  img { height:120px; margin:5px; }
  .mini {height:23px; margin:0px; }
  #tabs { width:450px; }
  p { text-align:justify; }
...HTML...
<div id="tabs">
  <ul>
    <li>
      <a href="#tabs1">
        <img class="mini" src="/images/arch.jpg" />Images</a>
    </li>
    <li><a href="#tabs2">Content</a></li>
    <li><a href="#tabs3">Widgets</a></li>
  </ul>
  <div id="tabs1">
    <img src="/images/jail.jpg" />
    <img src="/images/sunset.jpg" />
    <img src="/images/cliff.jpg" />
    <img src="/images/sunset2.jpg" />
  </div>
  <div id="tabs2">
    <h3>jQuery UI Widgets</h3>
    <p><b><i>jQuery UI</i></b>
      provides a wide array of pre-built widgets that
      provide extended functionality to HTML elements.
      These widgets provide functionality that make forms
      and other input controls more intuitive
      and easy to use. For example a calendar
      view for choosing dates and expandable menus.
    </p>
  </div>
  <div id="tabs3">
    <ul>
      <li>Accordion</li>
      <li>Menu</li>
      <li>Button</li>
      <li>Slider</li>
      <li>Progress Bar</li>
      <li>Tabs</li>
    </ul>
  </div>
</div>
```

Adding Tooltips to Page Elements

The tooltips widget allows you to easily add tooltips to form input, images, and just about any other page element. To implement tooltips, apply `.tooltip(options)` to the document or other container. Inside the options, specify the items that should include tooltips and then the tooltip content handler.

As the mouse hovers over an item supported by the tooltip, the tooltip message is displayed. The code in Listing 19.11 provides an example of implementing tooltips on `<input>` and `<img>` elements. The visual can be seen in Figure 19.11:

- **items**—Specifies the selector used to determine whether the page element supports tooltips.
- **content**—Tooltip handler function called when a supported element is hovered over. The function should return the content to be displayed. Notice that for the image, a mini version is displayed in the tooltip.
- **position**—Specifies the position to place the tooltip; for example:

Click here to view code image

```
position: {my: "left top+15", at: "left bottom", collision: "flipfit" }
```

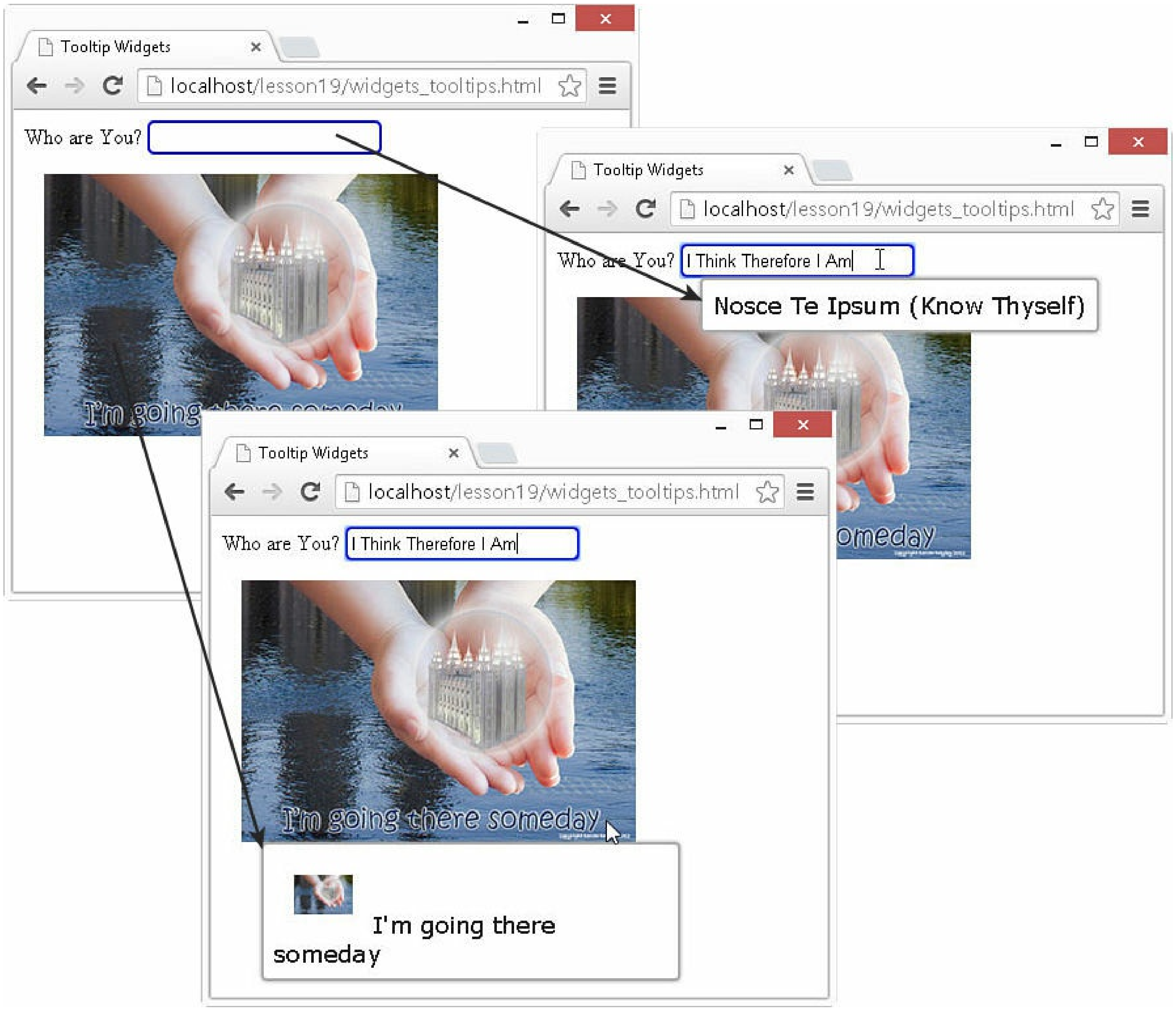


FIGURE 19.11 Tooltips are displayed when page elements are hovered over.

LISTING 19.11 widgets_tooltips.html jQuery, CSS, and HTML to Implement the Tabbed Panel

[Click here to view code image]

```
...jQuery...
     $(document).ready(function(){
       $(document).tooltip({
         items: "img, input",
         position: { my: "left+15 top", at: "left bottom",
                    collision: "flipfit" },
         content: function() {
           var obj = $(this);
           if (obj.is("input")) { return obj.attr( "title" ); }
           if (obj.is("img")) {
```

```
        var img = $("<img></img>").addClass("mini")
                    .attr("src", obj.attr("src"));
        var span = $("<span></span>")
                     .html(obj.attr( "alt" ));
        return $("<div></div>").append(img, span); }
    }});
});
...CSS...
input {
  border:2px ridge blue; border-radius:5px; padding:3px; }
img { height:200px; margin:15px; }
.mini {height:30px; }
...HTML...
<label for="size">Who are You?</label>
<input id="size" title="Nosce Te Ipsum (Know Thyself)" /><br>
<img src="/images/someday.jpg" alt="I'm going there someday"/>
```

Creating Custom Widgets

Creating custom widgets is a fairly simple process. The `$.widget()` factory needs a name and then the prototype object shown next. The widget factory will handle setting everything up so that the widget can be applied to page elements.

The code in Listing 19.12 shows a basic outline for a custom widget. To create the widget, follow these steps:

1. Replace "custom.mywidget" with your own name.
2. Add the options with default values.
3. Add any additional code in `_create()`, `_refresh()`, `_destroy()`, `_setOptions()`, and `_setOption()`.
4. Add additional attributes or methods to the prototype object to implement the widget.

LISTING 19.12 widgets_custom.html jQuery Code Outline to Implement a Custom Widget

Click here to view code image

```
$.widget("custom.mywidget", {
  options : {
    // custom options
  },
  _create: function() {
    //creation code
  },
  _refresh: function() {
    //refresh code called when element refreshed
  },
  _destroy: function() {
    //cleanup code called when widget is destroyed
  },
  _setOptions: function() {
    // _super and _superApply handle keeping the right this-context
    this._superApply( arguments );
    this._refresh();
  },
  _setOption: function( key, value ) {
    // set individual option value override code
    this._super( key, value );
  }
});
```

Summary

jQuery UI includes a large set of built-in widgets that provide some much-desired functionality and styling left out of conventional HTML elements. In this lesson, you saw the widgets and how to implement them on pages, including autocomplete elements, buttons, datepicker, dialog, menu, progress bar, sliders, spinners, tabs, and tooltips.

Q&A

Q. What is the Globalize plug-in and why would I want to use it?

A. The Globalize plug-in is a project that simplifies the process of internationalizing your web pages to match currency, time, and other value formatting that varies from locale to locale. If you plan to have a website with elements that need localization, it is a good idea to at least check it out.

Workshop

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

Quiz

1. How do you get the value of a slider from element `#mySlider`?

2. True or false: Datepicker widgets must be attached to an `<input>` element.

3. How do you attach an event handler to a button, a check box, or a radio stylized by jQuery UI?

4. How do you populate the possible values for an autocomplete element?

Quiz Answers

1. `$("#mySlider").slider("value");`
2. False.
3. The normal way you would with jQuery or JavaScript.
4. Set the value of the source option to a JavaScript array of values.

Exercises

1. Open the code in Listing 19.5 and add an Input element to the dialog box. Then add a second button that gets that value and displays it on the web page somewhere.
2. Open the code in Listing 19.10 and add a new tab called Sliders. Then have it link to the widgets_slider_bars.html file that contains the sliders example.

Part V: Building Web Applications with AngularJS

Lesson 20. Getting Started with AngularJS

What You'll Learn in This Lesson:

- The components of an AngularJS application
- The stages of the AngularJS life cycle
- How to bootstrap AngularJS in your webpages
- What functionality is provided in the AngularJS Global APIs
- How to build an AngularJS application

AngularJS is a JavaScript framework that provides a very structured method of creating websites and web applications. Essentially, AngularJS is a JavaScript library that is built on a lightweight version of jQuery—a combination that enables AngularJS to provide the best of JavaScript and jQuery and at the same time enforce a structured Model View Controller (MVC) framework.

AngularJS is a perfect client-side library for most web applications because it provides a very clean and structured approach. With a clean, structured front end, you will find that it is much easier to implement clean, well-structured server-side logic.

This lesson introduces you to AngularJS and the major components involved in an AngularJS application. Understanding these components is critical before you try to implement an AngularJS application, because the framework is different from more traditional JavaScript web application programming.

After you have a good grasp of the components and the life cycle of an AngularJS application, you'll learn how to construct a basic AngularJS application, step-by-step. This should prepare you to jump into the following lessons, which provide much more detail on implementing AngularJS.

Note

The examples in this book are based on AngularJS 1.x. Google has announced AngularJS 2 and given us some sneak peeks at what it will be like (although it seems like a release is still a long way out). AngularJS 2 will be a major upgrade from AngularJS 1. It appears the biggest changes that will affect developers will be around custom directives, which are the most complex part of AngularJS 1.x. The question is: Should I still learn AngularJS 1.x? The simple answer is yes. From what we can tell, the majority of the knowledge from AngularJS 1.x will translate to AngularJS 2 as the core concepts of code structure and separation of concern are the same. In the meantime, a good way to prep for AngularJS 2 is to keep your code as modularized as possible. Google has indicated that you will be able to run AngularJS 1.x wrapped inside AngularJS 2, and having modularized code will allow you to update a piece at a time.

Why AngularJS?

AngularJS is an MVC framework that is built on top of JavaScript and a lightweight version of jQuery. MVC frameworks separate the business logic in code from the view and the model. Without this separation, JavaScript-based web applications can quickly get out of hand when you are trying to manage all three together and a complex maze of functions.

Everything that AngularJS provides, you could implement yourself by using JavaScript and jQuery, or you could even try using another MVC JavaScript framework. However, AngularJS has a lot of functionality, and the design of the AngularJS framework makes it easy to implement MVC in the correct manner. The following are some of the reasons to choose AngularJS.

The AngularJS framework forces correct implementation of MVC and also makes it easy to implement MVC correctly:

- The declarative style of AngularJS HTML templates makes the intent of the HTML more intuitive and makes the HTML easier to maintain.
- The model portion of AngularJS is basic JavaScript objects, making it easy to manipulate, access, and implement.
- AngularJS uses a declarative approach to extend the functionality of HTML by having a direct link between the HTML declaratives and the JavaScript functionality behind them.
- AngularJS provides a very simple and flexible filter interface that enables you to easily format data as it passes from the model to the view.
- AngularJS applications tend to use a fraction of the code that traditional JavaScript applications use because you need to focus only on the logic and not all the little details, such as data binding.
- AngularJS requires a lot less Document Object Model (DOM) manipulation than traditional methods do and guides you to put the manipulations in the correct locations in applications. It is easier to design applications based on presenting data than on DOM manipulation.
- AngularJS provides several built-in services and enables you to implement your own in a structured and reusable way. This makes your code more maintainable and easier to test.
- Due to the clean separation of responsibilities in the AngularJS framework, it is easy to test your applications and even develop them using a test-driven approach.

Understanding AngularJS

AngularJS provides a very structured framework based on a Model View Controller (MVC) model. This framework enables you to build structured applications that are robust and easily understood and maintained. If you are not familiar with the MVC model, the following paragraph provides a quick synopsis to help you understand the basics. It is by no means complete and is intended only to give you enough reference to see how AngularJS applies MVC principles. The Wikipedia website is a great resource if you want additional information about MVC in general.

In MVC, there are three components: the Model is the data source, View is the rendered web page, and the Controller handles the interaction between the two. The idea is that when the model changes, the view is automatically updated with new data. As the user interacts with the view, the controller handles those interactions, which usually ends up in updating the model which then updates the view again.

A major purpose of MVC is to separate out responsibilities in your JavaScript code to keep it clean and easy to follow. AngularJS is one of the best MVC frameworks available because it makes it very easy to implement MVC

To get started with AngularJS, you first need to understand the various components that you will be implementing and how they interact with each other. The following sections discuss the various components involved in an AngularJS application, their purpose, and what each is responsible for.

Modules

AngularJS introduces the concept of a module representing components in an application. The module provides a namespace that enables you to reference directives, scopes, and other components based on model name. This makes it easier to package and reuse parts of an application.

Each view or web page in AngularJS has a single module assigned to it via the `ng-app` directive. (Directives are discussed later in this lesson.) However, you can add other modules to the main module as dependencies, which provides a very structured and componentized application. The main AngularJS module acts similar to the root namespace in C# and Java.

Scopes and the Data Model

AngularJS introduces the concept of a scope. A scope is just a JavaScript representation of data used to populate a view presented on a web page. The data can come from any source, such as a database, a remote web service, or the client-side AngularJS code, or it can be dynamically generated by the web server.

A great feature of scopes is that they are just plain JavaScript objects, which means you can manipulate them as needed in your AngularJS code with ease. Also, you can nest scopes to organize your data to match the context that they are being used in.

Views with Templates and Directives

HTML web pages are based on a DOM in which each HTML element is represented by a DOM object. A web browser reads the properties of a DOM object and knows how to render the HTML element on the web page, based on the DOM object's properties.

Most dynamic web applications use direct JavaScript or a JavaScript-based library such as jQuery to manipulate a DOM object to change the behavior and appearance of the rendered HTML element in the user view.

AngularJS introduces a new concept of combining templates that contain directives that extend the HTML tags and attributes directly with JavaScript code in the background to extend the capability of HTML. Directives have two parts. The first part is extra attributes, elements, and CSS classes that are added to an HTML template. The second part is JavaScript code that extends the normal behavior of the DOM.

The advantage of using directives is that the intended logic for visual elements is indicated by the HTML template so that it is easy to follow and is not hidden within a mass of JavaScript code. One of the best features of AngularJS is that the built-in AngularJS directives handle most of the necessary DOM manipulation functionality that you need to bind the data in the scope directly to the HTML elements in the view.

You can also create your own AngularJS directives to implement any necessary custom functionality you need in a web application. In fact, you should use your own custom directives to do any direct DOM manipulation that a web application needs.

Expressions

A great feature of AngularJS is the capability to add expressions inside the HTML template. AngularJS evaluates expressions and then dynamically adds the result to a web page. Because expressions are linked to the scope, you can have an expression that utilizes values in the scope, and as the model changes, so does the value of the expression.

Controllers

AngularJS completes the MVC framework through the implementation of controllers. Controllers augment the scope by setting up the initial state or values in the scope and by adding behavior to the scope. For example, you can add a function that sums values in a scope to provide a total so that if the model data behind the scope changes, the total value always changes.

You add controllers to HTML elements by using a directive and then implement them as

JavaScript code in the background.

Data Binding

One of the best features of AngularJS is the built-in data binding. Data binding is the process of linking data from the model with what is displayed in a web page. AngularJS provides a very clean interface to link the model data to elements in a web page.

In AngularJS, data binding is a two-way process: When data is changed on a web page, the model is updated, and when data is changed in the model, the web page is automatically updated. This way, the model is always the only source for data represented to the user, and the view is a projection of the model.

Services

Services are the major workhorses in the AngularJS environment. Services are singleton objects that provide functionality for a web app. For example, a common task of web applications is to perform AJAX requests to a web server. AngularJS provides an HTTP service that houses all the functionality to access a web server.

The service functionality is completely independent of context or state, so it can be easily consumed from the components of an application. AngularJS provides a lot of built-in service components for basic uses, such as HTTP requests, logging, parsing, and animation. You can also create your own services and reuse them throughout your code.

Dependency Injection

Dependency injection is a process in which a code component defines dependencies on other components. When the code is initialized, the dependent component is made available for access within the component. AngularJS applications make heavy use of dependency injection.

A common use for dependency injection is consuming services. For example, if you are defining a module that requires access to the web server via HTTP requests, you can inject the HTTP service into the module, and the functionality is available in the module code. In addition, one AngularJS module consumes the functionality of another via dependency.

Compiler

AngularJS provides an HTML compiler that will discover directives in the AngularJS template and use the JavaScript directive code to build out extended HTML elements. The AngularJS compiler is loaded into the browser when the AngularJS library is bootstrapped. When loaded, the compiler will search through the HTML DOM in the browser and link in any back-end JavaScript code to the HTML elements, and then the final application view will be rendered to the user.

An Overview of the AngularJS Life Cycle

Now that you understand the components involved in an AngularJS application, you need to understand what happens during the life cycle, which has three phases: bootstrap, compilation, and runtime. Understanding the life cycle of an AngularJS application makes it easier to understand how to design and implement your code.

The three phases of the life cycle of an AngularJS application happen each time a web page is loaded in the browser. The following sections describe these phases of an AngularJS application.

The Bootstrap Phase

The first phase of the AngularJS life cycle is the bootstrap phase, which occurs when the AngularJS JavaScript library is downloaded to the browser. AngularJS initializes its own necessary components and then initializes your module, which the `ng-app` directive points to. The module is loaded, and any dependencies are injected into your module and made available to code within the module.

The Compilation Phase

The second phase of the AngularJS life cycle is the HTML compilation stage. Initially when a web page is loaded, a static form of the DOM is loaded in the browser. During the compilation phase, the static DOM is replaced with a dynamic DOM that represents the AngularJS view.

This phase involves two parts: traversing the static DOM and collecting all the directives, and then linking the directives to the appropriate JavaScript functionality in the AngularJS built-in library or custom directive code. The directives are combined with a scope to produce the dynamic or live view.

The Runtime Data Binding Phase

The final phase of the AngularJS application is the runtime phase, which exists until the user reloads or navigates away from a web page. At that point, any changes in the scope are reflected in the view, and any changes in the view are directly updated in the scope, making the scope the single source of data for the view.

AngularJS behaves differently from traditional methods of binding data. Traditional methods combine a template with data received from the engine and then manipulate the DOM each time the data changes. AngularJS compiles the DOM only once and then links the compiled template as necessary, making it much more efficient than traditional methods.

Separation of Responsibilities

An extremely important part of designing AngularJS applications is the separation of responsibilities. The whole reason you choose a structured framework is to ensure that code is well implemented, easy to follow, maintainable, and testable. Angular provides a very structured framework to work from, but you still need to ensure that you implement AngularJS in the appropriate manner.

The following are a few rules to follow when implementing AngularJS:

- The view acts as the official presentation structure for the application. Indicate any presentation logic as directives in the HTML template of the view.
- If you need to perform any DOM manipulation, do it in a built-in or your own custom directive JavaScript code—and nowhere else.
- Implement any reusable tasks as services and add them to your modules by using dependency injection.
- Ensure that the scope reflects the current state of the model and is the single source for data consumed by the view.
- Ensure that the controller code acts only to augment the scope data and doesn't include any business logic.
- Define controllers within the module namespace and not globally. This ensures that your application can be packaged easily and prevents overwhelming the global namespace.

Integrating AngularJS with Existing JavaScript and jQuery

The fact that AngularJS is based on JavaScript and jQuery makes it tempting to try to add it to existing applications to provide data binding or other functionality. That approach will almost always end up in problem code that is difficult to maintain. However, using AngularJS doesn't mean that you need to simply toss out your existing code, either. Often you can selectively take working JavaScript/jQuery components and convert them to either directives or services.

This also brings up another issue: when to use the full version of jQuery rather than the jQuery lite version that is provided with AngularJS? I know that many people have strong views in both directions. On one hand, you want to keep your implementation as clean and simple as possible. But on the other hand, there might be times when you need functionality that's available only in the full version of jQuery. Our advice is to use what makes sense. If you need functionality that is not provided with AngularJS jQuery lite, load the full library. The mechanics of loading jQuery rather than jQuery lite are discussed later in this lesson.

The following steps suggest a method to integrate AngularJS into your existing JavaScript and jQuery applications:

1. Write at least one small AngularJS application from the ground up that uses a model, custom HTML directives, services, and controllers. In other words, in this application, ensure that you have a practical comprehension of the AngularJS separation of responsibilities.
2. Identify the model portion of your code. Specifically, try to separate out the code that augments the model data in the model into controller functions and code that accesses the back-end model data into services.
3. Identify the code that manipulates DOM elements in the view. Try to separate out the DOM manipulation code into well-defined custom directive components and provide an HTML directive for them. Also identify any of the directives for which AngularJS already provides built-in support.
4. Identify other task-based functions and separate them out into services.
5. Isolate the directives and controllers into modules to organize your code.
6. Use dependency injection to link up your services and modules appropriately.
7. Update the HTML templates to use the new directives.

Obviously, in some instances, it doesn’t make sense to use much, if any, of your existing code. However, by running through the preceding steps, you will get well into the design phase of implementing a project using AngularJS and can then make an informed decision.

Adding AngularJS to Your Environment

AngularJS is a client-side JavaScript library, which means the only thing you need to do to implement AngularJS in your environment is to provide a method for the client to get the `angular.js` library file by using a `<script>` tag in the HTML templates.

The simplest method of providing the `angular.js` library is to use the Content Delivery Network (CDN), which provides a URL for downloading the library from a third party. The downside of this method is that you must rely on a third party to serve the library, and if the client cannot connect to that third-party URL, your application will not work. For example, the following `<script>` tag loads the `angular.js` library from Google APIs CDN:

Click here to view code image

```
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.5/angular.min.js">
</script>
```

The other method of providing the `angular.js` library is to download it from the

AngularJS website (http://angularjs.org) and use your own web server to serve the file to the client. This method takes more effort and also requires extra bandwidth on your web server; however, it might be a better option if you want more control over how the client obtains the library.

Bootstrapping AngularJS in an HTML Document

To implement AngularJS in your web pages, you need to bootstrap the HTML document. Bootstrapping involves two parts. The first part is to define the application module by using the `ng-app` directive, and the second is to load the `angular.js` library in a `<script>` tag.

The `ng-app` directive tells the AngularJS compiler to treat that element as the root of the compilation. The `ng-app` directive is typically loaded in the `<html>` tag to ensure that the entire web page is included; however, you could add it to another container element, and only elements inside that container would be included in the AngularJS compilation and consequently in the AngularJS application functionality.

When possible, you should include the `angular.js` library as one of the last tags, if not the last tag, inside the `<body>` of the HTML. When the `angular.js` script is loaded, the compiler kicks off and begins searching for directives. Loading `angular.js` last allows the web page to load faster.

The following is an example of implementing the `ng-app` and `angular.js` bootstrap in an HTML document:

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <body>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="/lib/myApp.js"></script>
  </body>
</html>
```

Using the Global APIs

As you are implementing AngularJS applications, you will find that there are common JavaScript tasks that you need to perform regularly, such as comparing objects, deep copying, iterating through objects, and converting JSON data. AngularJS provides a lot of this basic functionality in the global APIs.

The global APIs are available when the `angular.js` library is loaded, and you can access them by using the `angular` object. For example, to create a copy of an object named `myObj`, you use the following syntax:

Click here to view code image

```
var myCopy = angular.copy(myObj);
```

The following code shows an example of iterating through an array of objects by using the `forEach()` global API:

Click here to view code image

```
var objArr = [{score: 95}, {score: 98}, {score: 92}];
var scores = [];
angular.forEach(objArr, function(value, key){
  this.push(key + '=' + value);
}, scores);
// scores == ['score=95', 'score=98', 'score=92']
```

Table 20.1 lists some of the most useful utilities provided in the global APIs. You will see these used in a number of examples in this book.

Utility	Description
`copy(src, [dst])`	Creates a deep copy of the `src` object or array. If a `dst` parameter is supplied, it is completely overwritten by a deep copy of the source.
`element(element)`	Returns the DOM element specified as a jQuery element. If you have loaded jQuery before loading AngularJS, the object is a full jQuery object; otherwise, it is only a subset of a jQuery object, using the jQuery lite version built in to AngularJS. Table 20.2 lists the jQuery lite methods available in AngularJS.
`equals(o1, o2)`	Compares `o1` with `o2` and returns `true` if they pass an `===` comparison.
`extend(dst, src)`	Copies all the properties from the `src` object to the `dst` object.
`forEach(obj, iterator, [context])`	Iterates through each object in the `obj` collection, which can be an object or an array. The iterator specifies a function to call, using the following syntax: `function(value, key)` The `context` parameter specifies a JavaScript object that acts as the context, accessible via the `this` keyword, inside the `forEach` loop.

`fromJson(json)`	Returns a JavaScript object from a JSON `string`.
`toJson(obj)`	Returns a JSON string form of the JavaScript object `obj`.
`isArray(value)`	Returns `true` if the `value` parameter passed in is an `Array` object.
`isDate(value)`	Returns `true` if the `value` parameter passed in is a `Date` object.
`isDefined(value)`	Returns `true` if the `value` parameter passed in is a defined object.
`isElement(value)`	Returns `true` if the `value` parameter passed in is a DOM element object or a jQuery element object.
`isFunction(value)`	Returns `true` if the `value` parameter passed in is a JavaScript function.
`isNumber(value)`	Returns `true` if the `value` parameter passed in is a number.
`isObject(value)`	Returns `true` if the `value` parameter passed in is a JavaScript object.
`isString(value)`	Returns `true` if the `value` parameter passed in is a `String` object.
`isUndefined(value)`	Returns `true` if the `value` parameter passed in is not defined.
`lowercase(string)`	Returns a lowercase version of the `string` parameter.
`uppercase(string)`	Returns an uppercase version of the `string` parameter.

TABLE 20.1 Useful Global API Utilities Provided in AngularJS

Try it Yourself: Creating a Basic AngularJS Application

Now that you understand the basic components in the AngularJS framework, the intent and design of the AngularJS framework, and how to bootstrap AngularJS, you are ready to get started implementing an AngularJS application. This section walks you through a very basic AngularJS application that implements an HTML template, an AngularJS module, a controller, a scope, and an expression.

The code for this example is an AngularJS HTML template, `first.html` shown in Listing 20.1, and an AngularJS JavaScript module, `first.js` shown in Listing 20.2. The following steps describe the important steps in implementing this code for the AngularJS application.

Each of these steps is described in more detail in later lessons, so don’t get bogged down in the details here. What is important at this point is that you understand the process of implementing the template, module, controller, and scope and generally how they interact with each other:

1. In Eclipse, add the lesson20/first.html and lesson18/js/first.js files.
2. In the first.html file, add the standard web page elements for doctype, html, head, and body.

3. Then you need to get the library loaded in an HTML template. Add the following lines shown in Listing 20.1 that load the `angular.js` library and then load the `first.js` JavaScript custom module:

Click here to view code image

```
<script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
<script src="/js/first.js"></script>
```

4. The next step is to define the `ng-app` parameter in the root element for the application so that AngularJS knows where to begin compiling. Add the following line shown in Listing 20.1 that bootstraps AngularJS in the `<html>` element. Notice that `ng-app` is assigned the module name `firstApp`, which corresponds to the module in the JavaScript code shown in Listing 20.2:

```
<html ng-app="firstApp">
```

5. Add the following line to the first.js file shown in Listing 20.2 to define the AngularJS application module with the name `firstApp`:

Click here to view code image

```
var firstApp = angular.module('firstApp', []);
```

6. Add a controller for HTML elements that you want the AngularJS module to control. Add the following line shown in Listing 20.1 that assigns a controller named `FirstController` to a `<div>` element. This maps the element in the view to a specific controller, which contains a scope:

Click here to view code image

```
<div ng-controller="FirstController">
```

7. Define the controller in your module code by adding the following lines shown in Listing 20.2 that define `FirstController` in the `firstApp` module:

Click here to view code image

```
firstApp.controller('FirstController', function($scope) {
});
```

8. After the controller has been defined, you can implement the scope, which involves linking HTML elements to scope variables, initializing the variables in the scope, and providing functionality to handle changes to the scope values. Add the following lines in Listing 20.1 to define two `<input>` elements that are bound to the `first` and `last` values in the scope using the `ng-model` directive. When the value of the input changes, so does the value of `first` and `last` in the scope and vice versa:

Click here to view code image

```
      <input type="text" ng-model="first">
      <input type="text" ng-model="last">
```

9. Add the following lines to Listing 20.2 to define the initial values of `first`, `last`, and `heading` in the scope:

Click here to view code image

```
$scope.first = 'Some';
$scope.last = 'One';
$scope.heading = 'Message: ';
```

10. Add the following line that will define a button in the HTML template that uses `ng-click` to bind the mouse button click event to the function `updateMessage()` function that will be defined in the scope:

Click here to view code image

```
      <button ng-click='updateMessage()'>Message</button>
```

11. Define the `updateMessage()` function in the controller code shown in Listing 20.2 by adding the following lines. Notice that the variable `message` in the scope is set to a combination of the values of `first` and `last`:

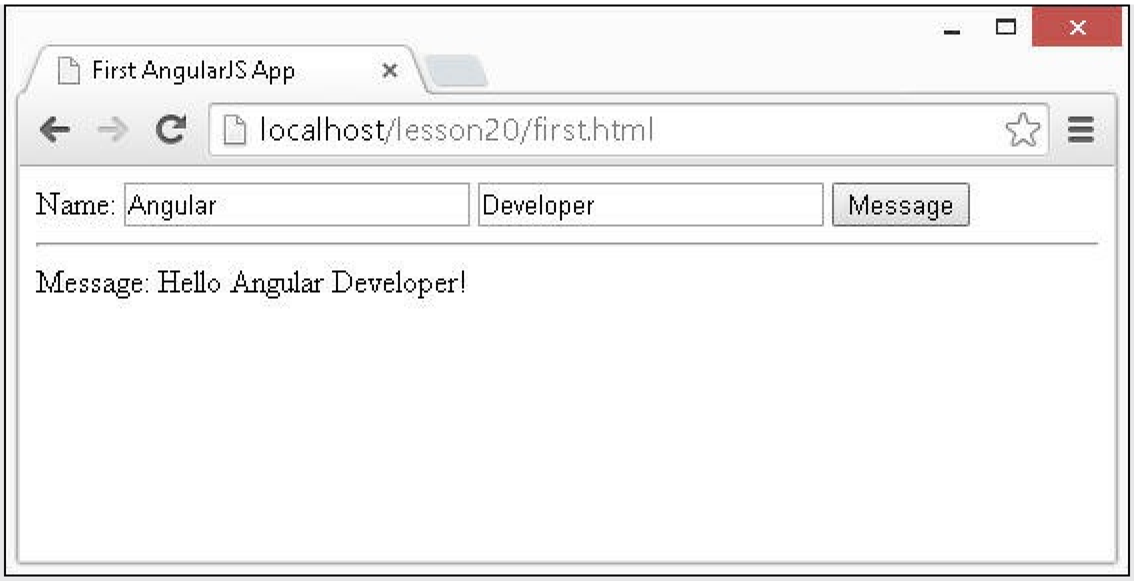
Click here to view code image

```
$scope.updateMessage = function() {
  $scope.message = 'Hello ' + $scope.first +' '+ $scope.last + '!';
};
```

12. Add the following line in the template file shown in Listing 20.1 that implements an expression that displays the value of the `heading` and `message` variables from the scope on the HTML page:

```
      {{heading + message}}
```

13. Now test the AngularJS application by starting the web server if necessary and loading first.html in a browser. You should be able to type strings into the text box inputs and then click the Message button and see the dynamic message created by the AngularJS application displayed, as shown in Figure 20.1.



Name: Angular Developer

Message: Hello Angular Developer!

FIGURE 20.1 Implementing a basic AngularJS web application that uses inputs and a button to manipulate the model and consequently the view.

LISTING 20.1 first.html A Simple AngularJS Template That Provides Two Input Elements and a Button to Interact with the Model

Click here to view code image

```
<!doctype html>
<html ng-app="firstApp">
  <head>
    <title>First AngularJS App</title>
  </head>
  <body>
    <div ng-controller="FirstController">
      <span>Name:</span>
      <input type="text" ng-model="first">
      <input type="text" ng-model="last">
      <button ng-click='updateMessage()'>Message</button>
      <hr>
      {{heading + message}}
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/first.js"></script>
  </body>
</html>
```

LISTING 20.2 first.js A Simple AngularJS Module That Implements a Controller to Support the Template in Listing 20.1

Click here to view code image

```
var firstApp = angular.module('firstApp', []);
firstApp.controller('FirstController', function($scope) {
  $scope.first = 'Some';
  $scope.last = 'One';
  $scope.heading = 'Message: ';
  $scope.updateMessage = function() {
    $scope.message = 'Hello ' + $scope.first +' '+ $scope.last + '!';
  };
});
```

Using jQuery or jQuery Lite in AngularJS Applications

You will be using at least jQuery lite in your AngularJS applications, so it is important to understand the interactions between jQuery, jQuery lite, and AngularJS. Even if you are not a jQuery developer, understanding these interactions will help you write better AngularJS applications. If you are a jQuery developer, understanding the interactions will enable you to leverage your jQuery knowledge in your AngularJS applications.

The following sections describe jQuery lite implementation and provide a brief introduction to the jQuery/jQuery lite interactions that you will be seeing in your AngularJS applications. The following lessons will expand on this topic as you see some practical examples that utilize jQuery objects in AngularJS applications.

What Is jQuery Lite?

jQuery lite is a stripped-down version of jQuery that is built directly into AngularJS. The intent is to provide all the useful features of jQuery and yet keep it constrained within the AngularJS separation of responsibilities paradigm.

Table 20.2 lists the jQuery methods available in jQuery lite along with any restrictions that might apply. The restrictions are necessary to enforce things like manipulating elements only within a custom directive, and so on.

jQuery Method	Limitations, If Any, in jQuery Lite
`addClass()`	
`after()`	
`append()`	
`attr()`	
`bind()`	Does not support namespaces, selectors, or eventData.
`children()`	Does not support selectors.
`clone()`	
`contents()`	
`css()`	
`data()`	
`detach()`	
`empty()`	
`eq()`	
`find()`	Limited to lookups by tag name.
`hasClass()`	
`html()`	
`text()`	Does not support selectors.

`on()`	Does not support namespaces, selectors, or eventData.
`off()`	Does not support namespaces or selectors.
`one()`	Does not support namespaces or selectors.
`parent()`	Does not support selectors.
`prepend()`	
`prop()`	
`ready()`	
`remove()`	
`removeAttr()`	
`removeClass()`	
`removeData()`	
`replaceWith()`	
`toggleClass()`	
`triggerHandler()`	Passes a dummy event object to handlers.
`unbind()`	Does not support namespaces.
`val()`	
`wrap()`	

TABLE 20.2 jQuery Methods That Are Supported in jQuery Lite

Table 20.3 lists the additional events and methods that AngularJS adds to jQuery lite objects.

Method/Event	Description
`$destroy`	AngularJS intercepts all jQuery or jQuery lite DOM destruction calls and fires this event on all DOM nodes being removed. This can be used to clean up any third-party bindings to the DOM element before it is removed.
`controller(`*`name`*`)`	Returns the `controller` object of the current element or its parent. If no `name` is specified, the controller associated with the `ngController` directive is returned. If a `name` is provided as a directive name, the controller for this directive is returned.
`injector()`	Returns the `injector` object of the current element or its parent.
`scope()`	Returns the `scope` object of the current element or its parent.
`isolateScope()`	Returns an isolate `scope` object if one is attached directly to the current element. This works only on elements that contain a directive that starts a new isolate scope.
`inheritedData()`	Works the same as the jQuery `data()` method, but walks up the DOM until a value is found or the top parent element is reached.

TABLE 20.3 Methods and Events Added to jQuery Lite Objects

Accessing jQuery or jQuery Lite Directly

For most AngularJS applications, the jQuery lite library built in to AngularJS is sufficient. However, if you need the additional functionality of the full version of jQuery, load the jQuery library before loading the AngularJS library. For example:

Click here to view code image

```
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
<script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
```

By the Way

If you decide to use the full version of jQuery with AngularJS, make certain that you load the jQuery library `<script>` tag before the AngularJS library `<script>` tag in your HTML. Otherwise, AngularJS will not be properly bound to the jQuery library.

Regardless of whether jQuery lite or the full jQuery library is loaded, jQuery is accessed from the AngularJS code using the `element` attribute of the `angular` variable available when AngularJS is bootstrapped. Essentially, `angular.element` will be an alias for the `jQuery` variable that is normally used in jQuery applications. One of the best ways this relationship is described is as follows:

Click here to view code image

```
   angular.element() === jQuery() === $()
```

More often than not, you will be using the jQuery or jQuery lite functionality in jQuery objects that AngularJS creates for you.

For example, when you create a directive in AngularJS as discussed later in this book, an element is passed to the link function. That element, as shown here, is a jQuery or jQuery lite object, and you can use the jQuery functionality accordingly:

Click here to view code image

```
   angular.module('myApp', [])
    .directive('myDirective', function() {
         . . .
         link: function(scope, elem, attrs, photosControl) {
           //elem is a jQuery lite object
           elem.addClass(...);
         }
       };
```

Another example of accessing the jQuery functionality is from events that are triggered on AngularJS bindings. For example, consider the following code that uses the `ngClick` binding to bind a browser click event on a `<div>` element to a `clicked()` function in the AngularJS code:

Click here to view code image

```
   <div ng-click="clicked($event)">Click Me</div>
```

You can access a jQuery version of the object using the following AngularJS code:

Click here to view code image

```
      $scope.clicked = function(event){
        var jQueryElement = angular.element(event.target);
      };
```

Note that it was necessary to use the `angular.element()` method to convert the `target` DOM object into a jQuery object.

Summary

AngularJS is a JavaScript library framework that provides a very structured method for creating websites and web applications. AngularJS structures a web application into a very clean MVC-styled approach. AngularJS scopes provide contextual binding to the data model for the application and are made up of basic JavaScript objects. AngularJS utilizes templates with directives that extend HTML capabilities, enabling you to implement totally customized HTML components.

In this lesson, you looked at the different components in an AngularJS application and how they interact with each other. You also learned about the life cycle of an AngularJS application, which involves bootstrap, compilation, and runtime phases. At the end of this lesson, you walked through a step-by-step example of implementing a basic AngularJS application, including a template, module, controller, and scope.

Q&A

Q. Where can I go to learn more about developing AngularJS applications?

A. The AngularJS developer guide at https://docs.angularjs.org/guide provides good resources on AngularJS development topics.

Q. Is there a way to build end-to-end testing of AngularJS applications?

A. You can use Protractor, a Node.js program, to run end-to-end tests. This is described at https://docs.angularjs.org/guide/e2e-testing in the developer guide.

Workshop

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

Quiz

1. When using the full version of jQuery, should the jQuery library be loaded before or after AngularJS library?
2. What are the three phases of the AngularJS life cycle?
3. When do global APIs become available?
4. Which AngularJS directive is used to bootstrap the AngularJS application in your HTML file?

Quiz Answers

1. Before the AngularJS library.
2. Bootstrapping, Compilation, and Runtime Data Binding.
3. When the angular.js library is loaded.
4. `ng-app`

Exercises

1. Load up the current AngularJS documentation by going to https://docs.angularjs.org/api and review some of the global API functions, such as `angular.copy` and `angular.element`. This will familiarize you with

using the API documentation.

2. Load up the current index of versions of AngularJS by going to https://code.angularjs.org/ so that you can see which versions of AngularJS are the latest and know where to load them from.

Lesson 21. Understanding AngularJS Application Dynamics

What You'll Learn in This Lesson:

- How AngularJS uses dependency injection
- How to create a dependency provider in AngularJS
- What the configuration block of an AngularJS application is used for
- What the run block of an AngularJS application is used for

One of the most important aspects of AngularJS to understand is dependency injection and how it relates to modules. Dependency is a common concept across many server-side languages but has not really been implemented much in JavaScript until AngularJS.

Dependency injection allows AngularJS modules to maintain a very clean, organized form yet more easily access functionality from other modules. When implemented correctly, it also tends to reduce the amount of code by a considerable amount.

This lesson provides a basic overview of dependency injection and then describes how to create modules that provide functionality and how to consume that functionality in other modules as well as other AngularJS components, such as controllers.

Looking at Modules and Dependency Injection

As you begin writing AngularJS applications, it is vital that you understand the basics of modules and dependency injection in the AngularJS world. This seems to be a difficult concept to grasp and implement correctly for some, especially those coming from a more open, anything-goes JavaScript background.

This section introduces you to the concepts behind AngularJS modules and dependency injection. Understanding how modules utilize dependency injection to access functionality in other modules will make it easier for you to implement your code inside the AngularJS framework.

Understanding Modules

AngularJS modules are containers that enable you to compartmentalize and organize your code into concise, clean, reusable chunks. Modules themselves do not provide direct functionality, but they contain instances of other objects that do, such as controllers, filters, services, and animations.

You build a module by defining the objects it provides. Then, by linking together modules through dependency injection, you build a full application.

AngularJS is built on the module principle. Most of the functionality provided by

AngularJS is built in to a module named `ng`, which contains most of the directives and services used throughout this book.

Dependency Injection

Dependency injection can be a difficult concept to fully grasp. However, it is a very important part of AngularJS, and after you understand the basics, the AngularJS implementation becomes quite clear. Dependency injection is a well-known design pattern in most server-side languages but has not been used extensively in a JavaScript framework until AngularJS.

The idea of AngularJS dependency injection is to define and dynamically inject a dependency object into another object, making available all the functionality provided by the dependency object.

An analogy I like to use for dependency injection is the virtual room. If you just wanted to create a virtual room, you would define four walls, a floor, and a ceiling as the basics. However, that room is not very functional for most purposes. If you want the room to be a dining room, you would need a table and dining chairs; however, if you want the room to be a living room, you may want a sofa and TV.

So in the virtual world, you would define a way to create a sofa, dining chair, dining table, TV, lamps, pictures, end table, and so on. Keep in mind that this is simply a way to create these objects, not an actual instance of a sofa or table object.

Then, dependency injection allows you to easily define any type of room you may want by stating which types of objects should be injected into the room when it is created.

For example, you could define something like `LivingRoom` equals `[sofa, table, TV]` injected into `Room`. This is a definition only, so there is not an instance of `Room`, `sofa`, `table`, or `TV` until you create an instance `LivingRoom`, at which point the `sofa`, `table`, and `TV` objects would be created and automatically injected into the newly created `LivingRoom` to provide the living room functionality you need to sit down and watch a virtual movie.

AngularJS provides dependency injection through the use of providers and an injector service. The providers define how to create the injectable objects with certain functionality. The injector service creates an instance of the injectable objects and makes it available in newly created objects that use them.

Providers

A provider is essentially a definition of how to create an instance of an object with all the necessary functionality. Providers should be defined as part of an AngularJS module. A module registers the provider with the injector server. Only one instance of a provider's object is ever created in the AngularJS application.

Injectors

The injector service is responsible for keeping track of instances of provider objects. An injector service instance is created for each module that registers a provider.

When an object that requires a provider is created, a dependency request is made for an instance of the provider object. The injector service first checks whether an instance already exists in the injector cache. If so, that instance is returned. If no instance is found in the cache, a new instance is created using the provider definition; it is stored in the cache, and then returned.

Defining an AngularJS Module Object

Creating AngularJS modules is a simple process that involves calling the `angular.module()` method. This method creates an instance of a `Module` object, registers it with the injector service, and then returns an instance of the newly created `Module` object that you can use to implement provider functionality. The `angular.module()` method uses the following syntax:

Click here to view code image

```
angular.module(name, [requires], [configFn])
```

The `name` parameter is the name under which the module is registered in the injector service. The `requires` parameter is an array of names of modules that are added to the injector service for this module to use. If you need functionality from another module, you need to add it in the `requires` list. The `ng` module is automatically added to every module instantiated by default, so you have access to the AngularJS providers without explicitly specifying `ng` in the list.

Instances of all dependencies are automatically injected into an instance of a module. Dependencies can be modules, services, and any other objects registered in the injector service. The `configFn` parameter is another function that is called during the module configuration phase. Configuration functions are described in the next section.

The following is an example of creating an AngularJS module with dependencies on the `$window` and `$http` services. The definition also includes a configuration function that adds a `value` provider named `myValue`:

Click here to view code image

```
var myModule = angular.module('myModule', ['$window', '$http'], function()
{
    $provide.value('myValue', 'Some Value');
});
```

If you do not specify a `requires` parameter, instead of a `Module` object being created, the already created instance is returned. For example, the following code overwrites the instance defined previously:

Click here to view code image

```
var myModule2 = angular.module('myModule', []);
```

However, the following code returns the instance created previously because no dependencies are listed in the `require` array in the parameters list:

Click here to view code image

```
var myModule3 = angular.module('myModule');
```

Creating Providers in AngularJS Modules

AngularJS provides a number of built-in providers for various objects and services. For example, the `$window` service has a provider that builds the AngularJS service object that enables you to interact with the underlying `Window` object in JavaScript. In addition to these providers, you can create providers of your own to inject functionality into AngularJS application components.

The `Module` object provides several helper methods for adding providers as an alternative to using the `config()` method. These methods are simpler to use and clearer in your code. You can add two types of provider objects to AngularJS modules. Each of these methods accepts two parameters: the name that will be registered with the dependency injector and the provider function that defines how to build the specific object. The following sections describe these methods in more detail.

Specialized AngularJS Object Providers

The `Module` object provides special constructor methods to add providers for the AngularJS objects that you need to implement in your modules. These specialized methods enable you to add definitions for the following types of objects:

- `animation(name, animationFactory)`
- `controller(`name, controllerFactory`)`
- `filter(`name, filterFactory`)`
- `directive(name, directiveFactory)`

These are specialized methods because there are corresponding `animation`, `controller`, `filter`, and `directive` objects defined in AngularJS for these provider methods.

Each of these objects is covered in more detail in later lessons. For now, here’s a quick look at a basic controller definition:

Click here to view code image

```
var mod = angular.module('myMod', []);
```

```
mod.controller('myController', function($scope) {
  $scope.someValue = 'Some Value';
});
```

A simple module named `mod` is created, and then the `controller()` method is called and passed in `myController` along with a `controllerFactory` function. The `controllerFactory` function accepts the `$scope` variable as a parameter. This is because AngularJS has a built-in controller object and knows that all controller objects must receive a scope object as the first parameter.

Service Providers

The service providers are a unique category of providers because there is not already a specific format for the resulting provider objects. Instead, a provider acts as a service to provide functionality. AngularJS provides some specific creation methods for building services and exposes them through the following methods:

- **value(name, object):** This is the most basic of all providers. The `object` parameter is simply assigned to `name`, so a direct correlation exists in the injector between the `name` value and the `object` value.
- **constant(name, object):** This is similar to the `value()` method, but the value is not changeable. Also, `constant()` methods are applied before other provider methods.
- **factory(name, factoryFunction):** This method uses the `factoryFunction` parameter to build an object that will be provided by the injector.
- **service(name, serviceFactory):** This method adds the concept of implementing a more object-oriented approach to the provider object. Much of the built-in functionality of AngularJS is provided through service providers.
- **provider(name, providerFactory):** This method is the core for all the other methods. Although it provides the most functionality, it is not used frequently because the other methods are simpler.

Later lessons cover these objects in more detail. For now, here’s a quick example of some basic `value` and `constant` definitions:

[Click here to view code image]

```
var mod = angular.module('myMod', []);
mod.constant("cID", "ABC");
mod.value('counter', 0);
mod.value('image', {name:'box.jpg', height:12, width:20});
```

A simple module named `mod` is created, and then the `constant()` and two `value()` providers are defined. The values defined in these methods are registered in the injector server for the `myMod` module and are then accessible by name.

Implementing Providers and Dependency Injection

After you have defined a module and appropriate providers, you can add the module as a dependency to other modules, controllers, and various other AngularJS objects. You can set the value of the `$inject` property of the object that depends on the providers. The `$inject` property contains an array of provider names that should be injected into it.

For example, the following code defines a basic controller that accepts the `$scope` and `appMsg` parameters. Then the `$inject` property is set to an array that contains `$scope`, which is the AngularJS scope service that provides access to the scope and a custom `appMsg`. Both `$scope` and `appMsg` are injected into the `myController` function:

Click here to view code image

```
var myController = function($scope, appMsg) {
  $scope.message = appMsg;
};
controller['$inject'] = ['$scope', 'appMsg'];
myApp.myController('controllerA', controller);
```

This method can become a bit clumsy when you're implementing certain objects, so AngularJS also provides a more elegant method for injecting the dependencies, using the following syntax in place of the normal constructor function:

Click here to view code image

```
[providerA, providerB, . . ., function(objectA, objectB, . . .) {} ]
```

For example, the preceding code can also be written this way:

Click here to view code image

```
myApp.controller('controllerA', ['$scope', 'appMsg', function($scope,
appMsg) {
  $scope.message = appMsg;
}]);
```

It is critical that you understand dependency injection before continuing, so the following sections provide sample listings that implement dependency injection in AngularJS applications.

Try it Yourself: Injecting a Built-In Provider into a Controller

In this section, you learn how to inject some of the providers that are built into AngularJS into your controllers. The code shown in Listing 21.1 implements the basic controller that is injected with the `$scope` and `$window` providers. The code in Listing 22.2 shows the HTML template that loads the AngularJS application.

The following describes the important aspects of using dependency injection to add functionality from AngularJS providers to a controller:

1. Add the lesson21/inject_builtin.html and lesson21/js/inject_builtin.js files.
2. Add the code shown in Listing 21.1 and Listing 21.2 to the HTML and JavaScript files.
3. The following lines of code define a controller that uses the `$scope` and `$window` providers built in to AngularJS. Notice that in the `function()` definition, the parameter names are also `$scope` and `$window`. That is not necessary, but it may make it easier to follow the code:

Click here to view code image

```
myMod.controller('controllerA', ['$scope', '$window',
                                  function($scope, $window) {
```

4. The following line of code in Listing 21.1 uses the `$scope` object created by the `$scope` provider to set a value in controller's scope:

Click here to view code image

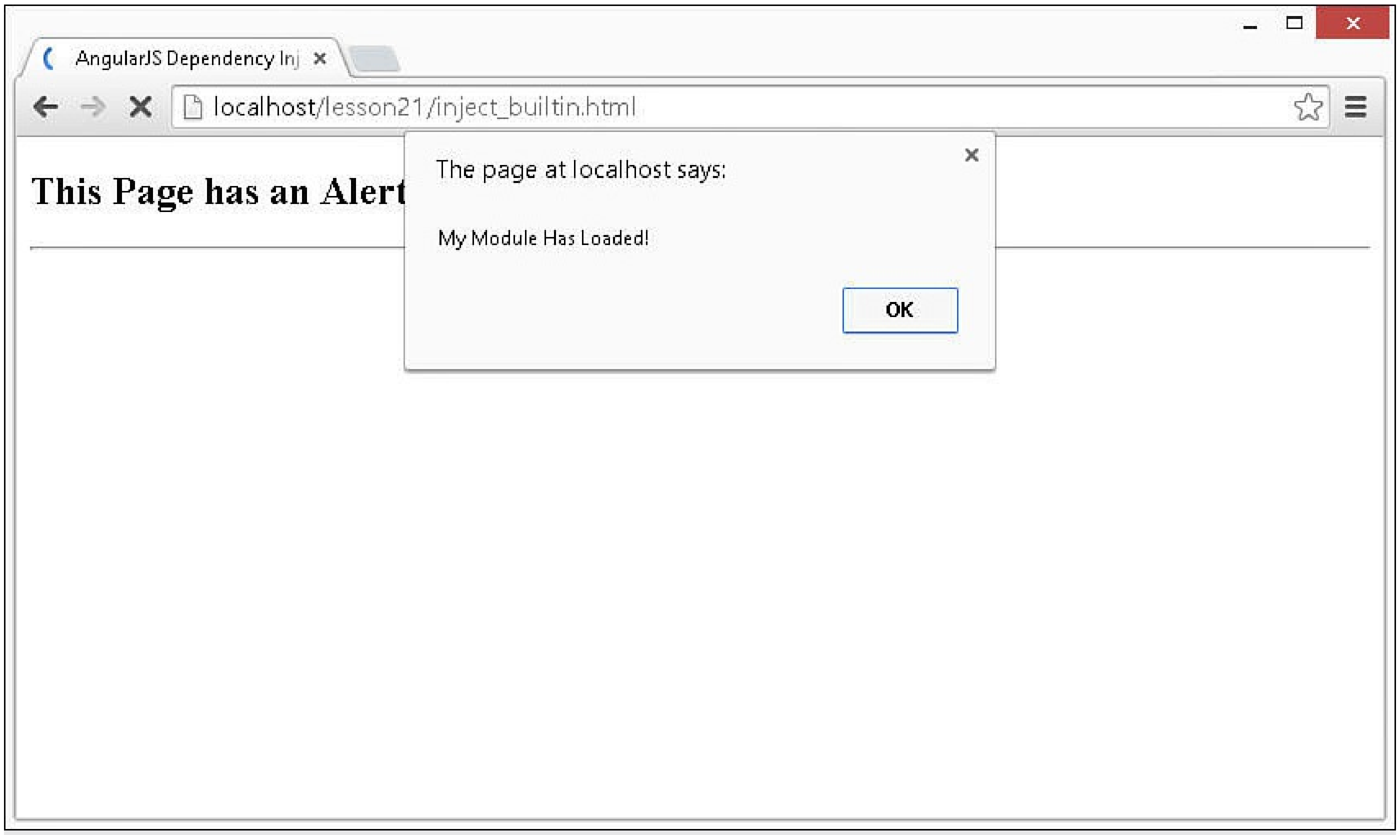
```
  $scope.message = "My Module Has Loaded!";
```

5. The following line in Listing 21.2 uses the `$window` object injected into the controller to display an alert in the browser:

Click here to view code image

```
  $window.alert($scope.message);
```

6. When you load inject_builtin.html in a browser, you should see a page similar to Figure 21.1 with the resulting web page and alert message.


AngularJS Dependency Inj

localhost/lesson21/inject_builtin.html

This Page has an Alert

The page at localhost says:

My Module Has Loaded!

OK

FIGURE 21.1 Implementing dependency injection of the `$window` service to display an alert message from the `$scope`.

LISTING 21.1 inject_builtin.js Implementing Dependency Injection of Built-in Services in a Controller

[Click here to view code image]

```
var myMod = angular.module('myApp', []);
myMod.controller('controllerA', ['$scope', '$window',
                                function($scope, $window) {
  $scope.message = "My Module Has Loaded!";
  $window.alert($scope.message);
}]);
```

LISTING 21.2 inject_builtin.html Using HTML Code to Implement an AngularJS Module That Implements Dependency Injection

[Click here to view code image]

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Dependency Injection</title>
  </head>
  <body>
    <div ng-controller="controllerA">
      <h2>This Page has an Alert</h2>
    </div><hr>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/inject_builtin.js"></script>
  </body>
</html>
```

Try it Yourself: Injecting One Module into Another Module

In this example, you learn how to inject the functionality of one module into another module. The code in Listing 21.3 implements two modules, `myMod` and `myApp`, and then uses dependency injection to inject the functionality of `myMod` into `myApp`. Listing 21.4 shows HTML that implements the `myApp` module as the AngularJS application. Notice that it uses both the `controllerA` and the `controllerB` controllers. They can be used because the `myMod` module was injected into the `myApp` module.

Use the following steps to build the modules and implement dependency injection:

1. Add the lesson21/inject_custom.html and lesson21/js/inject_custom.js files.

2. Add the code shown in Listing 21.3 and Listing 21.4 to the HTML and JavaScript files.

3. The following line of code defines the `myMod` module:

Click here to view code image

```
var myMod = angular.module('myMod', []);
```

4. The following line of code defines a value provider named `modMsg` for `myMod` that contains a simple string:

Click here to view code image

```
myMod.value('modMsg', 'Hello from My Module');
```

5. The following lines of code define `controllerB` and inject the `$scope` and `modMsg` providers into it and then sets the `message` value that will be displayed in the template:

Click here to view code image

```
myMod.controller('controllerB', ['$scope', 'modMsg',
                                 function($scope, msg) {
  $scope.message = msg;
06 }]);
```

6. Then the following line of code defines `myApp` with `myMod` as an injectable provider. The rest of the lines are similar to the definition of the `value` provider and controller defined in the preceding steps:

Click here to view code image

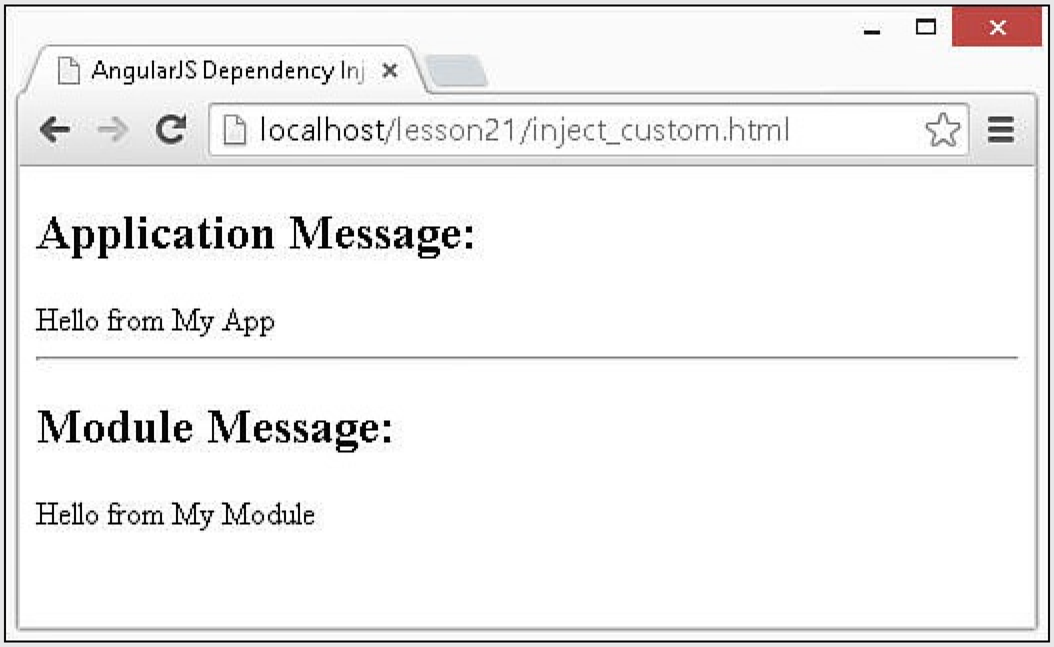
```
07 var myApp = angular.module('myApp', ['myMod']);
```

7. Notice in the following lines from Listing 21.4 that only the `myApp` module is bootstrapped but that `controllerB` from `myMod` is able to be used because it is injected into `myApp`:

Click here to view code image

```
02 <html ng-app="myApp">
. . .
07     <div ng-controller="controllerA">
. . .
11     <div ng-controller="controllerB">
```

8. Figure 21.2 shows the resulting web page, with a different message from each module's controller.



Application Message:

Hello from My App

Module Message:

Hello from My Module

FIGURE 21.2 Implementing dependency injection to provide additional functionality to modules and controllers.

LISTING 21.3 inject_custom.js Implementing Dependency Injection in Controller and Module Definitions

Click here to view code image

```
var myMod = angular.module('myMod', []);
myMod.value('modMsg', 'Hello from My Module');
myMod.controller('controllerB', ['$scope', 'modMsg',
                                function($scope, msg) {
  $scope.message = msg;
}]);
var myApp = angular.module('myApp', ['myMod']);
myApp.value('appMsg', 'Hello from My App');
myApp.controller('controllerA', ['$scope', 'appMsg',
                                function($scope, msg) {
  $scope.message = msg;
}]);
```

LISTING 21.4 inject_custom.html Using HTML Code to Implement an AngularJS Module That Depends on Another Module

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Dependency Injection</title>
  </head>
  <body>
    <div ng-controller="controllerA">
      <h2>Application Message:</h2>
      {{message}}
    </div><hr>
    <div ng-controller="controllerB">
      <h2>Module Message:</h2>
      {{message}}
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
    <script src="/js/injector_custom.js"></script>
  </body>
</html>
```

Applying Configuration and Run Blocks to Modules

Now that you understand the relationship between modules and dependency injection, you need to look at the process of implementing AngularJS modules. AngularJS modules are implemented in two phases: the configuration phase and the run phase. The following sections discuss these phases and the basic process of adding providers to an AngularJS module.

Adding Configuration Blocks

The AngularJS module configuration phase is executed when a module is being defined. During this phase, any providers defined are registered with the dependency injector. You should put only configuration and provider code inside the configuration block.

You implement the configuration block by calling the `config()` method on the instance of the `Module` object, using the following syntax:

Click here to view code image

```
config(function([injectable, . . .]))
```

A function with the injectable parameters is passed in. The injectable parameters are typically provider services functions, such as `$provide`.

The following is an example of a basic configuration block:

Click here to view code image

```
var myModule = angular.module('myModule', []).
  config(function($provide, $filterProvider) {
    $provide.value("startTime", new Date());
    $filterProvider.register('myFilter', function(){});
});
```

Notice that the `$provide` and `$filterProvider` services are passed into the `config` function. They are used to register a value provider named `startTime` and a filter provider named `myFilter` with the injector service.

Adding Run Blocks

After an entire configuration block has finished, the run phase of an AngularJS module can execute. During this phase, you can implement any code necessary to instantiate the module. You cannot implement any provider code during the run block because the entire module should already be configured and registered with the dependency injector by this point.

The run block is a great place to put event handlers that need to be executed at the root level for the application (for example, authentication handlers).

You implement the `run` block by calling the `run()` method of the `Module` object, using the following syntax:

Click here to view code image

```
run(function([injectable, . . .]))
```

A function with the instance `injectable` parameters is passed in. The `injectable` parameters should only be instances of injectors because configuration should already have been completed.

The following is a basic implementation of the `run` block continued from the example:

Click here to view code image

```
myModule.run(function(startTime) {
  startTime.setTime((new Date()).getTime());
});
```

Notice that the `startTime` instance defined in the `config()` section shown previously is passed into the `run()` function. This allows the `run()` function to update the `startTime` provider to a new value.

Try it Yourself: Implementing Configuration and Run Blocks

In this example, you learn how to implement very basic configuration and run blocks. The code in Listing 21.5 uses the `config()` method to implement two providers, `configTime` and `runTime`, that are JavaScript `Date` objects that will be used to display time values at configuration and then at run.

Use the following steps to build the application and implement the configuration and run blocks:

1. Add the lesson21/config_run_blocks.html and lesson21/js/config_run_blocks.js files.

2. Add the code shown in Listing 21.5 and Listing 21.6 to the HTML and JavaScript files.

3. The following code from Listing 21.5 implements the configuration block that defines two `value` providers named `configTime` and `runTime`. Notice that in lines 5–7, a simple loop is implemented to inject a delay simulating a delay that might be caused during configuration:

Click here to view code image

```
02 myModule.config(function($provide) {
03     $provide.value("configTime", new Date());
04     $provide.value("runTime", new Date());
05     for(var x=0; x<1000000000; x++){
06       var y = Math.sqrt(Math.log(x));
07     }
08 });
```

4. The following code then defines the run block that changes the value of `runTime`:

Click here to view code image

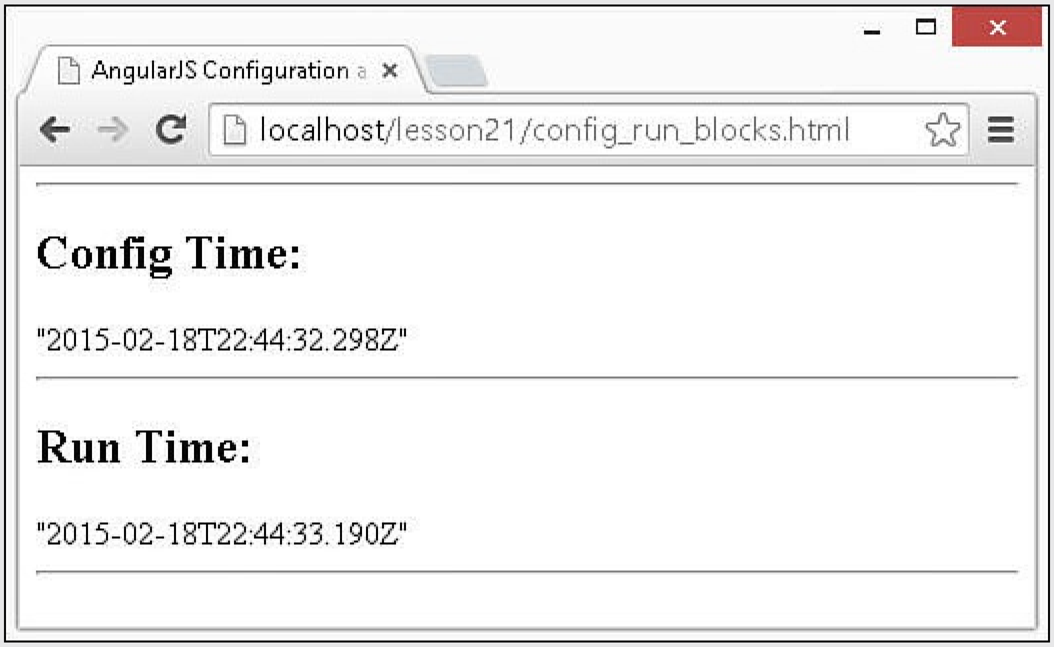
```
09 myModule.run(function(configTime, runTime) {
10   runTime.setTime((new Date()).getTime());
11 });
```

5. In the following lines, the controller is defined and injected with `configTime` and `runTime`, which are added to the scope values:

Click here to view code image

```
myModule.controller('controllerA',['$scope', 'configTime',
'runTime',
    function($scope, configTime, runTime){
  $scope.configTime = configTime;
  $scope.runTime = runTime;
}]);
```

6. The template from Listing 21.6 displays the `configTime` and `runTime` values, as shown in Figure 21.3.


AngularJS Configuration a
localhost/lesson21/config_run_blocks.html

Config Time:

"2015-02-18T22:44:32.298Z"

Run Time:

"2015-02-18T22:44:33.190Z"

FIGURE 21.3 Implementing configuration and run blocks that set and utilize JavaScript Date objects to display the time executed for each.

LISTING 21.5 config_run_blocks.js Implementing Configuration and Run Blocks in an AngularJS Module

Click here to view code image

```
var myModule = angular.module('myApp', []);
myModule.config(function($provide) {
    $provide.value("configTime", new Date());
    $provide.value("runTime", new Date());
    for(var x=0; x<1000000000; x++){
      var y = Math.sqrt(Math.log(x));
    }
});
myModule.run(function(configTime, runTime) {
  runTime.setTime((new Date()).getTime());
});
```

```
myModule.controller('controllerA',['$scope', 'configTime', 'runTime',
    function($scope, configTime, runTime){
  $scope.configTime = configTime;
  $scope.runTime = runTime;
}]);
```

LISTING 21.6 run_blocks.html Using HTML Code to Display the configTime and runTime Values Generated in the Configuration and Run Blocks of the AngularJS Module

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Configuration and Run Blocks</title>
  </head>
  <body>
    <div ng-controller="controllerA">
      <hr>
      <h2>Config Time:</h2>
      {{configTime}}
      <hr>
      <h2>Run Time:</h2>
      {{runTime}}
    </div><hr>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
    <script src="js/config_run_blocks.js"></script>
  </body>
</html>
```

Summary

Dependency injection enables you to define provider functionality that can be injected into other AngularJS components. The provider functionality is contained inside modules and registered with an injector service. Providers define how to build the functionality so that when another component defines a dependency on a provider, an instance of the provider object can be created and injected.

AngularJS provides a fairly robust dependency injection model that enables you to define different types of service providers. Using dependency injection rather than global definitions makes your code more modularized and easier to maintain. In this lesson, you were introduced to the dependency injection model and saw how to implement it in both modules and a controller component.

Q&A

Q. Will the controller and directive code for an AngularJS application be run before or after the run block?

A. The order of execution is config, run, directive.compile, controller, and directive link.

Workshop

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

Quiz

1. Which block is executed first—the run block or the configuration block?
2. Would you place provider code in the run block or configuration block?
3. How would you inject the `$scope` and `$http` services into a controller?
4. True or false: You can inject one controller into another.

Quiz Answers

1. The configuration block is run first.
2. You should place provider code in the configuration block because the instance of injectable objects should have been created before executing the run block.
3. Use something similar to the following code:

Click here to view code image

```
 app.controller('myController', ['$scope', '$http'], function($scope,
$http) { ...
```

4. True

Exercises

1. Modify the code in Listing 21.1 to show the screenX and screenY values in the alert pop-up. You can do that by adding the following code to the message:

Click here to view code image

```
+ ' ' + $window.screenX + 'x'+ $window.screenY
```

2. Modify the code in Listings 21.3 and 21.4 to add a third controller named controllerC that displays a third message.

Lesson 22. Implementing the Scope as a Data Model

What You'll Learn in This Lesson:

- How the scope relates to the data model
- How scopes relate to the AngularJS templates
- How the scope hierarchy in nested controllers works
- Understanding the scope life cycle

One of the most important aspects of an AngularJS application is scope. Scope not only provides the data represented in a model but also binds together all the other components of the AngularJS application, such as modules, controllers, services, and templates. This lesson explains the relationships between scope and other AngularJS components.

Scope provides the binding mechanism that enables DOM elements and other code to be updated when changes occur in the model data. In this lesson, you learn about the root scope and child scopes. You also learn about the scope hierarchies and how to implement them.

Understanding Scopes

In AngularJS, the scope acts as a data model for an application. It is one of the most critical parts of any application that relies on data in any fashion because it acts as the glue that binds together the views, business logic, and server-side data. Understanding how scopes work enables you to design your AngularJS applications to be more efficient, use less code, and be easier to follow.

The following sections discuss the relationships between scope and applications, controllers, templates, and server-side data. There is also a section that covers the life cycle of scope to help you see how scope is built, manipulated, and updated during the application life cycle.

The Relationship Between the Root Scope and Applications

When an application is bootstrapped, a root scope is created. The root scope stores data at the application level, and you can access it by using the `$rootScope` service. The root scope data should be initialized in the `run()` block of the module, but you can also access it in components of the module. To illustrate this point, the following code defines a value at the root scope level and then accesses it in a controller:

Click here to view code image

```
angular.module('myApp', [])
.run(function($rootScope) {
    $rootScope.rootValue = 5;
})
.controller('myController', function($scope, $rootScope) {
  $scope.value = 10;
  $scope.difference = function() {
        return $rootScope.rootValue - $scope.value;
    };
});
```

The Relationship Between Scopes and Controllers

Controllers are pieces of code that are intended to provide business logic by augmenting scope. You create controllers by using the `controller()` method on the `Model` object of an application. This function registers a controller as a provider in the module, but it does not create an instance of the controller. That occurs when the `ng-controller` directive is linked in an AngularJS template.

The `controller()` method accepts the controller name as the first parameter and an array of dependencies as the second parameter. For example, the following code defines a controller that uses dependency injection to access a `value` provider named `start`:

Click here to view code image

```
    angular.module('myApp', []).
      value('start', 200).
      controller('Counter', ['$scope', 'start',
                             function($scope, startingValue) {
      }]);
```

Notice that the `value()` and `controller()` methods are chained together using the dot '.' notation. You will see this used often in AngularJS applications. When a new instance of a controller is created in AngularJS, a new child scope specific to that controller is also created and is accessible via the `$scope` service that is injected into the preceding `Counter` controller. Also in the example shown previously, the `start` provider is injected into the controller and passed to the controller function as `startingValue`. The parameter injection is based on their position in the array passed to the `controller()` function.

The controller must initialize the state of a scope that is created and added to it. The controller is also responsible for any business logic attached to that scope. This can mean handling update changes to the scope, manipulating scope values, or emitting events based on the state of the scope.

Try it Yourself: Implementing a Basic Controller

In this example, you get a chance to play around a bit with controller code. Listing 22.1 shows how to implement a controller that utilizes dependency injection, initializes some values, and implements rudimentary business logic, using the `inc()`, `dec()`, and `calcDiff()` functions. Listing 22.2 shows a basic AngularJS HTML template that provides the view to see and manipulate the values stored in the scope.

Use the following steps to build the application and implement the application and controller:

1. Add the lesson22/scope_controller.html and lesson22/js/scope_controller.js files.

2. Add the code shown in Listing 22.1 and Listing 22.2 to the HTML and JavaScript files.

3. The following code from Listing 22.1 implements an AngularJS module and with a controller named Counter:

Click here to view code image

```
angular.module('myApp', []).
  value('start', 200).
  controller('Counter', ['$scope', 'start',
                          function($scope, start) {
```

4. Then the following lines define values in the `$scope` of the Counter controller:

Click here to view code image

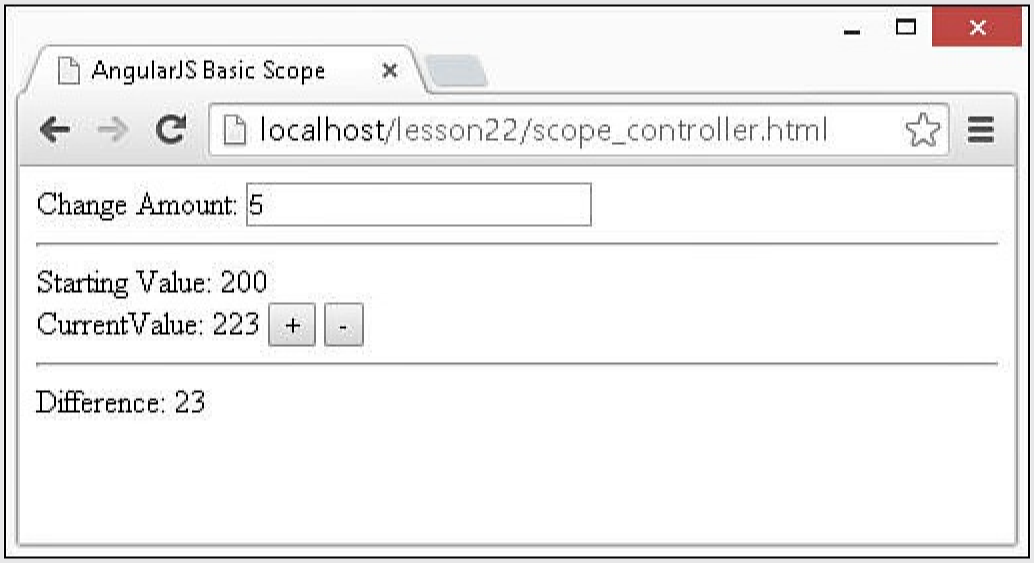
```
    $scope.start = start;
    $scope.current = start;
    $scope.difference = 0;
    $scope.change = 1;
```

5. Notice that in the following lines, the functions `inc()`, `dec()`, and `calcDiff()` are defined that manipulate the values for `current` and `difference`:

Click here to view code image

```
    $scope.inc = function() {
      $scope.current += $scope.change;
      $scope.calcDiff();
    };
    $scope.dec = function() {
      $scope.current -= $scope.change;
      $scope.calcDiff();
    };
    $scope.calcDiff = function() {
      $scope.difference = $scope.current - $scope.start;
    };
```

6. The code Listing 22.2 implements an AngularJS template that provides a number input to set the value of `change` in the scope, as well as + and – buttons that call `inc()` and `dec()` to change the value of current. The template also displays the starting, current, and difference values using simple AngularJS filters. Figure 22.1 shows the web page in action. You can set the increment/decrement value and then click +/– buttons to decrement the current value and see the difference change in the scope.



Change Amount: 5

Starting Value: 200
CurrentValue: 223

Difference: 23

FIGURE 22.1 A basic AngularJS application that stores initial and current values in the scope and then displays the difference to illustrate scope data interaction.

LISTING 22.1 scope_controller.js Implementing a Basic Controller That Uses Dependency Injection, Initializes Scope Values, and Implements Business Logic

Click here to view code image

```
angular.module('myApp', []).
  value('start', 200).
  controller('Counter', ['$scope', 'start',
                         function($scope, start) {
    $scope.start = start;
    $scope.current = start;
    $scope.difference = 0;
    $scope.change = 1;
    $scope.inc = function() {
      $scope.current += $scope.change;
      $scope.calcDiff();
    };
    $scope.dec = function() {
      $scope.current -= $scope.change;
      $scope.calcDiff();
    };
    $scope.calcDiff = function() {
      $scope.difference = $scope.current - $scope.start;
    };
}]);
```

LISTING 22.2 scope_controller.html HTML Template That Enables You to See the Data in the Scope Change Dynamically Based on Incrementing and Decrementing Values

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Basic Scope</title>
  </head>
  <body>
    <div ng-controller="Counter">
      <span>Change Amount:</span>
      <input type="number" ng-model="change"><hr>
      <span>Starting Value:</span>
      {{start}}
      <br>
      <span>CurrentValue:</span>
      {{current}}
      <button ng-click='inc()'>+</button>
      <button ng-click='dec()'>-</button><hr>
      <span>Difference:</span>
      {{difference}}
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
    <script src="js/scope_controller.js"></script>
  </body>
</html>
```

The Relationship Between Scopes and Templates

Templates provide the view for an AngularJS application. HTML elements are defined as controllers, using the `ng-controller` attribute. Inside a controller HTML element and its children, the scope for that controller is available for expressions and other AngularJS functionality.

Values in a scope can be directly linked to the values of `<input>`, `<select>`, and `<textarea>` elements in the template, using the `ng-model` directive. This directive links the value of an element to a property name in the scope. When the user changes the value of the input element, the scope is automatically updated. For example, the following links the value of a number of an `<input>` element to the scope named `valueA`:

Click here to view code image

```
<input type="number" ng-model="valueA" />
```

You can add scope properties and even functions to expressions in a template by using the `{{expression}}` syntax. The code inside the brackets is evaluated, and the results are displayed in the rendered view. For example, if a scope contains properties named `valueA` and `valueB`, you can reference these properties in an expression in the template, as shown here:

```
{{valueA + valueB}}
```

You can also use scope properties and functions when defining AngularJS directives in a template. For example, the `ng-click` directive binds the browser click event to a function in a scope named `addValues()` and passes the values of properties `valueA` and `valueB` in the scope:

Click here to view code image

```
<span ng-click="addValues(valueA, valueB")>Add Values{{valueA}} &
{{valueB}}</span>
```

Notice that in this code, the `{{}}` brackets are required. However, in the `addValues()` function call, they are not required. That is because `ng-click` and other AngularJS directives automatically evaluate as expressions.

Try it Yourself: Looking at Scope Values in the AngularJS Template

In this example, you put all these concepts together in a very basic example to make it easy to understand the relationship between the model and the scope. Listing 22.3 implements a controller named `SimpleTemplate` that initializes a scope with three values: `valueA`, `valueB`, and `valueC`. The scope also contains a function named `addValues()` that accepts two parameters and adds them together to set the value of `$scope.valueC`.

Listing 22.4 is an AngularJS template that initializes the `SimpleTemplate` controller defined in Listing 22.3 and provides number inputs to set the values of valueA and valueB and a button to calculate the value of valueC.

Use the following steps to build the application and implement the interaction between the scope and template:

1. Add the lesson22/scope_template.html and lesson22/js/scope_template.js files.
2. Add the code shown in Listing 22.3 and Listing 22.4 to the HTML and JavaScript files.
3. The following code from Listing 22.3 to define the values of `valueA`,

valueB, and valueC in the scope:

```
$scope.valueA = 5;
$scope.valueB = 7;
$scope.valueC = 12;
```

4. The following code provides a function in the scope that accepts two parameters and adds them together to set the value of valueC:

Click here to view code image

```
$scope.addValues = function(v1, v2) {
  $scope.valueC = v1 + v2;
};
```

5. The following lines in Listing 22.4 implement the number inputs and bind them to the values of valueA and valueB:

Click here to view code image

```
ValueA: <input type="number" ng-model="valueA" /><br>
ValueB: <input type="number" ng-model="valueB" /><br><br>
```

6. The following line shows the value of valueA and valueB in the template using the simple AngularJS expressions:

Click here to view code image

```
Expression: {{valueA}} + {{valueB}}<br><br>
```

7. The following line then uses an angularJS expression to display the value of valueA added to valueB:

Click here to view code image

```
Live Expression Value: {{valueA + valueB}}<br><br>
```

8. The following lines of code from Listing 22.4 implement an input button that calls addValues() when you click it and then passes in the value of valueA and valueB:

Click here to view code image

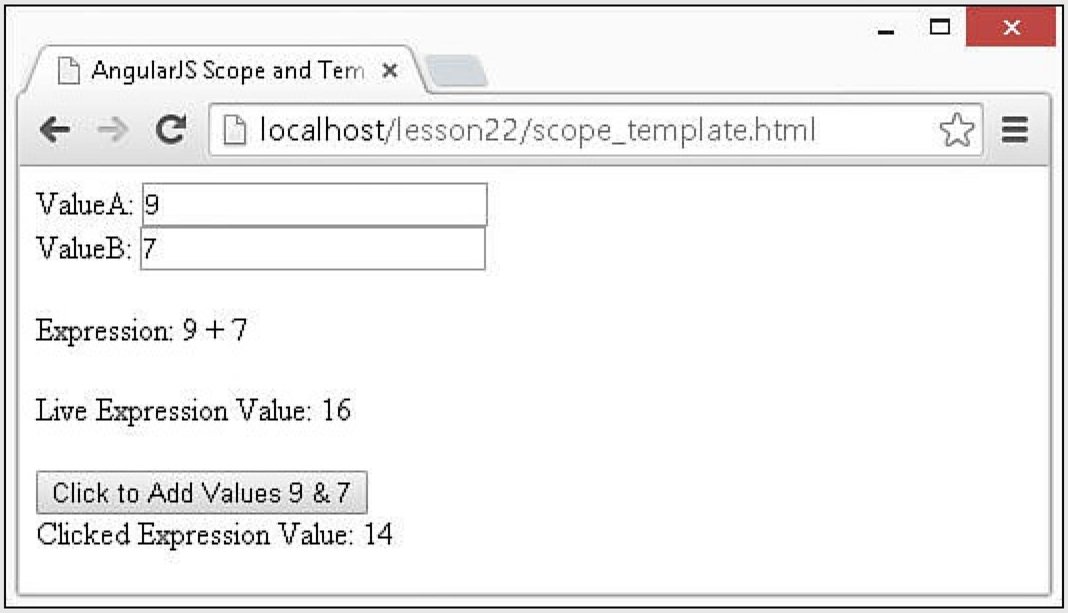
```
<input type="button" ng-click="addValues(valueA, valueB)"
  value ="Click to Add Values {{valueA}} & {{valueB}}" /><br>
```

9. The following line of code displays the value of valueC in the scope:

Click here to view code image

```
Clicked Expression Value: {{valueC}}<br>
```

10. Figure 22.2 shows this simple application in a web browser. As the two input elements are changed, the expressions change automatically. However, the `valueC` expression changes only when the `Click to Add Values` button is clicked.


AngularJS Scope and Tem
localhost/lesson22/scope_template.html

ValueA: 9
ValueB: 7

Expression: 9 + 7

Live Expression Value: 16

Click to Add Values 9 & 7
Clicked Expression Value: 14

FIGURE 22.2 A basic AngularJS template that implements a controller and links several fields to the scope to provide both input of values and displayed results.

LISTING 22.3 scope_template.js Implementing a Basic Controller to Support Template Functionality

Click here to view code image

```
angular.module('myApp', []).
  controller('SimpleTemplate', function($scope) {
    $scope.valueA = 5;
    $scope.valueB = 7;
    $scope.valueC = 12;
    $scope.addValues = function(v1, v2) {
      $scope.valueC = v1 + v2;
    };
  });
```

LISTING 22.4 scope_template.html HTML Template Code That Implements a Controller and Various HTML Fields Linked to the Scope

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Scope and Templates</title>
  </head>
  <body>
    <div ng-controller="SimpleTemplate">
      ValueA: <input type="number" ng-model="valueA" /><br>
      ValueB: <input type="number" ng-model="valueB" /><br><br>
      Expression: {{valueA}} + {{valueB}}<br><br>
      Live Expression Value: {{valueA + valueB}}<br><br>
      <input type="button" ng-click="addValues(valueA, valueB)"
        value ="Click to Add Values {{valueA}} & {{valueB}}" /><br>
      Clicked Expression Value: {{valueC}}<br>
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
    <script src="js/scope_template.js"></script>
  </body>
</html>
```

The Relationship Between Scope and Back-End Server Data

Data that is used for an AngularJS application often comes from a back-end data source, such as a database. In such instances, the scope still acts as the definitive source of data for the AngularJS application. You should use the following rules when interacting with data that is coming from the server side:

- Access data from the database or other back-end sources via AngularJS services, which are discussed in Lesson 27, "Implementing AngularJS Services in Web Applications." This includes both reading and updating data.
- Ensure that data read from the server updates the scope, which in turn updates the view. Avoid the temptation to manipulate the HTML values directly from the database, which can lead to the scope becoming out of sync with the view.
- Reflect changes that are made to the database or other back-end source in scope as well. You can do this by first updating the scope and then updating the database using a service, or you can update the database and then use the results from the database to repopulate the appropriate values in the scope.

The Scope Life Cycle

Scope data goes through a life cycle while the application is loaded in the browser. Understanding this life cycle will help you understand the interaction between scope and other AngularJS components, especially templates.

Scope data goes through the following life cycle phases:

1. Creation
2. Watcher registration
3. Model mutation

4. Mutation observation

5. Scope destruction

These life cycle phases are described in the sections that follow.

The Creation Phase

The creation phase occurs when a scope is initialized. Bootstrapping the application creates a root scope. Linking the template creates child scopes when `ng-controller` or `ng-repeat` directives are encountered.

During the creation phase, a digest loop also is created that interacts with the browser event loop. The digest loop is responsible for updating DOM elements with changes made to the model, as well as executing any registered watcher functions. Although you should never need to execute a digest loop manually, you can do so by executing the `$digest()` method on the scope. For example, the following evaluates any asynchronous changes and then executes the watch functions on the scope:

```
$scope.$digest()
```

The Watcher Registration Phase

The watcher registration phase registers watches for values in the scope that are represented in the template. These watches propagate model changes automatically to the DOM elements.

You can also register your own watch functions on a scope value by using the `$watch()` method. This method accepts a scope property name as the first parameter and then a callback function as the second parameter. The old and new values are passed to the callback function when the property is changed in the scope.

For example, the following adds a watch to the property `watchedItem` in the scope and increments a counter each time it is changed:

Click here to view code image

```
$scope.watchedItem = 'myItem';
$scope.counter = 0;
$scope.$watch('watchedItem', function(newValue, oldValue) {
  $scope.watchedItem = $scope.counter + 1;
});
```

The Model Mutation Phase

The model mutation phase occurs when data in the scope changes. When you make changes in your AngularJS code, a scope function called `$apply()` updates the model and calls the `$digest()` function to update the DOM and watches. This is how changes made in your AngularJS controllers or by the `$http`, `$timeout`, and `$interval` services are automatically updated in the DOM.

You should always try to make changes to scope inside the AngularJS controller or those services. However, if you must make changes to the scope outside the AngularJS realm, you need to call `scope.$apply()` on the scope to force the model and DOM to be updated correctly. The `$apply()` method accepts an expression as the only parameter. The expression is evaluated and returned, and the `$digest()` method is called to update the DOM and watches.

The Mutation Observation Phase

The mutation observation phase occurs when the `$digest()` method is executed by the digest loop, an `$apply()` call, or manually. When `$digest()` executes, it evaluates all watches for changes. If a value has changed, `$digest()` calls the `$watch` listener and updates the DOM.

The Scope Destruction Phase

The `$destroy()` method removes scopes from the browser memory. The AngularJS libraries automatically call this method when child scopes are no longer needed. The `$destroy()` method stops `$digest()` calls and removes watches, allowing the memory to be reclaimed by the browser garbage collector.

Implementing Scope Hierarchy

A great feature of scopes is that they are organized in a hierarchy. The hierarchy helps you keep scopes organized and relevant to the context of the view they represent. There is a root scope at the AngularJS module level and then child scopes can be implemented in subcomponents, such as controllers or directives. Child scopes can be nested within each other creating a hierarchy structure.

By the Way

The `$digest()` method uses the scope hierarchy to propagate scope changes to the appropriate watchers and the DOM elements.

Scope hierarchies are created automatically based on the location of `ng-controller` statements in the AngularJS template. For example, the following template code defines two `<div>` elements that create instances of controllers that are siblings:

Click here to view code image

```
<div ng-controller="controllerA"> . . . </div>
<div ng-controller="controllerB"> . . . </div>
```

However, the following template code defines controllers in which `controllerA` is the parent of `controllerB`:

Click here to view code image

```
<div ng-controller="controllerA">
  <div ng-controller="controllerB"> . . . </div>
</div>
```

The scope hierarchy works similar to the way object inheritance works in OO languages. You can access the values of parent scopes from a controller, but you can't access the values of sibling or children scopes. If you add a property name in a child scope, it does not overwrite the parent but creates a property of the same name in the child scope that has a different value from the parent.

Try it Yourself: Implementing Nested Controllers and Scopes

In this example, you look at the behavior of scopes within nested controllers. Listings 22.5 and 22.6 implement a basic nested scope hierarchy to demonstrate how scopes work in a hierarchy. Listing 22.5 creates an application with three controllers, each with two scope items defined. They all share the common scope property `title` and the scope properties `valueA`, `valueB`, and `valueC`. Listing 22.6 creates the three controllers in an AngularJS template.

Use the following steps to build the application with nested controllers and scopes:

1. Add the lesson22/scope_hierarchy.html and lesson22/js/scope_ hierarchy.js files.
2. Add the code shown in Listing 22.5 and Listing 22.6 to the HTML and JavaScript files.
3. The code in Listing 22.5 implements three controllers: `LevelA`, `LevelB`, and `LevelC`. Three controllers are defined, each with a `title`, `value`, and `inc()` function to increment the `value`. The following code shows the definition for `LevelA`:

Click here to view code image

```
controller('LevelA', function($scope) {
  $scope.title = "Level A"
  $scope.valueA = 1;
  $scope.inc = function() {
    $scope.valueA++;
  };
}).
```

4. In Listing 22.6, the following three `<div>` elements are nested, which also nests the controllers assigned to each:

Click here to view code image

```
<div ng-controller="LevelA">
. . .
  <div ng-controller="LevelB"><hr>
. . .
    <div ng-controller="LevelC"><hr>
. . .
    </div>
  </div>
</div>
```

5. The following template code in the `LevelA <div>` displays the `title` and `valueA` values from `LevelA` and calls the `inc()` function for `LevelA` when the button is clicked:

Click here to view code image

```
  <h3>{{title}}</h3>
  ValueA = {{valueA}} <input type="button" ng-click="inc()"
value="+" />
```

6. The following template code in the `LevelB <div>` displays the `title` and `valueB` values from `LevelB`, but also the `valueA` value from the `LevelA` scope. Also, when the button defined here is clicked, it is the `inc()` function for `LevelB` that is called:

Click here to view code image

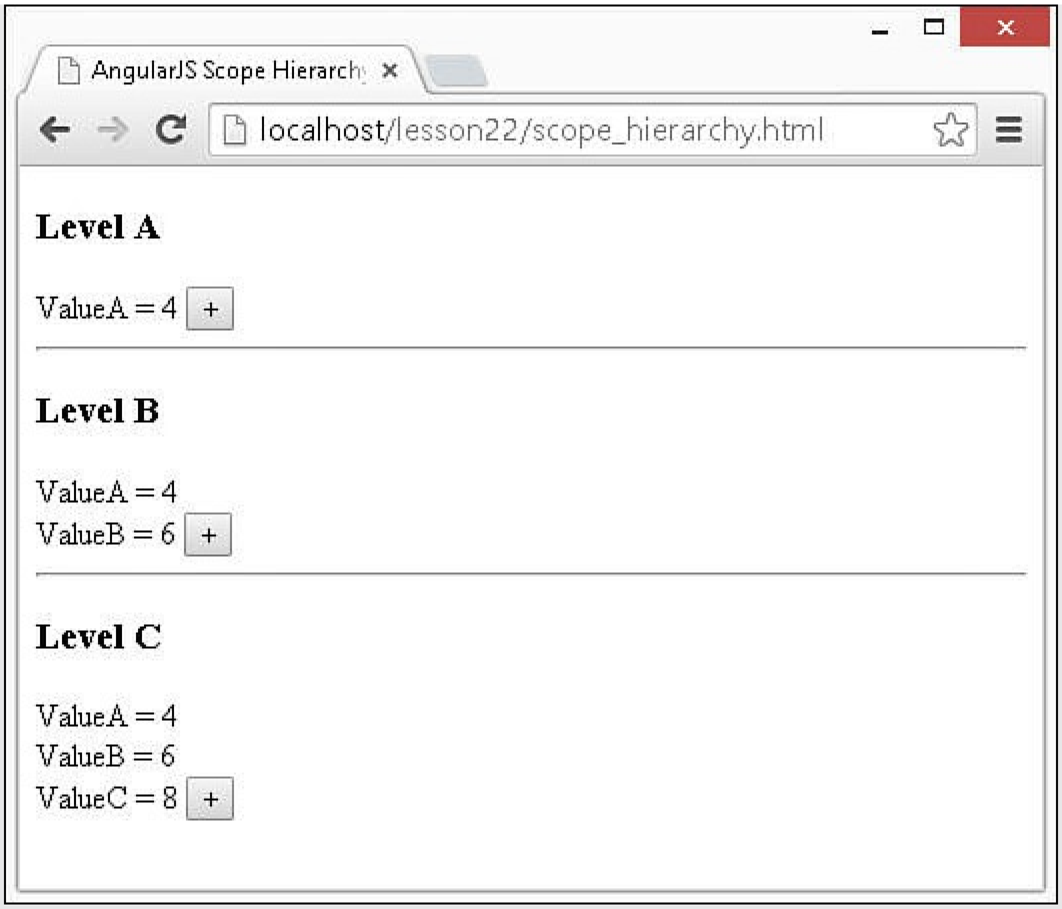
```
    <h3>{{title}}</h3>
    ValueA = {{valueA}}<br>
    ValueB = {{valueB}}
    <input type="button" ng-click="inc()" value="+" />
```

7. The following template code in the `LevelC <div>` displays the `title` and `valueC` values from `LevelB`, but also the `valueA` value from the `LevelA` scope and the `valueB` value from the `LevelB` scope. Also, when the button defined here is clicked, the `inc()` function for `LevelC` is called:

Click here to view code image

```
     <h3>{{title}}</h3>
      ValueA = {{valueA}}<br>
      ValueB = {{valueB}}<br>
      ValueC = {{valueC}}
      <input type="button" ng-click="inc()" value="+" />
```

8. Figure 22.3 shows the rendered AngularJS application. Notice that the value of the `title` property in all three scopes is different. That is because a new `title` property is created for each level in the hierarchy.



Level A

ValueA = 4 +

Level B

ValueA = 4
ValueB = 6 +

Level C

ValueA = 4
ValueB = 6
ValueC = 8 +

FIGURE 22.3 Implementing a hierarchy of controllers that render results from the multiple levels of scope.

LISTING 22.5 scope_hierarchy.js Implementing a Basic Scope Hierarchy with Access to Properties at Each Level

[Click here to view code image]

```
angular.module('myApp', []).
  controller('LevelA', function($scope) {
    $scope.title = "Level A"
    $scope.valueA = 1;
    $scope.inc = function() {
      $scope.valueA++;
    };
  }).
  controller('LevelB', function($scope) {
    $scope.title = "Level B"
    $scope.valueB = 1;
    $scope.inc = function() {
      $scope.valueB++;
    };
  }).
  controller('LevelC', function($scope) {
    $scope.title = "Level C"
    $scope.valueC = 1;
    $scope.inc = function() {
      $scope.valueC++;
    };
  });
```

LISTING 22.6 scope_hierarchy.html HTML Template Code That Implements a Hierarchy of Controllers and Renders Results from the Multiple Levels of Scope

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
<title>AngularJS Scope Hierarchy</title>
</head>
<body>
  <div ng-controller="LevelA">
    <h3>{{title}}</h3>
    ValueA = {{valueA}} <input type="button" ng-click="inc()" value="+" />
    <div ng-controller="LevelB"><hr>
      <h3>{{title}}</h3>
      ValueA = {{valueA}}<br>
      ValueB = {{valueB}}
      <input type="button" ng-click="inc()" value="+" />
      <div ng-controller="LevelC"><hr>
       <h3>{{title}}</h3>
        ValueA = {{valueA}}<br>
        ValueB = {{valueB}}<br>
        ValueC = {{valueC}}
        <input type="button" ng-click="inc()" value="+" />
      </div>
    </div>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/scope_hierarchy.js"></script>
</body>
</html>
```

Summary

A scope is the definitive source for data in AngularJS applications. A scope has direct relationships with the template views, controllers, modules, and services and acts as the glue that binds the application together. A scope also acts as a representation of a database or another server-side data source.

The scope life cycle is linked to the browser event loop so that changes in the browser can change the scope, and changes in the scope are reflected in the DOM elements that are bound to the scope. You can also add custom watch functions that are notified when the scope changes.

Scopes are organized into hierarchies, and the root scope is defined at that application level. Each instance of a controller also gets an instance of a child scope. From the child scopes, you can access data that is stored in the parent scope hierarchy.

Q&A

Q. Does the scope have to match the data model exactly—for instance, data coming from the server?

A. No, the scope for the AngularJS application should match what is necessary for the application itself. However, keeping it matched as close to the back-end data as possible can make it easier to maintain the code.

Q. Is it possible to modify the scope from outside the AngularJS application?

A. Yes, you can always access the scope using `anguler.element(<dom_element>).scope()`. However, it is usually best to modify the scope only from inside the application itself.

Workshop

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

Quiz

1. What are the phases of the scope life cycle?
2. Where should you typically make changes to the scope?
3. Will changing the value in the scope automatically update references in expressions within the view?
4. In the following example, is controller B a parent, sibling, or child of controllers A and/or C?

Click here to view code image

```
<div ng-controller="controllerA">
  <div ng-controller="controllerB">
    <div ng-controller="controllerC">. . . </div>
  </div>
</div>
```

Quiz Answers

1. Creation, watch registration, model mutation, mutation observation, and scope destruction.
2. Typically, scope value changes are made either in the controller or directive code.
3. Yes. The change is detected, which updates the DOM and consequently the browser view.
4. B is a parent of C and a child of A.

Exercises

1. Modify the example in Listing 22.1 and Listing 22.2 to add a button that resets the value of `current` in the scope to the `start` value when it is clicked.
2. Modify the example in Listings 22.3 and 22.4 to add another function named `multiplyValues()` to the scope that multiplies values instead of adding them. You will also need to add the appropriate expression and button to the template to expose that functionality.

Lesson 23. Using AngularJS Templates to Create Views

What You'll Learn in This Lesson:

- How and when to use AngularJS expressions
- How to create AngularJS templates
- Ways to use the built-in AngularJS filters
- How to create your own custom AngularJS filters

AngularJS templates provide a framework to represent the application view to the user. AngularJS templates contain expressions, filters, and directives that define additional functionality and behavior to the DOM elements. The templates are built on top of normal HTML and extend the functionality of HTML by adding additional elements and attributes.

This lesson focuses on AngularJS templates, as well as expressions and filters. Expressions enable you to implement JavaScript-like code alongside the HTML code in a template. Filters enable you to modify data values before you display them—for example, to format text.

Understanding Templates

AngularJS templates are fairly straightforward yet very powerful and easy to extend. Templates are based on standard HTML documents but extend the HTML functionality with three additional components:

- **Expressions:** Expressions are bits of JavaScript-like code that are evaluated within the context of a scope. Expressions are denoted by `{{}}` brackets. The results of an expression are added to a compiled HTML web page. Expressions can be placed in normal HTML text or in the values of attributes, as shown here:

Click here to view code image

```
<p>{{1+2}}</p>
href="/myPage.html/{{hash}}"
```

- **Filters:** Filters transform the appearance of data that is placed on a web page. For example, a filter can convert a number from the scope into a currency string or a time string.
- **Directives:** Directives are new HTML element names or attribute names within HTML elements. They add to and modify the behavior of HTML elements to provide data binding, event handling, and other support to an AngularJS application.

The following code snippet shows an example of implementing directives, expressions, and filters. The `ng-model="msg"` attribute is a directive that binds the value of the `<input>` element to `msg` in the scope. The code in the `{{}}` brackets is an expression that applies the `uppercase` filter:

```
<div>
  <input ng-model="msg">
  {{msg | uppercase}}
</div>
```

When you load an AngularJS web page into a browser, you load it in a raw state, containing template code along with HTML code. The initial DOM is built from that web page. When the AngularJS application is bootstrapped, the AngularJS template compiles into the DOM, dynamically adjusting the values, event bindings, and other properties of the DOM elements to the directives, expressions, and filters in the template.

During compilation, HTML tags and attributes are normalized to support the fact that AngularJS is case sensitive, whereas HTML is not. Normalization does two things:

- Strips the `x-` and `data-` prefixes from the front of elements and attributes.
- Converts names with `:` or `-` or `_` to camelCase.

For example, all of the following normalize to `ngModel`:

```
ng-model
data-ng-model
x-ng:model
ng_model
```

Using Expressions

Using expressions is the simplest way to represent data from the scope in an AngularJS view. Expressions are encapsulated blocks of code inside brackets: `{{expression}}`. The AngularJS compiler compiles an expression into HTML elements so that the results of the expression are displayed. For example, look at the following expressions:

```
{{1+5}}
{{'One' + 'Two'}}
```

Based on those expressions, the web page displays these values:

```
6
OneTwo
```

Expressions are bound to the data model, which provides two huge benefits. First, you can use the property names and functions that are defined in the scope inside your expressions. Second, because the expressions are bound to the scope, when data in the scope changes, so do the expressions. For example, suppose that the scope contains the following values:

```
$scope.name='Brad';
$scope.score=95;
```

You can directly reference the name and score values in the template expressions, as shown here:

```
Name: {{name}}
Score: {{score}}
Adjusted: {{score+5}}
```

AngularJS expressions are similar to JavaScript expressions in several ways, but they differ in these ways:

- **Attribute evaluation:** Property names are evaluated against the scope model instead of against the global JavaScript namespace.
- **More forgiving:** Expressions do not throw exceptions when they encounter undefined or null variable types; instead, they treat these as having no value.
- **No flow control:** Expressions do not allow JavaScript conditionals or loops. Also, you cannot throw an error inside an expression.

AngularJS evaluates the strings used to define the value of attributes in directive tags as expressions. This enables you to include expression-type syntax within a definition. For example, when you set the value of the `ng-click` directive in the template, you specify an expression. Inside that expression, you can reference scope variable and use other expression syntax, as shown here:

Click here to view code image

```
<span ng-click="scopeFunction()"></span>
<span ng-click="scopeFunction(scopeVariable, 'stringParameter')"></span>
<span ng-click="scopeFunction(5*scopeVariable)"></span>
```

Because the AngularJS template expressions have access to the scope, you can also make changes to the scope inside the AngularJS expression. For example, the following `ng-click` directive changes the value of `msg` inside the scope model:

Click here to view code image

```
<span ng-click="msg='clicked'"></span>
```

The following sections take you through some examples of utilizing the expression capability in AngularJS.

Try it Yourself: Using Basic Expressions

In this exercise, you see how AngularJS expressions handle rendering of strings and numbers. The purpose of this exercise is to illustrate how AngularJS evaluates expressions that contain strings and numbers as well as basic mathematical operators.

The code in Listing 23.1 is a simple AngularJS application with a controller named `myController`. The controller is empty because none of the expressions access the scope.

The code in Listing 23.2 is an AngularJS template that contains several types of expressions wrapped in `{{}}` brackets. Some of the expressions are numbers or strings, some include the + operation to combine strings and/or numbers, and one applies a === operator to compare two numbers.

Use the following steps to build the application that uses basic expressions:

1. Add the lesson23/expressions_basic.html and lesson23/js/expressions_basic.js files.

2. Add the code shown in Listing 23.1 and Listing 23.2 to the HTML and JavaScript files.

3. The following code in Listing 23.1 implements a basic AngularJS controller:

Click here to view code image

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
  });
```

4. The following line of code from Listing 23.2 uses the expression brackets `{{}}` to display a number:

```
      {{5}}<hr>
```

5. The following line of code from Listing 23.2 uses the expression brackets `{{}}` to display a string:

Click here to view code image

```
      {{'My String'}}<hr>
```

6. The following line of code from Listing 23.2 uses the expression brackets `{{}}` to display a string by adding strings together:

Click here to view code image

```
      {{'String1' + ' ' + 'String2'}}<hr>
```

7. The following line of code from Listing 23.2 uses the expression brackets `{{}}` to display a number by adding numbers together:

```
      {{5+5}}<hr>
```

8. The following line of code from Listing 23.2 uses the expression brackets

`{{}}` to display a string and number by adding them together. Adding `{{5+5}}` will display `10`, but adding `{{ 5 + '+ ' + 5}}` will display `5 + 5`:

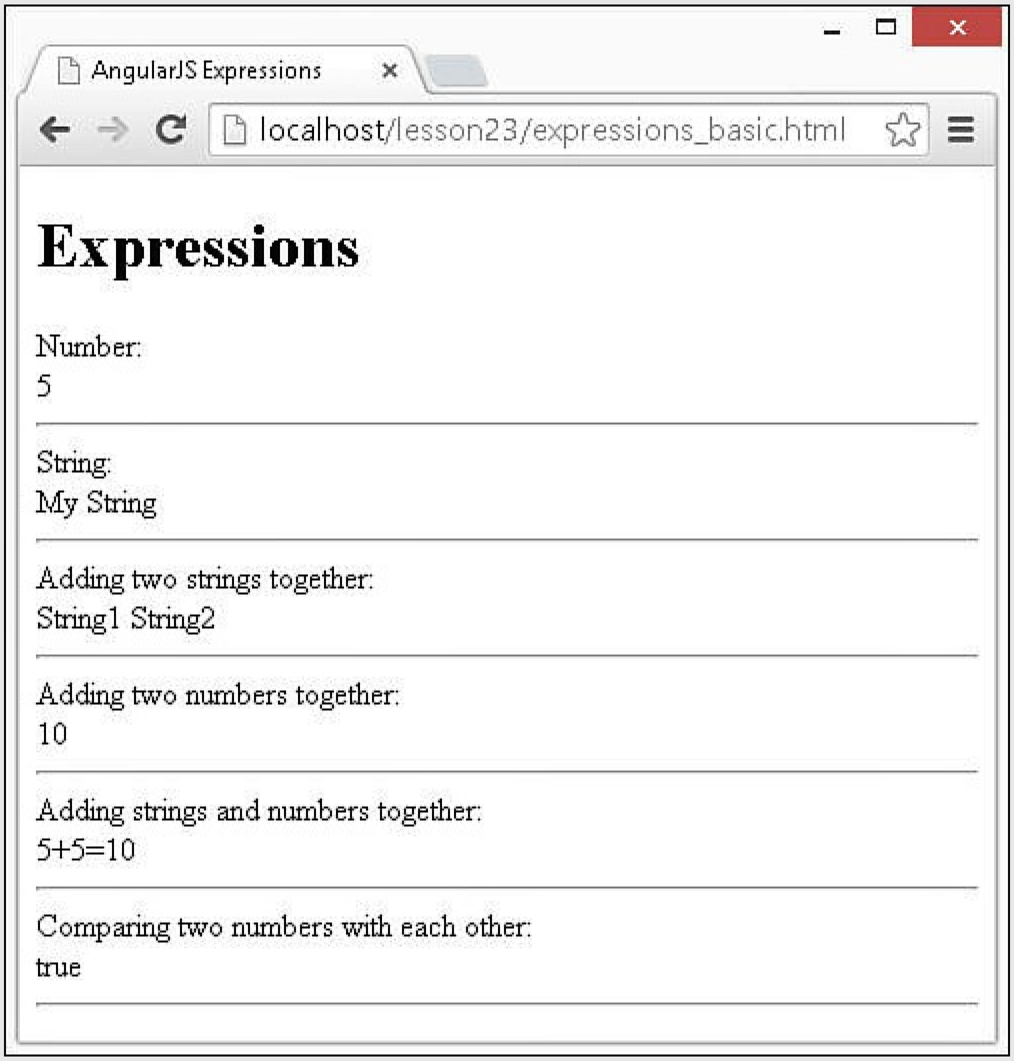
Click here to view code image

```
22        {{5 + '+' + 5 + '='}}{{5+5}}<hr>
```

9. The following line of code from Listing 23.2 uses the expression brackets `{{}}` to display a Boolean value by comparing 5 to 5:

```
24        {{5===5}}<hr>
```

10. Open expressions_basic.html in a browser. Figure 23.1 shows the rendered webpage. Note that numbers and strings are rendered directly to the final view. Adding strings and numbers together enables you to build text strings that are rendered to the view. Also note that using a comparison operator renders the word `true` or `false` to the view.



Expressions

Number:
5

String:
My String

Adding two strings together:
String1 String2

Adding two numbers together:
10

Adding strings and numbers together:
5+5=10

Comparing two numbers with each other:
true

FIGURE 23.1 Using AngularJS expressions that contain strings, numbers, and basic math operations.

LISTING 23.1 expressions_basic.js Basic AngularJS Application Code with Empty Controller

Click here to view code image

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
  });
```

LISTING 23.2 expressions_basic.html Applying Basic Strings and Numbers with Simple Math Operations to an AngularJS Template

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Expressions</title>
    <style>
      p{margin:0px;}
      p:after{color:red;}
    </style>
  </head>
  <body>
    <div ng-controller="myController">
      <h1>Expressions</h1>
      Number:<br>
      {{5}}<hr>
      String:<br>
      {{'My String'}}<hr>
      Adding two strings together:<br>
      {{'String1' + ' ' + 'String2'}}<hr>
      Adding two numbers together:<br>
      {{5+5}}<hr>
      Adding strings and numbers together:<br>
      {{5 + '+' + 5 + '='}}{{5+5}}<hr>
      Comparing two numbers with each other:<br>
      {{5===5}}<hr>
    <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
    <script src="js/expressions_basic.js"></script>
  </body>
</html>
```

Try it Yourself: Interacting with the Scope in Expressions

Now that you have seen some basic AngularJS expressions, let’s take a look at how to interact with the scope inside AngularJS expressions. In the previous example, all the input for the expressions came from explicit strings or numbers. This example illustrates the true power of AngularJS expressions that come from interacting with data in the scope.

Use the following steps to build the application that uses basic expressions:

1. Add the lesson23/expressions_scope.html and lesson23/js/expressions_scope.js files.
2. Add the code shown in Listing 23.3 and Listing 23.4 to the HTML and JavaScript files.
3. The following code in Listing 23.3 initializes the scope with the `speed`, `vehicle`, `new-Speed`, and `newVehicle` variables it needs:

Click here to view code image

```
$scope.speed = 'Slow';
$scope.vehicle = 'Train';
$scope.newSpeed = 'Hypersonic';
$scope.newVehicle = 'Plane';
```

4. The following code implements two functions that accept a string and return that string in uppercase or lowercase characters:

Click here to view code image

```
$scope.upper = function(aString){
  return angular.uppercase(aString);
};
$scope.lower = function(aString){
  return angular.lowercase(aString);
};
```

5. The following code creates a function that sets the values for `$scope.speed` and `$scope.vehicle` based on parameters passed in:

Click here to view code image

```
$scope.setValues = function(speed, vehicle){
  $scope.speed = speed;
  $scope.vehicle = vehicle;
};
```

6. The following code from Listing 23.4 is part of the AngularJS template that allows you to directly access the variables in the scope.

Click here to view code image

```
  {{speed}} {{vehicle}}<hr>
```

7. This next segment of code adds the variables `speed` and `vehicle` from the scope and displays them:

Click here to view code image

```
{{speed + ' ' + vehicle}}<hr>
```

8. The following code displays the results of calling the `lower()` and `upper()` functions in the scope. Notice that `speed` from the scope is passed into `lower()` and a string `Jeep` is passed into `upper()`:

Click here to view code image

```
{{lower(speed)}} {{upper('Jeep')}}<hr>
```

9. The following lines of code use `ng-click` to call `setValues()` in the controller. The first parameter is a string and the second parameter is the variable `newVehicle` from the scope:

Click here to view code image

```
<a ng-click="setValues('Fast', newVehicle)">
  Click to change to Fast {{newVehicle}}</a><hr>
```

10. The following lines of code use `ng-click` to call `setValues()` in the controller. The first parameter is the variable `newSpeed` from the scope and the second parameter is the string `Rocket`:

Click here to view code image

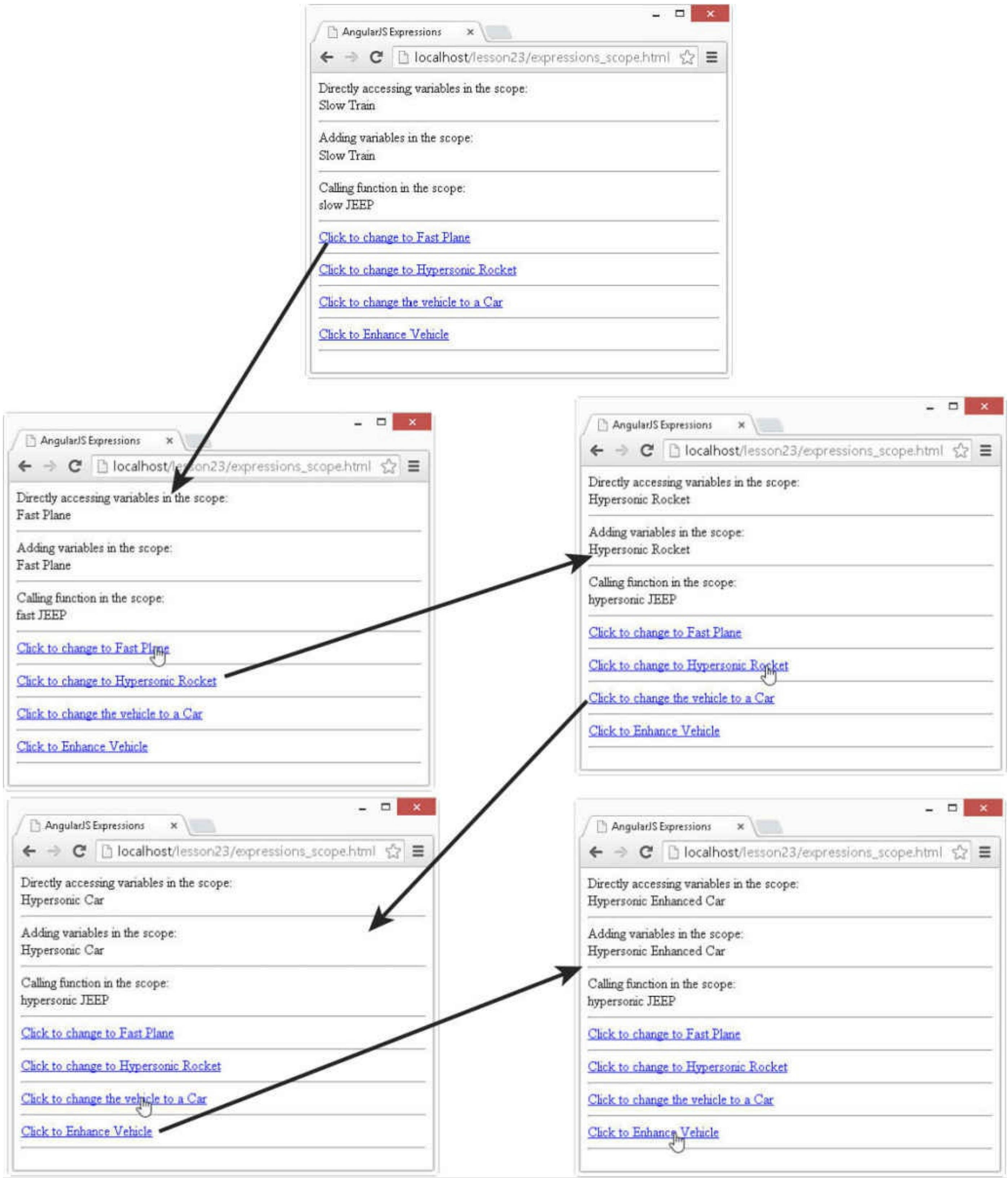
```
<a ng-click="setValues(newSpeed, 'Rocket')">
  Click to change to {{newSpeed}} Rocket</a><hr>
```

11. The following lines of code directly set the variable `vehicle` in the scope when they are clicked on:

Click here to view code image

```
<a ng-click="vehicle='Car'">
  Click to change the vehicle to a Car</a><hr>
<a ng-click="vehicle='Enhanced ' + vehicle">
  Click to Enhance Vehicle</a><hr>
```

12. Open expressions_scope.html in a browser. Figure 23.2 shows the rendered web page based on the expressions. Note that when the links of the page are clicked on, the resulting function calls adjust the scope, which changes how the previously discussed expressions are rendered.


AngularJS Expressions
localhost/lesson23/expressions_scope.html

Directly accessing variables in the scope:
Slow Train

Adding variables in the scope:
Slow Train

Calling function in the scope:
slow JEEP

Click to change to Fast Plane

Click to change to Hypersonic Rocket

Click to change the vehicle to a Car

Click to Enhance Vehicle

AngularJS Expressions
localhost/lesson23/expressions_scope.html

Directly accessing variables in the scope:
Fast Plane

Adding variables in the scope:
Fast Plane

Calling function in the scope:
fast JEEP

Click to change to Fast Plane

Click to change to Hypersonic Rocket

Click to change the vehicle to a Car

Click to Enhance Vehicle

Directly accessing variables in the scope:
Hypersonic Rocket

Adding variables in the scope:
Hypersonic Rocket

Calling function in the scope:
hypersonic JEEP

Click to change to Fast Plane

Click to change to Hypersonic Rocket

Click to change the vehicle to a Car

Click to Enhance Vehicle

Directly accessing variables in the scope:
Hypersonic Car

Adding variables in the scope:
Hypersonic Car

Calling function in the scope:
hypersonic JEEP

Click to change to Fast Plane

Click to change to Hypersonic Rocket

Click to change the vehicle to a Car

Click to Enhance Vehicle

Directly accessing variables in the scope:
Hypersonic Enhanced Car

Adding variables in the scope:
Hypersonic Enhanced Car

Calling function in the scope:
hypersonic JEEP

Click to change to Fast Plane

Click to change to Hypersonic Rocket

Click to change the vehicle to a Car

Click to Enhance Vehicle

FIGURE 23.2 Using AngularJS expressions to represent and use scope data in the AngularJS view.

LISTING 23.3 expressions_scope.js Building a Scope That AngularJS Expressions Can Use

[Click here to view code image]

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
    $scope.speed = 'Slow';
    $scope.vehicle = 'Train';
    $scope.newSpeed = 'Hypersonic';
    $scope.newVehicle = 'Plane';
    $scope.upper = function(aString){
      return angular.uppercase(aString);
    };
    $scope.lower = function(aString){
      return angular.lowercase(aString);
    };
    $scope.setValues = function(speed, vehicle){
      $scope.speed = speed;
      $scope.vehicle = vehicle;
    };
  });
```

LISTING 23.4 expressions_scope.html An AngularJS Template That Uses Expressions in Various Ways to Interact with Data from the Scope Model

[Click here to view code image]

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Expressions</title>
    <style>
      a{color: blue; text-decoration: underline; cursor: pointer}
    </style>
  </head>
  <body>
    <div ng-controller="myController">
      Directly accessing variables in the scope:<br>
      {{speed}} {{vehicle}}<hr>
      Adding variables in the scope:<br>
      {{speed + ' ' + vehicle}}<hr>
      Calling function in the scope:<br>
      {{lower(speed)}} {{upper('Jeep')}}<hr>
      <a ng-click="setValues('Fast', newVehicle)">
        Click to change to Fast {{newVehicle}}</a><hr>
      <a ng-click="setValues(newSpeed, 'Rocket')">
        Click to change to {{newSpeed}} Rocket</a><hr>
      <a ng-click="vehicle='Car'">
        Click to change the vehicle to a Car</a><hr>
      <a ng-click="vehicle='Enhanced ' + vehicle">
        Click to Enhance Vehicle</a><hr>
    <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
```

```
    <script src="js/expressions_scope.js"></script>
  </body>
</html>
```

Try it Yourself: Using JavaScript in AngularJS Expressions

In this final example, we take a look at some additional JavaScript interactions within the scope. As described previously, much of the JavaScript functionality is supported in AngularJS expression. To illustrate this better, the example shows some array manipulation as well as utilizing the JavaScript `Math` object within expressions.

Use the following steps to build the application that uses JavaScript in basic expressions:

1. Add the lesson23/expressions_javascript.html and lesson23/js/expressions_javascript.js files.
2. Add the code shown in Listing 23.5 and Listing 23.6 to the HTML and JavaScript files.
3. The following code in Listing 23.5 creates a simple AngularJS controller that contains two arrays in the scope that will be used in the expressions. Notice that line 3 adds a `Math` variable to the scope by assigning it to `windows.Math`. This is necessary to use the JavaScript `Math` functionality because only the scope variables are available when AngularJS expressions are evaluated:

Click here to view code image

```
01 angular.module('myApp', [])
02   .controller('myController', function($scope) {
03     $scope.Math = window.Math;
04     $scope.myArr = [1];
05     $scope.removedArr = [];
06   });
```

4. Listing 23.6 implements an AngularJS template that uses AngularJS expressions to display the arrays, show the array length, and manipulate the array elements using `push()` and `shift()` directly in the expressions. The following lines display the `myArr` and `removedArr` variables from the scope:

Click here to view code image

```
13         {{myArr}}<hr>
. . .
15         {{removedArr}}<hr>
```

5. The following lines apply JavaScript `push` and `shift` functionality on the `myArr` and `removedArray` variables in the scope directly from within the expression:

Click here to view code image

```
<a ng-click="myArr.push(Math.floor(Math.random()*100 + 1))">
  Click to append a value to the array</a><hr>
<a ng-click="removedArr.push(myArr.shift())">
  Click to remove the first value from the array</a><hr>
```

6. The following line displays the length of `myArry` using the JavaScript length value on the array object:

Click here to view code image

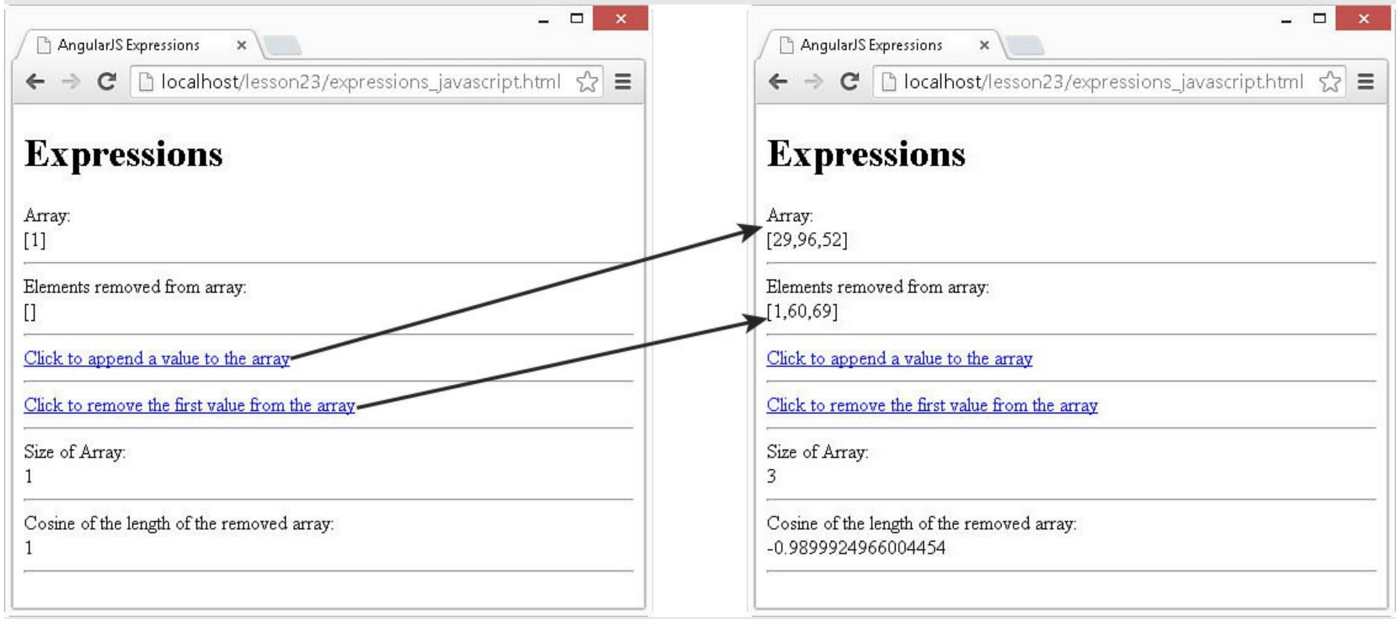
```
{{myArr.length}}<hr>
```

7. The following line uses the `Math` variable in the scope that points to the JavaScript `Math` object to get the cosine of the `length` of `removedArr`:

Click here to view code image

```
{{Math.cos(removedArr.length)}}<hr>
```

8. Open expressions_javascript.html in a browser and click to add and remove elements from the array. Figure 23.3 shows the AngularJS web page rendered. Notice that as the links are clicked, the arrays get adjusted and the expressions are reevaluated.


AngularJS Expressions
localhost/lesson23/expressions_javascript.html
AngularJS Expressions
localhost/lesson23/expressions_javascript.html

Expressions

Array:
[1]

Elements removed from array:
[]

Click to append a value to the array

Click to remove the first value from the array

Size of Array:
1

Cosine of the length of the removed array:
1

Expressions

Array:
[29,96,52]

Elements removed from array:
[1,60,69]

Click to append a value to the array

Click to remove the first value from the array

Size of Array:
3

Cosine of the length of the removed array:
-0.9899924966004454

FIGURE 23.3 Using AngularJS expressions that apply JavaScript array and `Math` operations to interact with scope data.

LISTING 23.5 expressions_javascript.js Building a Scope with Arrays and the

Math Object That AngularJS Expressions Can Use

Click here to view code image

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
    $scope.Math = window.Math;
    $scope.myArr = [1];
    $scope.removedArr = [];
  });
```

LISTING 23.6 expressions_javascript.html An AngularJS Template That Uses Expressions That Contain Arrays and Math Logic in Various Ways to Interact with Data from the Scope Model

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Expressions</title>
    <style>
      a{color: blue; text-decoration: underline; cursor: pointer}
    </style>
  </head>
  <body>
    <div ng-controller="myController">
      <h1>Expressions</h1>
      Array:<br>
        {{myArr}}<hr>
      Elements removed from array:<br>
        {{removedArr}}<hr>
      <a ng-click="myArr.push(Math.floor(Math.random()*100 + 1))">
        Click to append a value to the array</a><hr>
      <a ng-click="removedArr.push(myArr.shift())">
        Click to remove the first value from the array</a><hr>
      Size of Array:<br>
        {{myArr.length}}<hr>
      Cosine of the length of the removed array:<br>
        {{Math.cos(removedArr.length)}}<hr>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
    <script src="js/expressions_javascript.js"></script>
  </body>
</html>
```

Using Filters

A great feature of AngularJS is the capability to implement filters. Filters are a type of provider that hooks into the expression parser and modifies the results of the expression for display in a view—for example, to format time or currency values.

You implement filters inside expressions, using the following syntax:

```
{{ expression | filter}}
```

If you chain multiple filters together, they are executed in the order in which you specify them:

Click here to view code image

```
{{ expression | filter | filter }}
```

Some filters allow you to provide input in the form of function parameters. You add these parameters by using the following syntax:

Click here to view code image

```
{{ expression | filter:parameter1:parameter2 }}
```

Also, you can add filters, which are providers, to controllers and services by using dependency injection. The filter provider name is the name of the filter plus `Filter`. For example, the currency filter provider is named `currencyFilter`. The filter provider acts as a function, with the expression as the first parameter and any additional parameters after that. The following code defines a controller that injects `currencyFilter` and uses it to format results. Notice that `currencyFilter` is added to the dependency injection for the controller and is called as a function:

Click here to view code image

```
controller('myController', ['$scope', 'currencyFilter',
                            function($scope, myCurrencyFilter){
  $scope.getCurrencyValue = function(value){
    return myCurrencyFilter(value, "$USD");
  };
}]);
```

Using Built-in Filters

AngularJS provides several types of filters that enable you to easily format strings, objects, and arrays in AngularJS templates. Table 23.1 lists the built-in filters provided with AngularJS.

Filter	Description
`currency[:symbol]`	Formats a number as currency, based on the `symbol` value provided. If no `symbol` value is provided, the default symbol for the locale is used. For example: `{{123.46 | currency:"$USD" }}`
`filter:exp:compare`	Filters the expression with the value of the `exp` parameter, based on the value of `compare`. The `exp` parameter can be a string, an object, or a function. The `compare` parameter can be a function that accepts expected and actual values and returns `true` or `false`. The `compare` parameter can also be a Boolean, where `true` is a strict comparison of `actual===expected` or `false` is a relaxed comparison that checks whether the value is a subset of the actual value. For example: `{{"Some Text to Compare" | filter:"text":false`
`json`	Formats a JavaScript object into a JSON string. For example: `{{ {'name':'Brad'} | json }}`
`limitTo:limit`	Limits the data represented in the expression by the `limit` amount. If the expression is a `String`, it is limited by number of characters. If the result of the expression is an `Array`, it is limited by the number of elements. For example: `{{ ['a','b','c','d'] | limitTo:2 }}`
`lowercase`	Outputs the result of the expression as lowercase.
`uppercase`	Outputs the result of the expression as uppercase.

`number[:fraction]`	Formats the number as text. If a `fraction` parameter is specified, the number of decimal places displayed is limited to that size. For example: `{{ 123.4567 | number:3 }}`
`orderBy:exp:reverse`	Orders an array based on the `exp` parameter. The `exp` parameter can be a function that calculates the value of an item in the array or a string that specifies an object property in an array of objects. The `reverse` parameter is `true` for descending order or `false` for ascending.
`date[:`*`format`*`]`	Formats a JavaScript `Date` object, a timestamp, or date ISO 8601 date strings, using the *`format`* parameter. For example: `{{1389323623006 | date:'yyyy-MM-dd HH:mm:ss Z'}}` The `format` parameter uses the following date formatting characters: ▶ `yyyy`: Four-digit year ▶ `yy`: Two-digit year since 2000 ▶ `MMMM`: Month in year, `January–December` ▶ `MMM`: Month in year, `Jan–Dec` ▶ `MM`: Month in year, padded, `01–12` ▶ `M`: Month in year, `1–12` ▶ `dd`: Day in month, padded, `01–31` ▶ `d`: Day in month, `1–31` ▶ `EEEE`: Day in week, `Sunday–Saturday` ▶ `EEE`: Day in week, `Sun–Sat` ▶ `HH`: Hour in day, padded, `00–23` ▶ `H`: Hour in day, `0–23`

- hh: Hour in am/pm, padded, 01–12
- h: Hour in am/pm, 1–12
- mm: Minute in hour, padded, 00–59
- m: Minute in hour, 0–59
- ss: Second in minute, padded, 00–59
- s: Second in minute, 0–59
- .sss or ,sss: Millisecond in second, padded, 000–999
- a: am/pm marker
- Z: Four-digit time zone offset, -1200–+1200

The format string for date can also be one of the following predefined names. The format that follows is shown as en_US but will match the locale of the AngularJS application:

- medium: 'MMM d, y h:mm:ss a'
- short: 'M/d/yy h:mm a'
- fullDate: 'EEEE, MMMM d,y'
- longDate: 'MMMM d, y'
- mediumDate: 'MMM d, y'
- shortDate: 'M/d/yy'
- mediumTime: 'h:mm:ss a'
- shortTime: 'h:mm a'

TABLE 23.1 Filters That Modify Expressions in AngularJS Templates

Try it Yourself: Using Filters to Control How Data Is Rendered

In this example, you play around with the built-in AngularJS filters. The code in Listing 23.7 defines an AngularJS application with various types of data in the scope. The code in Listing 23.8 implements an AngularJS template that utilizes some basic filters to display the values.

Use the following steps to create the application and implement the filters in AngularJS expressions:

1. Add the lesson23/filters.html and lesson23/js/filters.js files.
2. Add the code shown in Listing 23.7 and Listing 23.8 to the HTML and JavaScript files.
3. The following code implements a controller with scope values that include date, object, string, and array objects:

Click here to view code image

```
  .controller('myController', function($scope) {
    $scope.currentDate = new Date();
    $scope.JSONObj = { title: "myTitle" };
    $scope.word="Supercalifragilisticexpialidocious";
    $scope.days=['Monday', 'Tuesday', 'Wednesday',
                    'Thursday', 'Friday'];
  });
```

4. The following code from Listing 23.8 implements several filters. The following line implements the `number` filter and limits the number of decimal places displayed to 3:

Click here to view code image

```
          Number: {{123.45678|number:3}}<br>
```

5. The following code implements the `currency` filter, which accepts a number, places a dollar sign in front, and limits the decimal value to two places format for U.S. currency:

Click here to view code image

```
          Currency: {{123.45678|currency:"$"}}<br>
```

6. The following code implements the `date` filter, which formats the date object `currentDate` in the scope and displays it:

Click here to view code image

```
          Date: {{ currentDate | date:'yyyy-MM-dd HH:mm:ss Z'}}<br>
```

7. Remember the JSONobj from earlier? The following code utilizes that and fills in the expression:

Click here to view code image

```
          JSON: {{ JSONObj | json }}<br>
```

8. The following code uses the limit filter to display only 3 elements of the days array:

Click here to view code image

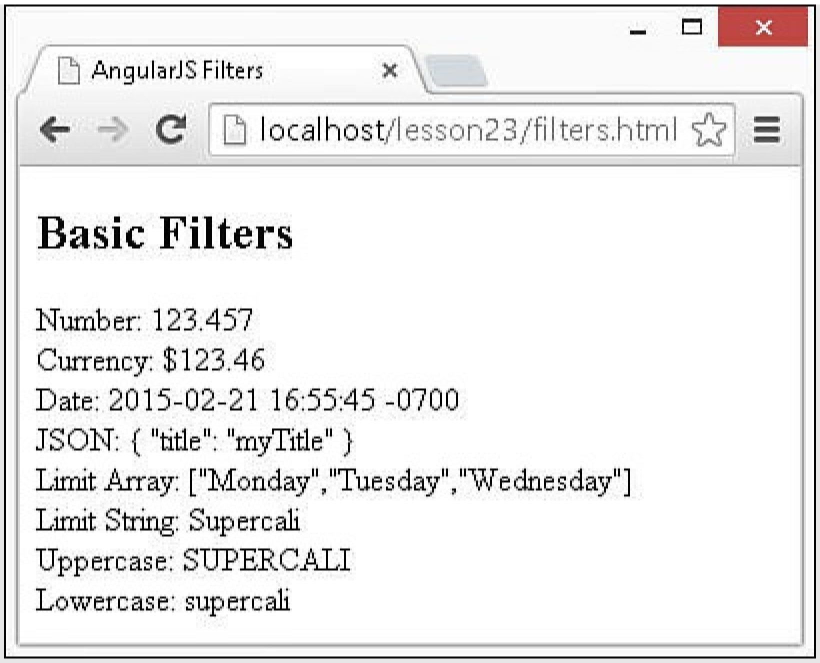
```
          Limit Array: {{ days | limitTo:3 }}<br>
```

9. The following lines of code us the `limit`, `uppercase`, and `lowercase` filter to modify how the string variable `word` from the scope is displayed:

Click here to view code image

```
          Limit String: {{ word | limitTo:9 }}<br>
          Uppercase: {{ word | uppercase | limitTo:9 }}<br>
          Lowercase: {{ word | lowercase | limitTo:9 }}
```

10. Load filters.html into a browser. Figure 23.4 shows the output generated by the filters.



Basic Filters

Number: 123.457
Currency: $123.46
Date: 2015-02-21 16:55:45 -0700
JSON: { "title": "myTitle" }
Limit Array: ["Monday","Tuesday","Wednesday"]
Limit String: Supercali
Uppercase: SUPERCALI
Lowercase: supercali

FIGURE 23.4 Using AngularJS filters to modify data before displaying it in the AngularJS view.

LISTING 23.7 filters.js Building a Scope That AngularJS Filters Can Use

Click here to view code image

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
    $scope.currentDate = new Date();
    $scope.JSONObj = { title: "myTitle" };
    $scope.word="Supercalifragilisticexpialidocious";
    $scope.days=['Monday', 'Tuesday', 'Wednesday',
                 'Thursday', 'Friday'];
  });
```

LISTING 23.8 filters.html An AngularJS Template That Implements Various Types of Filters to Modify Data Displayed in the Rendered View

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Filters</title>
  </head>
  <body>
    <div ng-controller="myController">
      <h2>Basic Filters</h2>
      Number: {{123.45678|number:3}}<br>
      Currency: {{123.45678|currency:"$"}}<br>
      Date: {{ currentDate | date:'yyyy-MM-dd HH:mm:ss Z'}}<br>
      JSON: {{ JSONObj | json }}<br>
      Limit Array: {{ days | limitTo:3 }}<br>
      Limit String: {{ word | limitTo:9 }}<br>
      Uppercase: {{ word | uppercase | limitTo:9 }}<br>
      Lowercase: {{ word | lowercase | limitTo:9 }}
    <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
    <script src="js/filters.js"></script>
  </body>
</html>
```

Try it Yourself: Using Filters to Implement Ordering and Filtering

A very common use of filters is to order or filter out dynamic elements built using the `ng-repeat` directive from JavaScript arrays. This section provides an example of implementing `orderBy` filters to generate a table that can be sorted by column and filtered by a string from an `<input>` element.

Use the following steps to create the AngularJS application that utilizes the ordering and sorting filters on tabular data:

1. Add the lesson23/filter_sort.html and lesson23/js/filter_sort.js files.
2. Add the code shown in Listing 23.9 and Listing 23.10 to the HTML and JavaScript files.
3. The code in Listing 23.9 implements an AngularJS application with a single controller. The following lines of code inject the `filterFilter` provider into the controller:

Click here to view code image

```
.controller('myController', ['$scope', 'filterFilter',
                            function($scope, filterFilter) {
```

4. The following lines then define several variables, including `planes`, which stores an array of objects that will act as the data source for the application; `filteredPlanes`, which will store the filtered version of planes that will be displayed; `reverse`, which keeps track of the sort order; and `column`, which keeps track of which column you are sorting on:

Click here to view code image

```
  $scope.planes = [
    {make: 'Boeing', model: '777', capacity: 440},
. . .
  $scope.filteredPlanes = $scope.planes;
  $scope.reverse = true;
```

```
$scope.column = 'make';
```

5. The following lines of code define the `setSort()` function, which is used to update the `column` and `reverse` values:

Click here to view code image

```
$scope.setSort = function(column){
  $scope.column = column;
  $scope.reverse = !$scope.reverse;
};
```

6. The following lines define the `filterString` value, which filters the objects to include in `filteredPlanes`. They also define the `setFilter()` function, which calls the `filterFilter()` provider to limit the items in `filteredPlanes` to the ones that loosely match `filterString`:

Click here to view code image

```
$scope.filterString = '';
$scope.setFilter = function(value){
  $scope.filteredPlanes =
    filterFilter($scope.planes, $scope.filterString);
};
```

7. The code in Listing 23.10 implements a template that includes a text input and button to set a string filter as well as a table that allows the sorting by clicking a column header. The following lines show the text `<input>` that binds to the filterString value and a button `<input>` that calls `setFilter()` when clicked:

Click here to view code image

```
<input type="text" ng-model="filterString">
<input type="button" ng-click="setFilter()" value="Filter">
```

8. The following lines show that the table headers apply `ng-click` directives to call `setSort()` to set the sort column. The sort order is reversed each time `setSort()` is called:

Click here to view code image

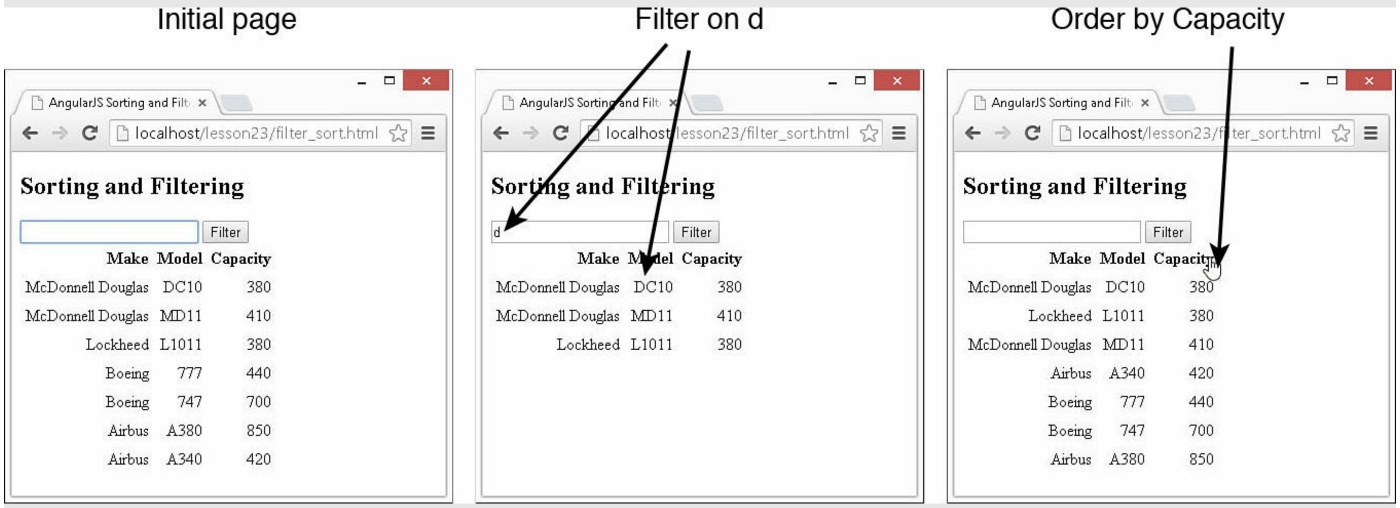
```
<th ng-click="setSort('make')">Make</th>
<th ng-click="setSort('model')">Model</th>
<th ng-click="setSort('capacity')">Capacity</th>
```

9. The following lines implement the rows of the table by using the `ng-repeat` directive. Notice that the `ng-repeat` directive uses the `orderBy` filter to specify the column name and reverse values set by the `setSort()` function:

Click here to view code image

```
        <tr ng-repeat=
            "plane in filteredPlanes | orderBy:column:reverse">
          <td>{{plane.make}}</td>
          <td>{{plane.model}}</td>
          <td>{{plane.capacity}}</td>
        </tr>
```

10. Open filter_sort.html in a browser. Figure 23.5 shows the resulting web page. Clicking a column header sorts the table, and adding text to the filter box filters the contents of the table.



Initial page
Filter on d
Order by Capacity
AngularJS Sorting and Filt
localhost/lesson23/filter_sort.html
Filter

Sorting and Filtering

Make	Model	Capacity
McDonnell Douglas	DC10	380
McDonnell Douglas	MD11	410
Lockheed	L1011	380
Boeing	777	440
Boeing	747	700
Airbus	A380	850
Airbus	A340	420

Sorting and Filtering

d

Make	Model	Capacity
McDonnell Douglas	DC10	380
McDonnell Douglas	MD11	410
Lockheed	L1011	380

Sorting and Filtering

Make	Model	Capacity
McDonnell Douglas	DC10	380
Lockheed	L1011	380
McDonnell Douglas	MD11	410
Airbus	A340	420
Boeing	777	440
Boeing	747	700
Airbus	A380	850

FIGURE 23.5 Using AngularJS filters to filter and order items in a table in the AngularJS view.

LISTING 23.9 filter_sort.js AngularJS Module that Builds a List of Planes and Provides Functionality to Sort and Order the List

Click here to view code image

```
angular.module('myApp', [])
  .controller('myController', ['$scope', 'filterFilter',
                              function($scope, filterFilter) {
    $scope.planes = [
      {make: 'Boeing', model: '777', capacity: 440},
      {make: 'Boeing', model: '747', capacity: 700},
      {make: 'Airbus', model: 'A380', capacity: 850},
      {make: 'Airbus', model: 'A340', capacity: 420},
      {make: 'McDonnell Douglas', model: 'DC10', capacity: 380},
      {make: 'McDonnell Douglas', model: 'MD11', capacity: 410},
      {make: 'Lockheed', model: 'L1011', capacity: 380}];
    $scope.filteredPlanes = $scope.planes;
    $scope.reverse = true;
    $scope.column = 'make';
    $scope.setSort = function(column){
      $scope.column = column;
      $scope.reverse = !$scope.reverse;
    };
    $scope.filterString = '';
    $scope.setFilter = function(value){
      $scope.filteredPlanes =
        filterFilter($scope.planes, $scope.filterString);
    };
  }]);
```

LISTING 23.10 filter_sort.html An AngularJS Template That Implements `filter` and `orderBy` Filters to Order and Filter Items in a Table View

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Sorting and Filtering</title>
    <style>
      table{text-align:right;}
      td,th{padding:3px;}
      th{cursor:pointer;}
    </style>
  </head>
  <body>
    <div ng-controller="myController">
      <h2>Sorting and Filtering</h2>
      <input type="text" ng-model="filterString">
      <input type="button" ng-click="setFilter()" value="Filter">
      <table>
      <tr>
        <th ng-click="setSort('make')">Make</th>
        <th ng-click="setSort('model')">Model</th>
        <th ng-click="setSort('capacity')">Capacity</th>
      </tr>
      <tr ng-repeat=
          "plane in filteredPlanes | orderBy:column:reverse">
        <td>{{plane.make}}</td>
        <td>{{plane.model}}</td>
        <td>{{plane.capacity}}</td>
      </tr>
      </table>
    <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
    <script src="js/filter_sort.js"></script>
  </body>
</html>
```

Creating Custom Filters

AngularJS enables you to create your own custom filter provider and then use it in expressions, controllers, and services as if it were a built-in filter. AngularJS provides the `filter()` method to create a filter provider and register it with the dependency injector server.

The `filter()` method accepts a name for the filter as the first argument and a function for the second argument. The filter function should accept the expression input as the first parameter and any additional parameters following that. For example:

Click here to view code image

```
filter('myFilter', function(){
  return function(input, param1, param2){
    return <<modified input>>;
  };
});
```

Inside the filter function, you can change the value of the input in any way you like. Whatever value is returned from the filter function is returned as the expression results.

Try it Yourself: Creating and Implementing a Custom Filter

In this exercise, you build and use a custom AngularJS filter that applies a censor to words in text. The purpose of this exercise is to illustrate how to build and use a custom AngularJS template.

The code in Listings 23.11 and 23.12 create a custom filter function that censors words from a string and allows for a replacement value as an optional parameter.

Use the following steps to build the application that creates and applies a custom filter:

1. Add the lesson23/filter_custom.html and lesson23/filter_custom.js files.
2. Add the code shown in Listing 23.11 and Listing 23.12 to the HTML and JavaScript files.
3. The following code in Listing 23.11 defines the filter named `censor`. Notice that the filter returns a function that accepts an input and replacement value and then iterates through the censored words and replaces the text with the replacement value:

Click here to view code image

```
.filter('censor', function() {
  return function(input, replacement) {
    var cWords = ['bad', 'evil', 'dark'];
    var out = input;
    for(var i=0; i<cWords.length; i++){
      var regex = new RegExp(cWords[i], "gi");
      out = out.replace(regex, replacement);
    }
    return out;
  };
})
```

4. The following code implements the controller for the application. Notice that the `censorFilter` is injected into the controller. Also look at the function called `filterText` that applies the `censorFilter` to a string of text manually:

Click here to view code image

```
.controller('myController', ['$scope', 'censorFilter',
                            function($scope, myCensorFilter) {
  $scope.censorText = "***";
  $scope.phrase="This is a bad phrase.";
  $scope.txt = "Click to filter out dark and evil.";
  $scope.filterText = function(){
    $scope.txt = myCensorFilter($scope.txt, '<<censored>>');
  };
}]);
```

5. The code in Listing 23.12 implements a template that utilizes the filter in a couple of ways. The following line implements the filter on the `phrase` value in the scope by passing the `censorText` variable from the scope as the replacement string value:

Click here to view code image

```
    {{phrase | censor:censorText}}<br>
```

6. The following line of code from Listing 23.12 implements the filter on a string by passing the string `"happy"` in as the replacement string:

Click here to view code image

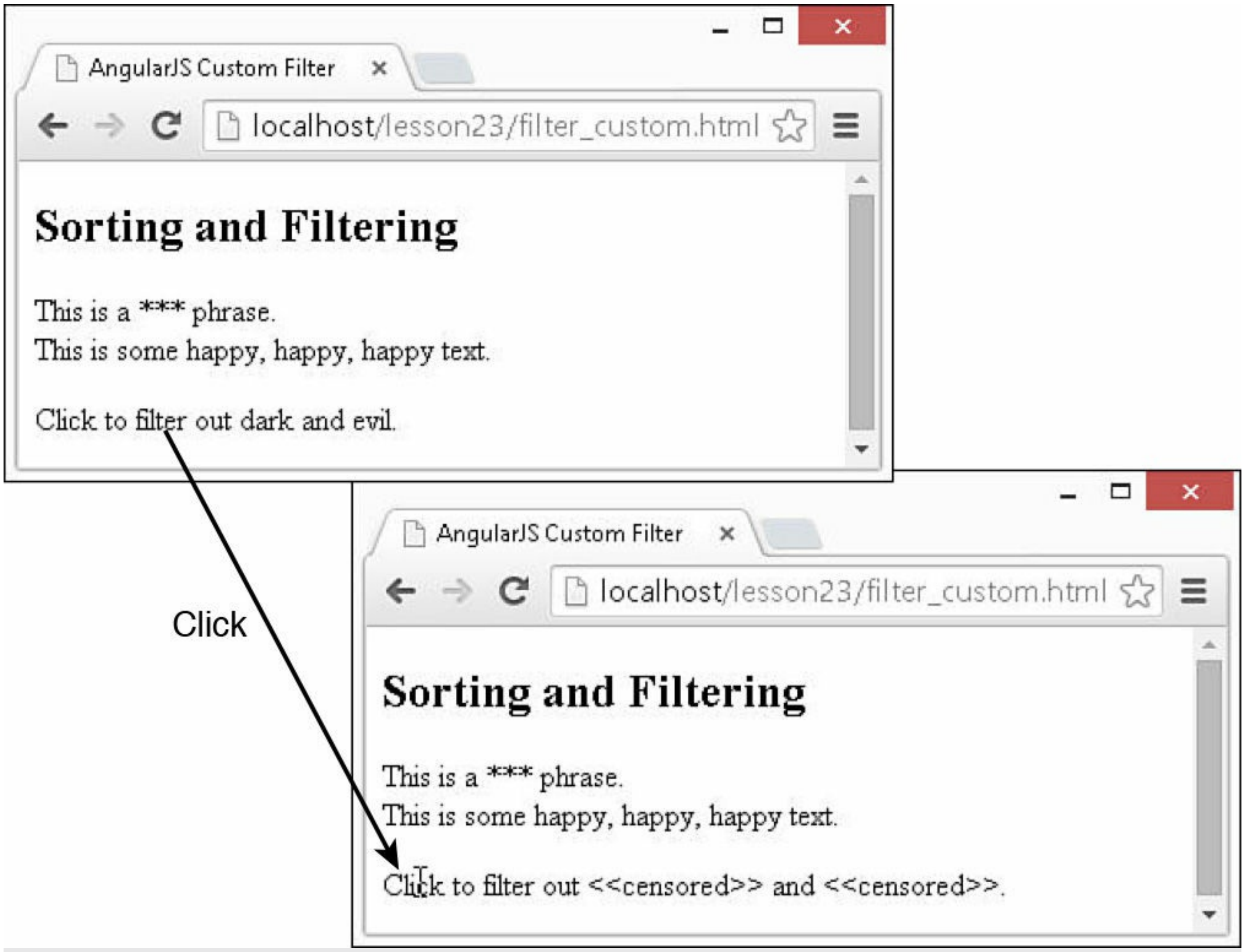
```
    {{"This is some bad, dark, evil text." | censor:"happy"}}
```

7. The following line of code uses `ng-click` to call the `filterText()` function in the controller, which will programmatically call the filter and pass `<<censored>>` in as the replacement string:

Click here to view code image

```
    <p ng-click="filterText()">{{txt}}</p>
```

8. Load the filter_custom.HTML file in a browser to see the behavior shown in Figure 23.6.


AngularJS Custom Filter

localhost/lesson23/filter_custom.html

Sorting and Filtering

This is a *** phrase.
This is some happy, happy, happy text.

Click to filter out dark and evil.

Click

AngularJS Custom Filter

localhost/lesson23/filter_custom.html

Sorting and Filtering

This is a *** phrase.
This is some happy, happy, happy text.

Click to filter out <<censored>> and <<censored>>.

FIGURE 23.6 Creating and using custom filters in an AngularJS view.

LISTING 23.11 filter_custom.js Implementing a Custom Filter Provider in AngularJS

[Click here to view code image]

```
angular.module('myApp', [])
  .filter('censor', function() {
    return function(input, replacement) {
      var cWords = ['bad', 'evil', 'dark'];
      var out = input;
      for(var i=0; i<cWords.length; i++){
        var regex = new RegExp(cWords[i], "gi");
        out = out.replace(regex, replacement);
      }
      return out;
    };
  })
  .controller('myController', ['$scope', 'censorFilter',
                              function($scope, myCensorFilter) {
```

```
    $scope.censorText = "***";
    $scope.phrase="This is a bad phrase.";
    $scope.txt = "Click to filter out dark and evil.";
    $scope.filterText = function(){
      $scope.txt = myCensorFilter($scope.txt, '<<censored>>');
    };
  }]);
```

LISTING 23.12 filter_custom.html An AngularJS Template That Uses a Custom Filter

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Custom Filter</title>
  </head>
  <body>
    <div ng-controller="myController">
      <h2>Sorting and Filtering</h2>
      {{phrase | censor:censorText}}<br>
      {{"This is some bad, dark, evil text." | censor:"happy"}}
      <p ng-click="filterText()">{{txt}}</p>
    <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
    <script src="js/filter_custom.js"></script>
  </body>
</html>
```

Summary

AngularJS templates are simple to implement yet very powerful and extensive. This lesson discusses the components of AngularJS templates and how they work together to extend HTML DOM behavior and functionality. Expressions are bits of JavaScript code contained in `{{}}` brackets or within directive definitions in the AngularJS template. Expressions have access to the scope, so you can render scope values to the view.

Filters act as modifiers to expressions and enable you to format expression results for specific purposes. AngularJS provides several built-in filters, such as for currency and date formatting. You can also create your own custom filters that provide any formatting or modifications you want apply before rendering data to the page. You inject filters as providers into the injector service and can therefore access them inside controllers and templates, using dependency injection. This means you have access to filters within your JavaScript code as well.

Q&A

Q. Why use a custom filter rather than a simple function in the controller?

A. The AngularJS mentality is to allow as much of the intent as possible to be expressed in the HTML templates. This makes applications easier to design and follow. It is also much easier to reuse a filter than it is a custom function.

Q. Why doesn't the HTML standard adopt all of the functionality provided by AngularJS as part of the HTML spec?

A. HTML will likely add some of the functionality provided by AngularJS as time goes on. However, it is too difficult to move a massively adopted specification such as HTML at the rate developers need. AngularJS is the next best thing.

Workshop

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

Quiz

1. How do you compare scope values, `valueA` and `valueB`, in an AngularJS expression?
2. What is the output of the following AngularJS expression: `{{5 + '+' + 5}}`?
3. How would you use AngularJS to properly display the U.S. dollar amount stored in the `myMoney` variable from the scope?
4. What AngularJS filter would you use to sort rows in an HTML table?

Quiz Answers

1. `{{valueA===valueB}`
2. `5+5`
3. `{{myMoney | currency:"$USD" }}`
4. `orderBy`

Exercises

1. Modify the code in Listings 23.1 and 23.2 to store a string variable named `myScopeText` in the scope and then display it in the template using the following expression. Then do the same for a number variable in the scope named `myScopeNumber`:

```
{{myScopeText}}
```

2. Look at the date filter listed in Table 23.1 and then modify the date filter in Listing 23.8 to change the format of the date string displayed.
3. Modify the code in Listings 23.11 and 23.12 by adding an additional string to the scope to be censored. Then add a statement to the template that specifies that variable name in the scope for the word and a string as the replacement value. For example:

Click here to view code image

```
{{myNewPhrase | censor:"XXX" }}<br>
```

Lesson 24. Implementing Directives in AngularJS Views

What You’ll Learn in This Lesson:

- How AngularJS directives relate to HTML
- Ways to implement AngularJS directives in your web pages
- Using AngularJS directives to interact with mouse and keyboard events
- Which form elements are extended by AngularJS directives
- How to build form interactions into your AngularJS applications

One of the most powerful features of AngularJS is directives. Directives extend the behavior of HTML, enabling you to create custom HTML elements, attributes, and classes with functionality specific to an application. AngularJS provides several built-in directives. In fact, the majority of the AngularJS library is built-in directives. These directives provide the capability to interact with form elements, bind data in the scope to the view, and interact with browser events.

This lesson discusses the built-in directives and how to implement them in AngularJS templates. You learn how to apply these directives in your AngularJS templates and support them in the back-end controllers to quickly turn the rendered view into an interactive application.

Understanding Directives

Directives are a combination of AngularJS template markups and supporting JavaScript code. AngularJS directive markups can be HTML attributes, element names, or CSS classes. The JavaScript directive code defines the template data and behavior of the HTML elements.

The AngularJS compiler traverses the template DOM and compiles all directives. Then it links the directives by combining a directive with a scope to produce a new live view. The live view contains the DOM elements and functionality defined in the directive.

Using Built-In Directives

Most of the AngularJS functionality that you need to implement in HTML elements is provided in the built-in directives. These directives are provided by the library and are available when the AngularJS JavaScript library is loaded.

Directives provide a variety of support for AngularJS applications. The following sections describe most of the AngularJS directives, which fall into the following categories:

- Directives that support AngularJS functionality
- Directives that extend form elements
- Directives that bind the page elements to values in the scope model
- Directives that bind page events to controllers

Each of the following sections includes a table containing the related directives along with a basic description. You do not need to understand all these directives right now; the tables are there more for reference. Subsequent sections and lessons provide sample code for using many of these directives.

Directives That Support AngularJS Functionality

Several directives provide support for AngularJS functionality. These directives do everything from bootstrapping an application to ensuring that Boolean expressions that AngularJS requires are preserved in the DOM.

Table 24.1 lists these directives and describes the behavior and usage of each.

Directive	Description
`ngApp`	This directive is used to bootstrap an application to a root element. This attribute is set to the name of the AngularJS module to use as the application root, and the HTML element that contains it acts as the compilation root for the template. For example, the following sets module `myApp` as the application in the `<html>` element: `<html ng-app="myApp">`
`ngCloak`	When this attribute is present in an element, that element is not displayed until after the AngularJS template has been fully compiled. Otherwise, the raw form of the element with the template code is displayed.
`ngController`	This directive attaches a controller to this element in the view to create a new scope, as described in earlier lessons. For example: `<div ng-controller="myController">`
`ngHref`	This is an option you can use instead of using the `href` attribute, which might be broken if the user clicks the link before the expression has been evaluated if you include template syntax such as `{{hash}}`.
`ngInclude`	This directive automatically fetches, compiles, and includes an external HTML fragment from the server. Using it is a great way to include partial HTML data from server-side scripts. For example: `<div ng-include="/info/sidebar.html">`
`ngList`	This directive converts an `Array` object in the scope into a delimiter-separated string. (Comma is the default delimiter.) For example, if the scope contains an array named `items`, the displayed value in the following `<input>` would be `item1, item2, item3,...`: `<input ng-model="items" ng-list=",">`

`ngNonBindable`	When this directive is present in an element, AngularJS does not compile or bind the contents of the element during compilation. This is useful if you are trying to display code in the element. For example: `<p ng-non-bindable>Expression Syntax: {{exp}}</p>`
`ngOpen`	Browsers are not required to preserve Boolean attributes of elements. If this attribute is present, it is `true`. This directive enables you to preserve the `true/false` state of an element by testing the existence of the attribute. For example, the following applies `ngOpen` based on the `open` value in the scope: `<details ng-open="open">`

`ngPluralize`	This directive enables you to display messages according to the `en-US` localization rules bundled with AngularJS. You can configure `ngPluralize` by adding the `count` and `when` attributes, as shown here:
	`<p ng-pluralize count="itemCount"`
	`   when="{'0': 'Cart is empty.',`
	`          'one': 'Purchase 1 item.',`
	`          'other': 'Purchase {{itemCount}} items.'}">`
	`</p>`

`ngReadonly`	Similar to `ngOpen` but for the `readonly` Boolean value. For example, the following applies `ngReadonly` based on the `notChangeable` value in the scope: `<input type="text" ng-readonly="notChangeable">`
`ngRequired`	This directive is similar to `ngOpen` but for the `required` Boolean value for `<input>` elements in a form. For example, the following applies `ngRequired` based on the required value in the scope: `<input type="text" ng-required="required">`

`ngSelected`	This directive is similar to `ngOpen` but for the `selected` Boolean value. For example, the following applies `ngSelected` based on the selected value in the scope:
	`<option id="optionA" ng-selected="selected">`
	`  Option A`
	`</option>`

`ngSrc`	You can use this directive instead of using the `src` attribute, which is broken until the expression has been evaluated, if you include template syntax such as `{{username}}`. For example: `<img ng-src="/images/{{username}}/test.jpg" />`
`ngSrcset`	You can use this directive instead of using the `srcset` attribute, which is broken before the expression has been evaluated. For example: `<img ng-srcset="/images/{{username}}/test.jpg 2x" />`
`ngTransclude`	This directive marks the element as the transclude point for directives that use the transclude option to wrap other elements.
`ngView`	This directive includes a rendered template of the current route into the main layout file. Routes are discussed in Lesson 27.
`script`	This directive loads the content of a `script` tag with `next/ng-template` so that it can be used by `ngInclude`, `ngView`, or other template directives.

TABLE 24.1 Directives That Support AngularJS Template Functionality

The directives in Table 24.1 are used in different ways in various parts of the code. You have already seen a few of them, such as `ngApp` and `ngController`, used in previous examples. Some are fairly intuitive, such as using `ng-src` instead of `src` when implementing `<img>` elements in a template. Others will be used in various examples in subsequent lessons.

I did want to give you an example at this point of using the `ngInclude` directive. This little directive is simple to employ and can be used for a variety of purposes, especially if you are trying to introduce AngularJS into an existing system. In this example, we use `ngInclude` to swap the banner bar at the top of a basic web page by loading different partial HTML files from the server.

Try it Yourself: Using `ng-include` to Dynamically Change Content

The code in Listing 24.1 implements a basic AngularJS controller that stores an HTML filename in a variable named `titleBar`. The code in Listing 24.2 implements an AngularJS template that includes a couple of links at the top to switch pages and a `<div>` element on line 24 that uses `ng-include` to change the contents of the div to the file specified by `titleBar`.

Use the following steps to build the application that uses `ng-include` to quickly switch between two partial templates:

1. Add the lesson24/directive_angular_include.html, lesson24/large_title.html, lesson24/small_title.html, and lesson24/js/directive_angular_include.js files.
2. Add the code shown in Listing 24.1, Listing 24.2, Listing 24.3, and Listing 24.4 to the HTML and JavaScript files.

3. The code in Listing 24.1 implements a simple application that sets the scope value of `titleBar` to `small_tile.html`, which is the AngularJS partial template for the small banner.
4. The code in Listing 24.2 implements the AngularJS template that loads the application. The following lines in Listing 24.2 create links that set the value of `titleBar` in the scope to one of the two template partial files listed in Listings 24.3 and 24.4:

Click here to view code image

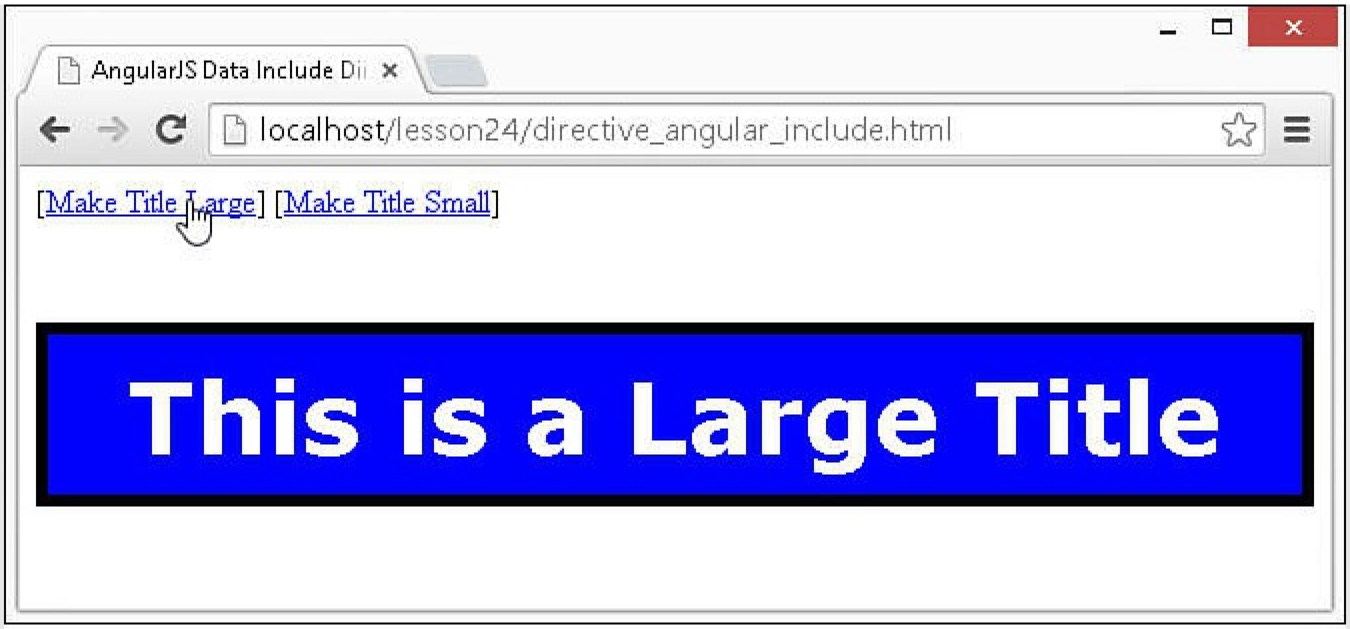
```
[<a ng-click="titleBar='large_title.html'">Make Title Large</a>]
[<a ng-click="titleBar='small_title.html'">Make Title Small</a>]
```

5. The following line of code in Listing 24.2 loads uses the `ng-include` directive to fill the contents of a `<div>` element with whichever template partial `titleBar` is set to:

Click here to view code image

```
<div ng-include="titleBar"></div>
```

6. The different versions of the title bar are located in the files shown in Listings 24.3 and 24.4. Basically, these files contain a `<p>` element that has either the `large` or `small` class assigned to it. The class definitions are located in the `<style>` element of Listing 24.2.
7. Open the directive_angular_include.html file in a browser. Figure 24.1 shows the two title banners. When the links are clicked to switch banners, the contents of the `<div>` element in the original are replaced by the new HTML file being loaded.


AngularJS Data Include Di
localhost/lesson24/directive_angular_include.html
[Make Title Large] [Make Title Small]
This is a Large Title

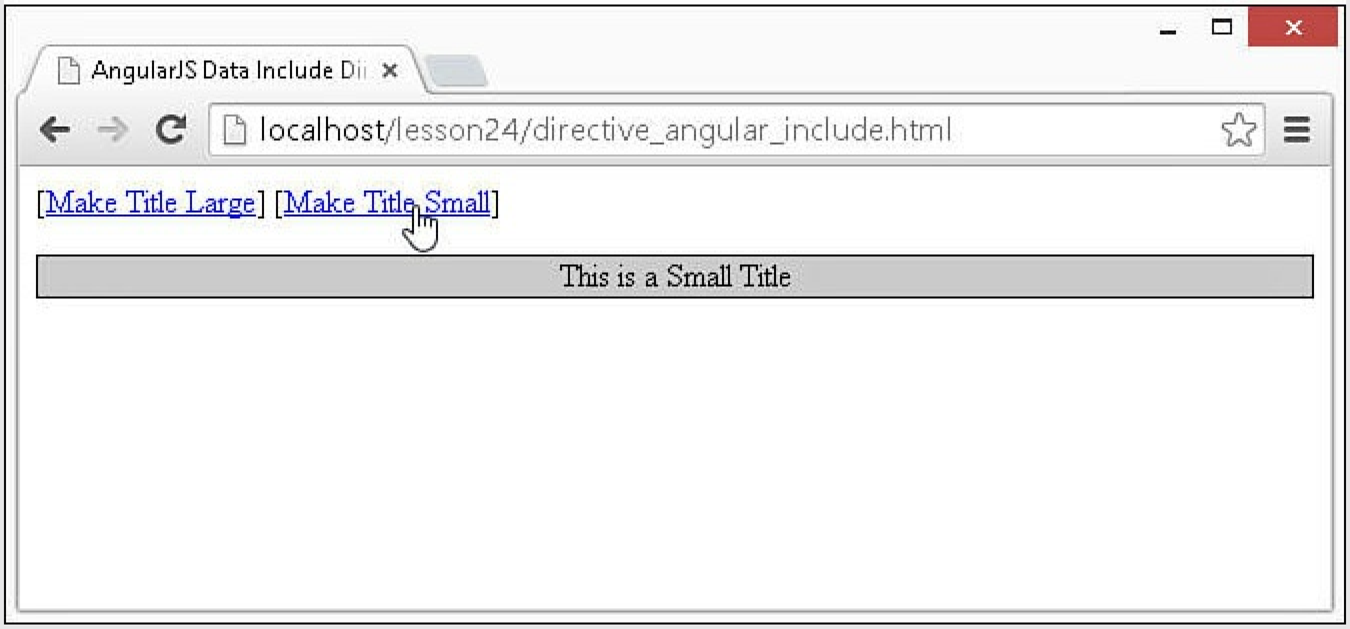

AngularJS Data Include Di
localhost/lesson24/directive_angular_include.html
[Make Title Large] [Make Title Small]
This is a Small Title

FIGURE 24.1 Using the `ng-include` directive to dynamically change the view using different HTML partial files.

LISTING 24.1 directive_angular_include.js Implementing a Controller to Store the HTML Filename for a Title Element in the Scope

[Click here to view code image]

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.titleBar = "small_title.html";
  });
```

LISTING 24.2 directive_angular_include.html An AngularJS Template That Uses the nd-include Directive to Change the Title Bar of the Page by Swapping Between Two HTML Files

[Click here to view code image]

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Data Include Directive</title>
  <style>
    .large{
      background-color: blue; color: white;
      text-align: center;
      font: bold 50px/80px verdana, serif;
      border: 6px black ridge; }
    .small{
      background-color: lightgrey;
      text-align: center;
      border: 1px black solid; }
    a{
     color: blue; text-decoration: underline;
     cursor: pointer; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    [<a ng-click="titleBar='large_title.html'">Make Title Large</a>]
    [<a ng-click="titleBar='small_title.html'">Make Title Small</a>]
    <div ng-include="titleBar"></div>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/directive_angular_include.js"></script>
</body>
</html>
```

LISTING 24.3 small_title.html A Partial HTML File That Contains the Small Version of the Title

```
<p class="small">
  This is a Small Title
</p>
```

LISTING 24.4 large_title.html A Partial HTML File That Contains the Large Version of the Title

```
<p class="large">
  This is a Large Title
</p>?
```

Directives That Extend Form Elements

AngularJS is heavily integrated with form elements to provide data binding and event binding for form elements in applications. To provide AngularJS functionality in the correct way, form elements are extended when compiled.

Table 24.2 lists the form elements that AngularJS extends.

Directive	Description
`a`	This directive modifies the default behavior to prevent the default action when the `href` attribute is empty. This enables you to create action links by using `ngClick` or other event directives. For example: `<a href="" ng-click="handleClick()">Click Me</a>`

`form/ngForm`	AngularJS allows forms to be nested for validation purposes so that a form is valid when all child forms are valid as well. However, browsers do not allow nesting of `<form>` elements; therefore, you should use `<ng-form>` instead. For example:
	`<ng-form name="myForm>`
	`  <input type="text" ng-model="myName" required>`
	`</ng-form>`

`input`	You can modify this directive to provide the following additional AngularJS attributes: ▶ `ngModel`: Binds the value of this input to a variable in the scope. ▶ `name`: Specifies the name of the form. ▶ `required`: When present, a value is required for this field. ▶ `ngRequired`: Sets the required attribute based on the evaluation of the `ngRequired` expression. ▶ `ngMinlength`: Sets the `minlength` validation error amount. ▶ `ngMaxlength`: Sets the `maxlength` validation error amount. ▶ `ngPattern`: Specifies a `regex` pattern to match the input value against for validation. ▶ `ngChange`: Specifies an expression to be executed when the input changes—for example, executing a function in the scope.

`input.checkbox`	This directive adds the following extra AngularJS attributes in addition to those already provided with `input`: ▶ `ngTrueValue`: Sets a value in the scope when the element is checked. ▶ `ngFalseValue`: Sets a value in the model when the element is not checked.
`input.email`	This directive is the same as `input`.
`input.number`	This directive adds the following extra AngularJS attributes in addition to those already provided to `input`: ▶ `min`: Sets the `min` validation error amount. ▶ `max`: Sets the `max` validation error amount.
`input.radio`	This directive adds the following extra AngularJS attribute in addition to those already provided to `input`: ▶ `value`: Sets a value in the scope when the element is selected.
`input.text`	This directive is the same as `input`.
`input.url`	This directive is the same as `input`.
`input.date`	This directive adds date validation and transformation for date input elements. The value in the scope model must also be a JavaScript `Date` object. This directive adds the following extra AngularJS attributes in addition to those already provided to `input`: ▶ `min:` Sets the `min` date validation error amount. ▶ `max:` Sets the `max` date validation error amount.

`input.dateTimeLocal`	This directive is the same as `input.date`, except that the format must be entered as a valid ISO-8601 local date-time format (yyyy-MM-ddTHH:mm). For example: `2014-11-28T12:37`
`input.month`	This directive is the same as `input.date`, except that the format must be entered as a valid ISO-8601 month format (yyyy-MM). For example: `2014-11`
`input.time`	This directive is the same as `input.date`, except that the format must be entered as a valid ISO-8601 time format (HH:mm). For example: `12:37`
`input.week`	This directive is the same as `input.date`, except that the format must be entered as a valid ISO-8601 week format (yyyy-W##). For example: `2014-W02`
`select`	This directive adds the additional `ngOptions` directive to the `<select>` element.

`ngOptions`	This directive enables you to add options based on an iterative expression. If the data source in the scope is an array, use the following expressions for `ngOptions` to set the `label`, `name`, and `value` attributes of each `<option>` element in the `<select>`:
	`label for value in array`
	`select as label for value in array`
	`label group by group for value in array`
	`select as label group by group for value in array track by trackexpr`
	
	If the source for `ngOptions` in the scope is a JavaScript object, use the following expression syntax:
	`label for (key , value) in object`
	`select as label for (key , value) in object`
	`label group by group for (key, value) in object`
	`select as label group by group for (key, value) in object`
	
	For example:
	`<select ng-model="color"`
	`        ng-options="c.name for c in colors">`
	`  <option value="">-- choose color --</option>`
	`</select>`

`textarea`	This directive is the same as `input`.

TABLE 24.2 Directives That Extend Form Elements to Support AngularJS Template Functionality

---

Try it Yourself: Using AngularJS Form Directives to Bind Form Elements to the Scope

Listings 24.5 and 24.6 implement some basic AngularJS form element integration with the scope. Listing 24.5 initializes the scope. Listing 24.6 implements several common form components, including a text box, a check box, radio buttons, and a `select` element to illustrate how they are defined in the template and interact with data in the scope.

Use the following steps to build the application using AngularJS form elements to bind form elements to the controller's scope:

1. Add the lesson24/directive_form.html and lesson24/js/directive_form.js files.

2. Add the code shown in Listing 24.5 and Listing 24.6 to the HTML and JavaScript files.

3. The code in Listing 24.5 implements a basic scope that contains an array of objects named `cameras` that will be used to dynamically populate a `<select>` element in the template. It also provides values named `cameraObj`, `cameraName`, `cbValue`, and `someText` that will be bound to various form elements in the template.

4. The code in Listing 24.6 implements several AngularJS form elements that interact with the scope to get and set values. The following line of code binds the value of a `text input` to the `someText` variable in the scope and then displays that value next to the `input` box in the web page:

Click here to view code image

```
<input type="text" ng-model="someText"> {{someText}}<hr>
```

5. The following lines of code bind the value of a `checkbox` element to the scope value of `cbValue` and then change the value of `cbValue` to different strings based on whether the check box is checked:

Click here to view code image

```
<input type="checkbox" ng-model="cbValue"
       ng-true-value="'Checked'" ng-false-value="'Not Checked'">
Checkbox: {{cbValue}}<hr>
```

6. The following lines of code define a radio button group that has a value bound to `cameraName` and then displays that value after the radio button group in the web page:

Click here to view code image

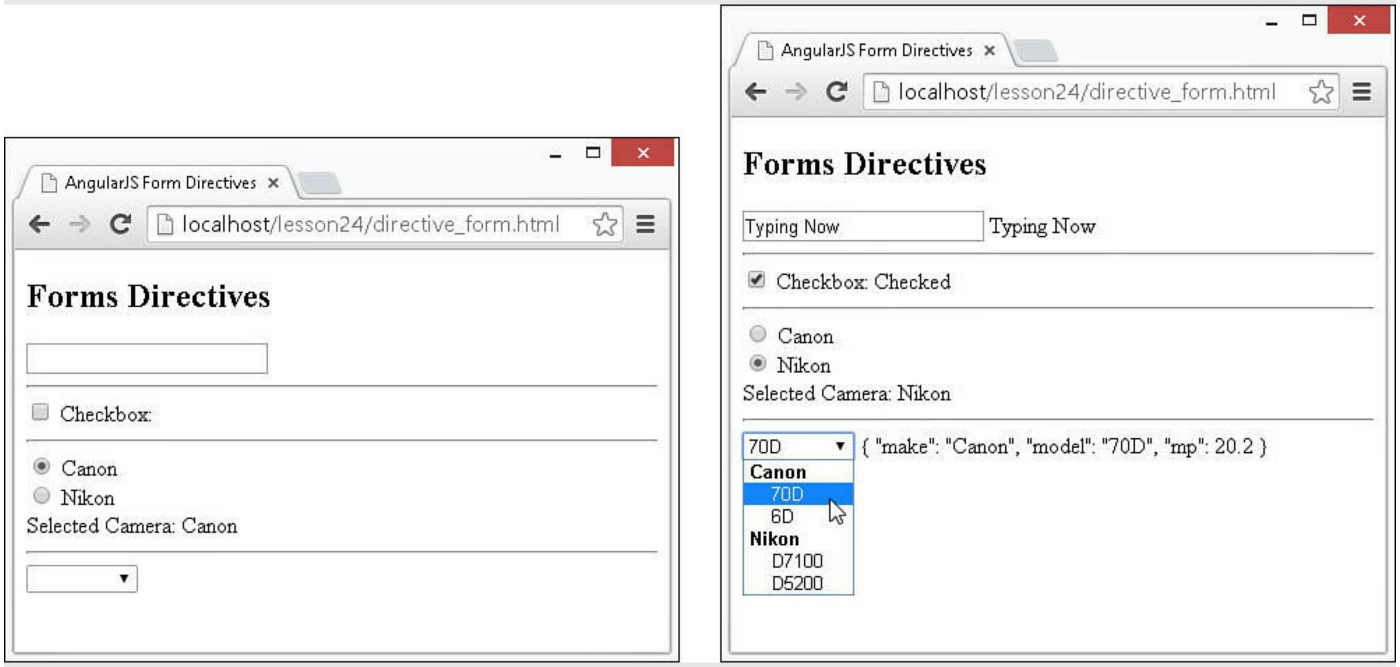
```
<input type="radio"
  ng-model="cameraName" value="Canon"> Canon<br/>
<input type="radio"
  ng-model="cameraName" value="Nikon"> Nikon<br/>
Selected Camera: {{cameraName}} <hr>
```

7. The following lines of code use the camera array that is defined in the scope to build option groups and options in a `<select>` element and then displays the object associated with the currently selected camera below:

Click here to view code image

```
<select ng-model="camera"
  ng-options="c.model group by c.make for c in cameras">
</select>
{{camera|json}}
```

8. Open directive_form.html in a browser and try changing the values of the form elements, as shown in Figure 24.2.


AngularJS Form Directives
localhost/lesson24/directive_form.html
AngularJS Form Directives
localhost/lesson24/directive_form.html

Forms Directives

Checkbox:

Canon
Nikon
Selected Camera: Canon

Forms Directives

Typing Now Typing Now

Checkbox: Checked

Canon
Nikon
Selected Camera: Nikon

70D { "make": "Canon", "model": "70D", "mp": 20.2 }
Canon
70D
6D
Nikon
D7100
D5200

FIGURE 24.2 Implementing form directive elements in AngularJS template views.

LISTING 24.5 directive_form.js Implementing a Controller for Form Directives

Click here to view code image

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.cameras = [
      {make:'Canon', model:'70D', mp:20.2},
      {make:'Canon', model:'6D', mp:20},
      {make:'Nikon', model:'D7100', mp:24.1},
      {make:'Nikon', model:'D5200', mp:24.1}];
    $scope.cameraObj=$scope.cameras[0];
    $scope.cameraName = 'Canon';
    $scope.cbValue = '';
    $scope.someText = '';
  });
```

LISTING 24.6 directive_form.html An AngularJS Template That Implements Several Form Element Directives

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Form Directives</title>
</head>
<body>
  <div ng-controller="myController">
    <h2>Forms Directives</h2>
    <input type="text" ng-model="someText"> {{someText}}<hr>
    <input type="checkbox" ng-model="cbValue"
           ng-true-value="'Checked'" ng-false-value="'Not Checked'">
    Checkbox: {{cbValue}}<hr>
    <input type="radio"
      ng-model="cameraName" value="Canon"> Canon<br/>
    <input type="radio"
      ng-model="cameraName" value="Nikon"> Nikon<br/>
    Selected Camera: {{cameraName}} <hr>
    <select ng-model="camera"
      ng-options="c.model group by c.make for c in cameras">
    </select>
    {{camera|json}}
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/directive_form.js"></script>
</body>
</html>
```

Directives That Bind the Model to Page Elements

AngularJS templates enable you to bind data in the scope directly to what is displayed in HTML elements. You can bind data to the view in several ways, including these:

- **Value:** You can directly represent the value of a form element in the scope. For example, a text input can be a `String` variable in the scope, but a check box would be represented by a `Boolean` value.

- **HTML:** You can represent the value of data in the scope in the HTML output of an element by using expressions such as this:

  ```
  <p>{{myTitle}}</p>
  ```

- **Attributes:** The value of HTML element attributes can reflect the data in the scope by using expressions in the definition such as this:

Click here to view code image

```
<a ng-href="/{{hash}}/index.html">{{hash}}</a>.
```

- **Visibility:** The visibility of an element can reflect the scope in the view. For example, when an expression based on the scope is `true`, the element is visible; otherwise, it is invisible.
- **Existence:** You can omit elements from the compiled DOM, based on values in the scope.

Table 24.3 lists the directives that bind the data in the scope directly to elements in the view.

Directive	Description
`ngBind`	This directive tells AngularJS to replace the `text` content of the HTML element with the value of a given expression and also to update the `text` content if the value in the scope changes. For example: `<span ng-bind="titleString"></span>`
`ngBindHtml`	This directive tells AngularJS to replace the `innerHTML` content of the HTML element with the value of a given expression and also to update the `innerHTML` content if the value in the scope changes. For example: `<div ng-bind-html="someHTML"></div>`

`ngBindTemplate`	This directive is similar to `ngBind`, except that the expression can contain multiple `{{}}` expression blocks. For example:
	`<span`
	`  ng-bind-template="{{aValue}} and {{anotherValue}}">`
	`</span>`

`ngClass`	This directive dynamically sets the CSS class of the element by data binding an expression that represents the classes to be added. When the value of the expression changes, the CSS classes of the element are automatically updated. For example: `<p ng-class="myPStyles"></p>`

`ngClassEven`	This directive is the same as `ngClass` except that it works with `ngRepeat` to apply the class changes only to even indexed elements in the set. For example:
	`<li ng-repeat="item in items">`
	`  <span ng-class-even="evenRowClass">{{item}}</span>`
	`</li>`

ngClassOdd	This directive is the same as ngClass except that it works with ngRepeat to apply the class changes only to odd indexed elements in the set. For example: `<li ng-repeat="item in items">` `<span ng-class-odd="oddRowClass">{{item}}</span>` `</li>`
ngDisabled	This directive disables a button element if the expression evaluates to true.
ngHide	This directive shows or hides the HTML element based on the expression provided, using the .ng-hide CSS class provided in AngularJS. If the expression evaluates to false in the scope, the element is displayed; otherwise, it is hidden. For example: `<div ng-hide="myValue"></div>`
ngShow	This directive is the same as ngHide except in reverse: If the expression evaluates to true in the scope, the element is displayed; otherwise, it is hidden. For example: `<div ng-show="myValue"></div>`
ngIf	This directive deletes or re-creates a portion of the DOM tree, based on the expression. This is different from show or hide because the HTML does not show up at all in the DOM. For example: `<div ng-if="present"> </div>`
ngModel	This directive binds the value of an `<input>`, `<select>`, or `<textarea>` element to a value in the scope model. When the user changes the value of the element, the value is automatically changed in the scope, and vice versa. For example: `<input type="text" ng-model="myString">`
ngRepeat	This directive enables you to add multiple HTML elements based on an array in the scope. This is extremely useful for lists, tables, and menus. ngRepeat uses the item in collections style of iteration syntax. A new scope is created for each HTML element created. During the looping to generate the HTML elements, the following variables are visible in the scope: ▶ $index: An iterator index based on 0 for the first element. ▶ $first: A Boolean that is true if this is the first element. ▶ $middle: A Boolean that is true if this is not the first or last element. ▶ $last: A Boolean that is true if this is the last element. ▶ $even: A Boolean that is true if the iterator is even. ▶ $odd: A Boolean that is true if the iterator is odd.

	For example, the following iterates and builds a series of `<li>` elements based on an array of users with a `firstname` property:
	`<li ng-repeat="user in users">`
	` {{$index}}: {{user.firstname}}</li>`

`ngInit`	This directive is used with `ngRepeat` to initialize a value during the iteration. For example:
	`<div ng-repeat="user in users" ng-init="offset=21">`
	` {{$index+offset}}: {{user.firstname}}</div>`

`ngStyle`	This directive enables you to set the style dynamically, based on an object in the scope where the property names and values match CSS attributes. For example: `<span ng-style="myStyle">Stylized Text</span>`

`ngSwitch`	This directive enables you to dynamically swap which DOM element to include in the compiled template, based on a scope expression. The following is an example of the syntax used for multiple elements:
	`<div ng-switch="myLocation">`
	`  <div ng-switch-when="home">Home Info</div>`
	`  <div ng-switch-when="work">Work Info</div>`
	`  <div ng-switch-default>Default Info</div>`
	`</div>`

`ngValue`	This directive binds the selected value of an `input[select]` or `input[radio]` to the expression specified in `ngModel`. For example:
	`<div ng-repeat="pizza in pizzas"`
	`  <input type="radio" name="pizza"`
	`    ng-model="myPizza" ng-value="pizza" id="{{pizza}}" >`
	`</div>`

TABLE 24.3 Directives That Bind Data in the Scope to the Value, Expressions, Visibility, and Existence of HTML Elements

Try it Yourself: Using AngularJS Directives to Dynamically Interact with Page Elements

In this example, you use some of the AngularJS directives to dynamically interact with page elements. Listings 24.7 and 24.8 provide some examples of basic AngularJS binding directives. Listing 24.7 initializes the scope values. Listing 24.8 provides the actual implementation of the binding directives in the template.

Use the following steps to build the application using AngularJS elements to interact with page elements:

1. Add the lesson24/directive_bind.html and lesson24/js/directive_bind.js files.
2. Add the code shown in Listing 24.7 and Listing 24.8 to the HTML and

JavaScript files.

3. The code in Listing 24.7 implements a basic scope with values that will be used by the AngularJS template. The `colors` array will be used to build a radio button group, the `myStyle` value will be used to set the CSS style for a page element, the `days` array will be used to create a list of days, and the `msg` value will be displayed on the page.
4. The code in Listing 24.8 provides the AngularJS template that uses the values in the scope to build and interact with page elements. The following lines of code use `ng-repeat` on the `colors` value in the scope to create a `radio` button `<input>`. Notice that the value of the radio button group is bound to the `myStyle['background-color']` property in the scope. This illustrates how to handle style names that do not allow the dot notation that's usually used (for example, `myStyle.color`). Also note that the value of the radio buttons is set using `ng-value to the color color` value that comes from the `ng-repeat` loop:

Click here to view code image

```
<label ng-repeat="color in colors">
  {{color}}
  <input type="radio" ng-model="myStyle['background-color']"
         ng-value="color" id="{{color}}" name="mColor">
</label>
```

5. The following line of code uses `ng-style` to set the CSS style of the `<span>` element to the value of `myStyle` in the scope. Keep in mind that the radio button group sets the `background-color` in `myStyle` when you select one of the radio buttons:

Click here to view code image

```
<span class="rect" ng-style="myStyle"></span><hr>
```

6. The following lines of code use `ng-repeat` to create a list of days. Notice that when you set the class name using `ng-class-even`, the class name `even` needs to be in single quotes because it is a string:

Click here to view code image

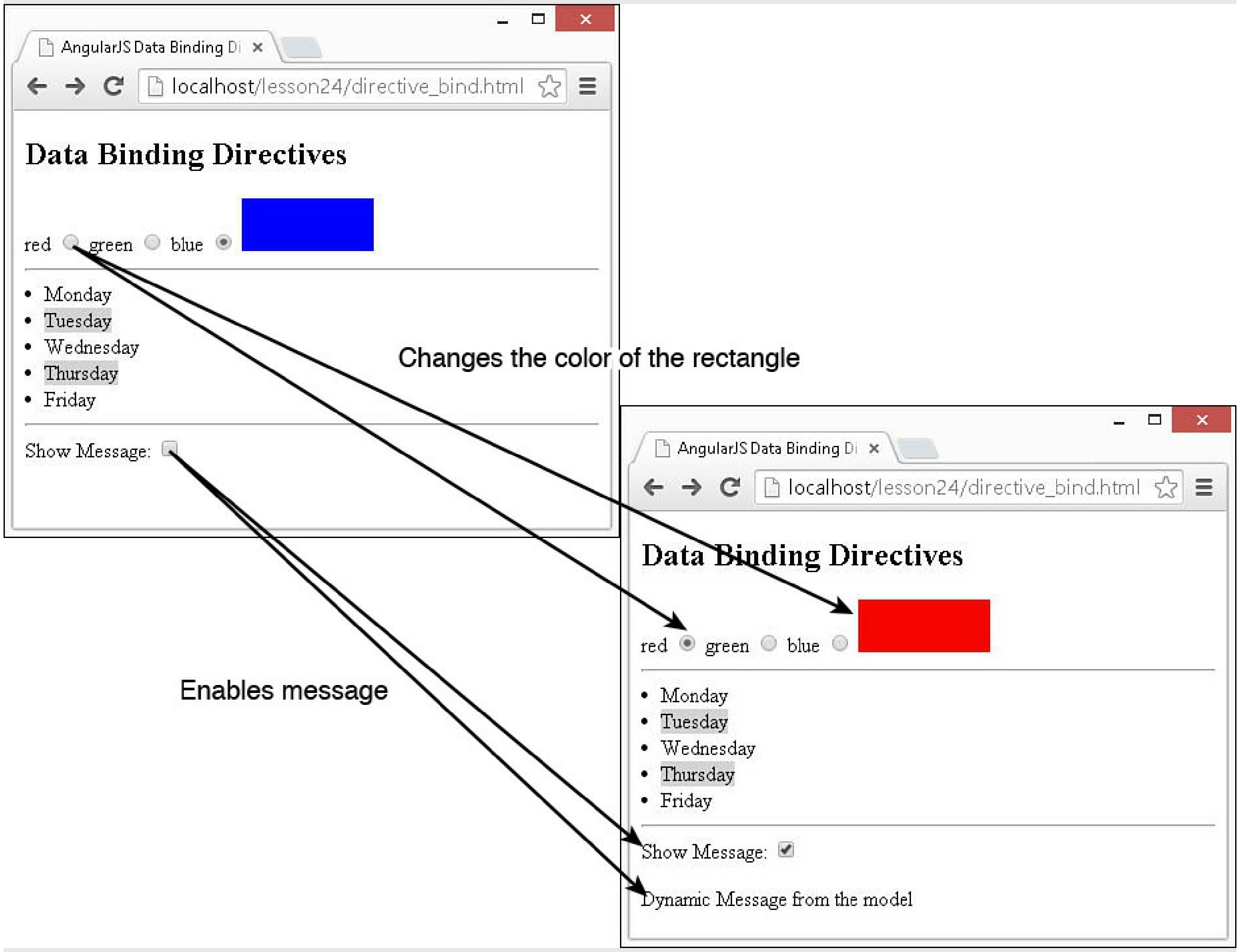
```
<li ng-repeat="day in days">
  <span ng-class-even="'even'">{{day}}</span>
</li><hr>
```

7. The following lines of code use `ng-model` to bind the value of the check box (true/false) to the `checked` value in the scope. The `ng-if` directive reads the value of `checked` from the scope and determines whether to bind the `msg` value in the scope to the paragraph element:

Click here to view code image

```
      Show Message: <input type="checkbox" ng-model="checked" />
      <p ng-if="checked" ng-bind="msg"> </p>
```

8. Open the directive_bind.html file in a browser and play around with changing the radio buttons to see the color of the rectangle change, view the style of even elements in the list, and turn the message on and off with the check box. Figure 24.3 shows the resulting web page.



AngularJS Data Binding Di
localhost/lesson24/directive_bind.html
Data Binding Directives
red
green
blue
Monday
Tuesday
Wednesday
Thursday
Friday
Show Message:
Changes the color of the rectangle
Enables message
Dynamic Message from the model

FIGURE 24.3 Implementing data binding directives in AngularJS template views.

LISTING 24.7 directive_bind.js Implementing a Controller with a Scope Model to Support Data Binding Directives

Click here to view code image

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.colors=['red','green','blue'];
    $scope.myStyle = { "background-color": 'blue' };
    $scope.days=['Monday', 'Tuesday', 'Wednesday',
                 'Thursday', 'Friday'];
    $scope.msg="Dynamic Message from the model";
  });
```

LISTING 24.8 directive_bind.html An AngularJS Template That Implements Several Data Binding Directives

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Data Binding Directives</title>
  <style>
    .even{background-color:lightgrey;}
    .rect{display:inline-block; height:40px; width:100px;}
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h2>Data Binding Directives</h2>
    <label ng-repeat="color in colors">
      {{color}}
      <input type="radio" ng-model="myStyle['background-color']"
             ng-value="color" id="{{color}}" name="mColor">
    </label>
    <span class="rect" ng-style="myStyle"></span><hr>
    <li ng-repeat="day in days">
      <span ng-class-even="'even'">{{day}}</span>
    </li><hr>
    Show Message: <input type="checkbox" ng-model="checked" />
    <p ng-if="checked" ng-bind="msg"> </p>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/directive_bind.js"></script>
</body>
</html>
```

Directives That Bind Page Events to Controllers

AngularJS templates enable you to bind browser events to controller code. This means you can handle user input from the scope's perspective. You can then implement handlers for browser events directly to the appropriate scope. The `event` directive works very much like the normal browser event handlers, except that they are directly linked to the scope context.

Table 24.4 lists the directives that bind page and device events to the AngularJS model. Each of these directives enables you to specify an expression, which is typically a function defined in the scope, as discussed in Lesson 23, “Using AngularJS Templates to Create Views.” For example, the following is a function named `setTitle` in the scope:

Click here to view code image

```
$scope.setTitle = function(title){
  $scope.title = title;
};
```

Directive	Description
`ngBlur`	Evaluates an expression when the blur event is triggered.
`ngChange`	Evaluates an expression when the value of a form element is changed.
`ngChecked`	Evaluates an expression when a check box or radio element is checked.
`ngClick`	Evaluates an expression when the mouse is clicked.
`ngCopy`	Evaluates an expression when the copy event is triggered.
`ngCut`	Evaluates an expression when the cut event is triggered.
`ngDblclick`	Evaluates an expression when the mouse is double-clicked.
`ngFocus`	Evaluates an expression when the focus event is triggered by the element coming into focus.
`ngKeydown`	Evaluates an expression when a keyboard key is pressed.
`ngKeypress`	Evaluates an expression when a keyboard key is pressed and released.
`ngKeyup`	Evaluates an expression when a keyboard key is released.
`ngMousedown`	Evaluates an expression when a mouse key is pressed.
`ngMouseenter`	Evaluates an expression when the mouse enters the element.
`ngMouseleave`	Evaluates an expression when the mouse leaves the element.
`ngMousemove`	Evaluates an expression when the mouse cursor moves.
`ngMouseover`	Evaluates an expression when the mouse hovers over an element.
`ngMouseup`	Evaluates an expression when the mouse button is released.
`ngPaste`	Evaluates an expression when the paste event is triggered.
`ngSubmit`	Prevents the default form submit action, which sends a request to the server and instead evaluates the specified expression.
`ngSwipeLeft`	Evaluates an expression when the swipe left event is triggered.
`ngSwipeRight`	Evaluates an expression when the swipe right event is triggered.

TABLE 24.4 Directives That Bind Page/Device Events to AngularJS Model

Functionality

You can bind the `setTitle()` function in the scope directly to an input button in the view by using the following `ng-click` directive:

Click here to view code image

```
<input type="button" ng-click="setTitle('New Title')">
```

You can pass the JavaScript `Event` object into the event expressions by using the `$event` keyword. This enables you to access information about the event as well as stop propagation and everything else you normally can do with a JavaScript `Event` object. For example, the following `ng-click` directive passes the mouse click event to the `myClick()` handler function:

Click here to view code image

```
<input type="button" ng-click="myClick($event)">
```

The following sections provide some examples of using the AngularJS event directives to interact with events coming from the browser.

Try it Yourself: Using the Focus and Blur Events

The AngularJS `ngBlur` and `ngFocus` directives are useful to track when form elements go in and out of focus. For example, you might want to execute some code in the controller when a particular input element goes in and out of focus—for instance, to manipulate the input before updating the model. The code in Listings 24.9 and 24.10 illustrate an example of using the `ngBlur` and `ngFocus` directives to set values in the scope based on entering and leaving text inputs.

Use the following steps to build the application that utilizes `ngBlur` and `ngFocus` on elements to interact with page elements:

1. Add the lesson24/directive_focus_events.html and lesson24/js/directive_focus_events.js files.
2. Add the code shown in Listing 24.9 and Listing 24.10 to the HTML and JavaScript files.
3. The code in Listing 24.9 implements a basic controller. The following lines define an object named `inputData` in the scope, which will contain information about the value and focus state of input elements on the page:

Click here to view code image

```
    $scope.inputData = { input1: {value: "", state: ""},
                         input2: {value: "", state: ""} };
```

4. The following lines define the `focusGained()` function that is called when an input comes into focus and uses the `input` parameter to set the value for that input in `inputData` to an empty string:

Click here to view code image

```
$scope.focusGained = function(input){
  $scope.inputData[input]['value'] = '';
  $scope.inputData[input]['state'] = 'Focus Gained';
};
```

5. The following lines define the `focusLost()` function, which will accept the `event` and the `input` as inputs and will use the `event` object to get the value of the `target` element and update the corresponding property in `inputData`:

Click here to view code image

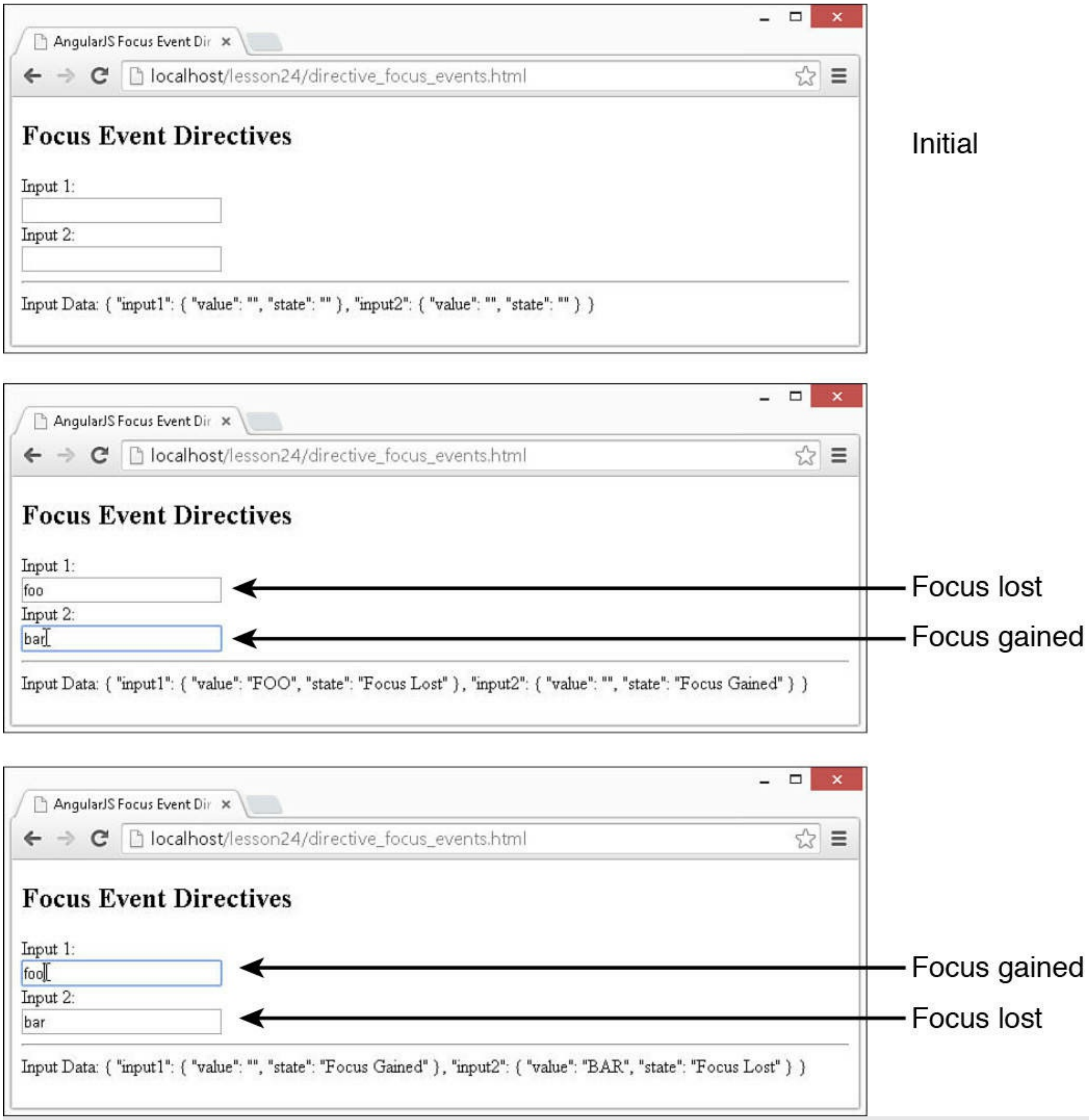
```
$scope.focusLost = function(event, input){
  var element = angular.element(event.target);
  var value = element.val();
  $scope.inputData[input]['value'] = value.toUpperCase();
  $scope.inputData[input]['state'] = "Focus Lost";
};
```

6. The following code in Listing 24.10 implements the two `<input>` elements and assigns the `focusGained()` and `focusLost()` handlers to the `ng-focus` and `ng-blur` attributes:

Click here to view code image

```
<input type="text"
  ng-blur="focusLost($event, 'input1')"
  ng-focus="focusGained('input1')"><br>
. . .
<input type="text"
  ng-blur="focusLost($event, 'input2')"
  ng-focus="focusGained('input2')"><hr>
```

7. Open the directive_focus_events.html file in a browser. Figure 24.4 shows the basic example in action. Note that when you click an input element, the value stored in `inputData` is set to an empty string, and when you leave the input element, the value is updated.



AngularJS Focus Event Dir

localhost/lesson24/directive_focus_events.html

Focus Event Directives

Input 1:

Input 2:

Input Data: { "input1": { "value": "", "state": "" }, "input2": { "value": "", "state": "" } }

AngularJS Focus Event Dir

localhost/lesson24/directive_focus_events.html

Focus Event Directives

Input 1:

foo

Input 2:

bar

Input Data: { "input1": { "value": "FOO", "state": "Focus Lost" }, "input2": { "value": "", "state": "Focus Gained" } }

AngularJS Focus Event Dir

localhost/lesson24/directive_focus_events.html

Focus Event Directives

Input 1:

foo

Input 2:

bar

Input Data: { "input1": { "value": "", "state": "Focus Gained" }, "input2": { "value": "BAR", "state": "Focus Lost" } }

FIGURE 24.4 Implementing focus event directives in AngularJS template views.

LISTING 24.9 directive_focus_events.js Implementing a Controller with Scope Data and Event Handlers to Support Blur and Focus Events from the View

Click here to view code image

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.inputData = { input1: {value: "", state: ""},
                         input2: {value: "", state: ""} };
    $scope.focusGained = function(input){
      $scope.inputData[input]['value'] = '';
      $scope.inputData[input]['state'] = 'Focus Gained';
    };
    $scope.focusLost = function(event, input){
      var element = angular.element(event.target);
      var value = element.val();
      $scope.inputData[input]['value'] = value.toUpperCase();
      $scope.inputData[input]['state'] = "Focus Lost";
    };
  });
```

LISTING 24.10 directive_focus_events.html An AngularJS Template That Implements the `ngFocus` and `ngBlur` Directives

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Focus Event Directives</title>
</head>
<body>
  <div ng-controller="myController">
    <h2>Focus Event Directives</h2>
    Input 1:<br>
    <input type="text"
      ng-blur="focusLost($event, 'input1')"
      ng-focus="focusGained('input1')"><br>
    Input 2:<br>
    <input type="text"
      ng-blur="focusLost($event, 'input2')"
      ng-focus="focusGained('input2')"><hr>
    Input Data: {{inputData|json}}<br/>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/directive_focus_events.js"></script>
</body>
</html>
```

Try it Yourself: Handling Keyboard Events on AngularJS Elements

The most common keyboard event directives that you will use are the `ngKeydown` and `ngKeyup` events that are triggered when a keyboard key is pressed and released, respectively. Keyboard events are useful for interacting more closely with users as they type on the keyboard. Probably the most common keyboard interaction is to apply some action when a user presses the Enter key on the keyboard. The code in Listings 24.11 and 24.12 illustrate the usage of the `ngKeydown` and `ngKeyup` directives.

Use the following steps to build the application that utilizes `ngKeydown` and `ngKeyup` directives on elements to interact with page elements:

1. Add the lesson24/directive_keyboard_events.html and lesson24/js/directive_keyboard_events.js files.

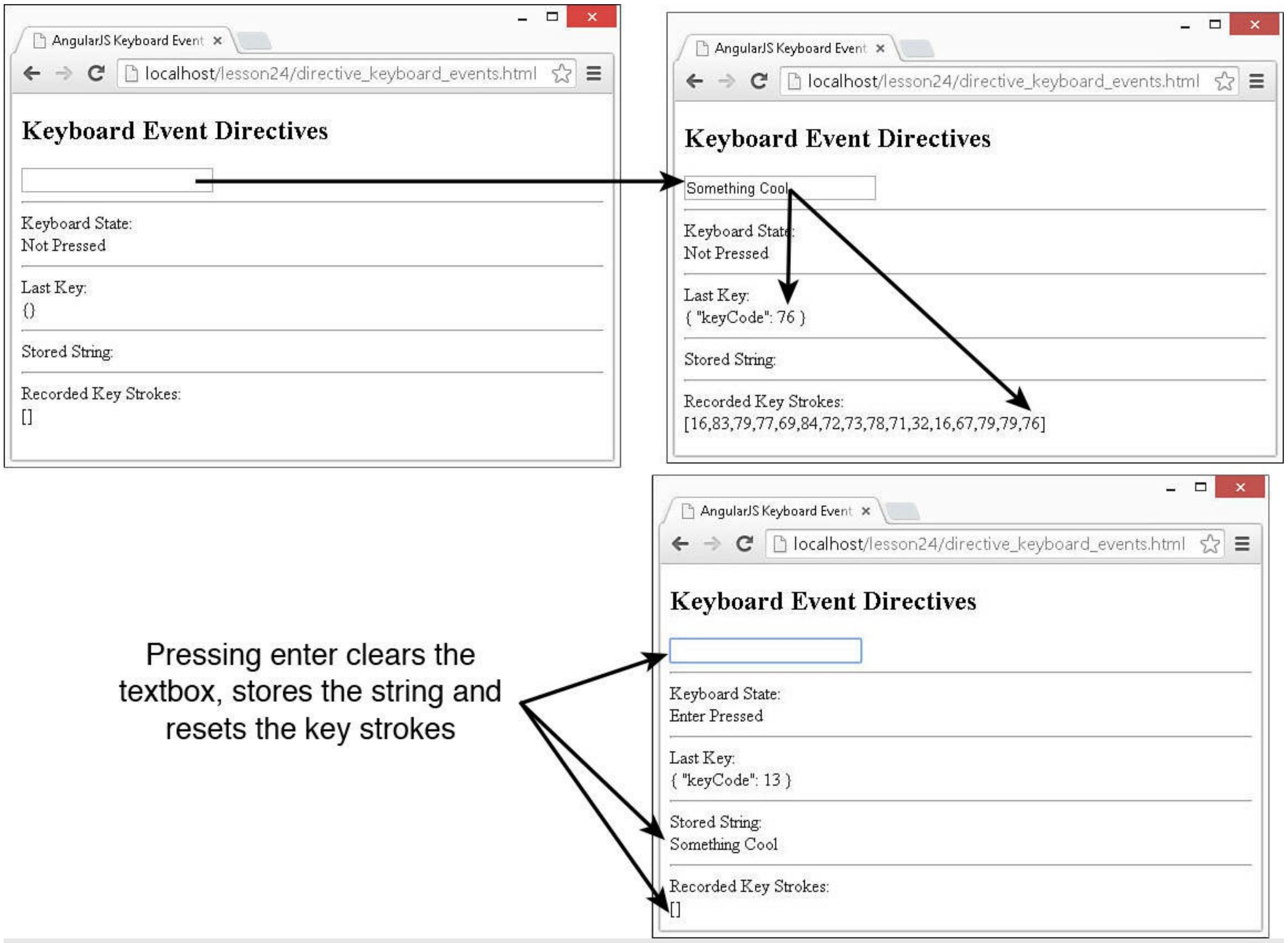
2. Add the code shown in Listing 24.11 and Listing 24.12 to the HTML and JavaScript files.

3. The code in Listing 24.11 implements a controller that provides the model and keyboard handler functions for the key-down and key-up events. The `storedString` variable is used to store the value of a text input whenever the user presses Enter while typing in the input. The `keyInfo` variable stores the `keyCode` for the last key pressed, and the `keyStrokes` array records the previous `keyCodes` for keys pressed.

4. The following lines of code in Listing 24.11 implement the `keyPressed()` function. This function will be called when the key is pressed in the input field. Notice that we check to see whether the `keyCode` is 13, meaning Enter was pressed, and if so, we record the `storedString` and reset the other variables. If the key was not the Enter key, we store the `keyCode`, add it to the `keyStrokes` array, and set the `keyState`.

5. The following lines of code in Listing 24.12 assign the `ng-keydown` and `ng-keyup` directives to an `<input>` element. When `ng-keydown` is triggered, the `keyState` variable in the scope is updated, and when `ng-keyup` is triggered, the `keyPressed()` handler is called.

6. Open the directive_keyboard_events.html file in a browser and try typing into the text box. Figure 24.5 shows the AngularJS web page in action. Note that as you type each character in the text input, the values of `keyPressed` in the model is updated, and when Enter is pressed, the word is stored and the keystrokes are reset.


AngularJS Keyboard Event
localhost/lesson24/directive_keyboard_events.html
Pressing enter clears the textbox, stores the string and resets the key strokes

Keyboard Event Directives

Keyboard State:
Not Pressed

Last Key:
{}

Stored String:

Recorded Key Strokes:
[]

Keyboard Event Directives

Something Cool

Keyboard State:
Not Pressed

Last Key:
{ "keyCode": 76 }

Stored String:

Recorded Key Strokes:
[16,83,79,77,69,84,72,73,78,71,32,16,67,79,79,76]

Keyboard Event Directives

Keyboard State:
Enter Pressed

Last Key:
{ "keyCode": 13 }

Stored String:
Something Cool

Recorded Key Strokes:
[]

FIGURE 24.5 Implementing keyboard event directives in AngularJS template views.

LISTING 24.11 directive_keyboard_events.js Implementing a Controller with Scope Data and Event Handlers to Support Key-Down and Key-Up Events from the View

Click here to view code image

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.storedString = '';
    $scope.keyInfo = {};
    $scope.keyStrokes = [];
    $scope.keyState = 'Not Pressed';
    $scope.keyPressed = function(event){
      if (event.keyCode == 13){
        var element = angular.element(event.target);
        $scope.storedString = element.val();
        element.val('');
        $scope.keyInfo.keyCode = event.keyCode;
        $scope.keyStrokes = [];
        $scope.keyState = 'Enter Pressed';
      } else {
        $scope.keyInfo.keyCode = event.keyCode;
        $scope.keyStrokes.push(event.keyCode);
        $scope.keyState = 'Not Pressed';
      }
    };
  });
```

LISTING 24.12 directive_keyboard_events.html An AngularJS Template That Implements the `ngKeydown` and `ngKeyup` Directives

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Keyboard Event Directives</title>
</head>
<body>
  <div ng-controller="myController">
    <h2>Keyboard Event Directives</h2>
    <input type="text"
        ng-keydown="keyState='Pressed'"
        ng-keyup="keyPressed($event)"><hr>
    Keyboard State:<br>
      {{keyState}}<hr>
    Last Key:<br>
      {{keyInfo|json}}<hr>
    Stored String:<br>
      {{storedString}}<hr>
    Recorded Key Strokes:<br>
      {{keyStrokes}}
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_keyboard_events.js"></script>
</body>
</html>
```

Try it Yourself: Handling Mouse Events in the AngularJS Elements

AngularJS provides several mouse event directives that enable you to easily enhance your AngularJS applications with mouse interactivity. The most common mouse event directive that you will use is `ngClick` when the mouse is clicked. However, you also have several other mouse events that can come in handy to create richly interactive components. The code in Listings 24.13 and 24.14 illustrate the usage of the `ngClick`, `ngMouseenter`, `ngMouseleave`, ngMousedown, ngMouseup, and ngMousemove directives.

Use the following steps to build the application that utilizes ngKeydown and ngKeyup directives on elements to interact with page elements:

1. Add the lesson24/directive_mouse_events.html and lesson24/js/directive_mouse_events.js files.
2. Add the code shown in Listing 24.13 and Listing 24.14 to the HTML and JavaScript files.
3. The code in Listing 24.13 implements a controller that stores the current mouse position info in mouseInfo and last click position info in lastClickinfo.
4. The following lines of code in Listing 24.13 define the mouseClick() event handler. Notice that the event information is passed into the handler and allows us to set the clientX, clientY, screenX, and screenY information in lastClickInfo:

Click here to view code image

```
$scope.mouseClick = function(event){
  $scope.lastClickInfo.clientX = event.clientX;
  $scope.lastClickInfo.clientY = event.clientY;
  $scope.lastClickInfo.screenX = event.screenX;
  $scope.lastClickInfo.screenY = event.screenY;
};
```

5. The following lines of code in Listing 24.13 define the mouseMove() event handler. Notice that the event information is passed into the handler and allows us to set the clientX, clientY, screenX, and screenY information in mouseInfo:

Click here to view code image

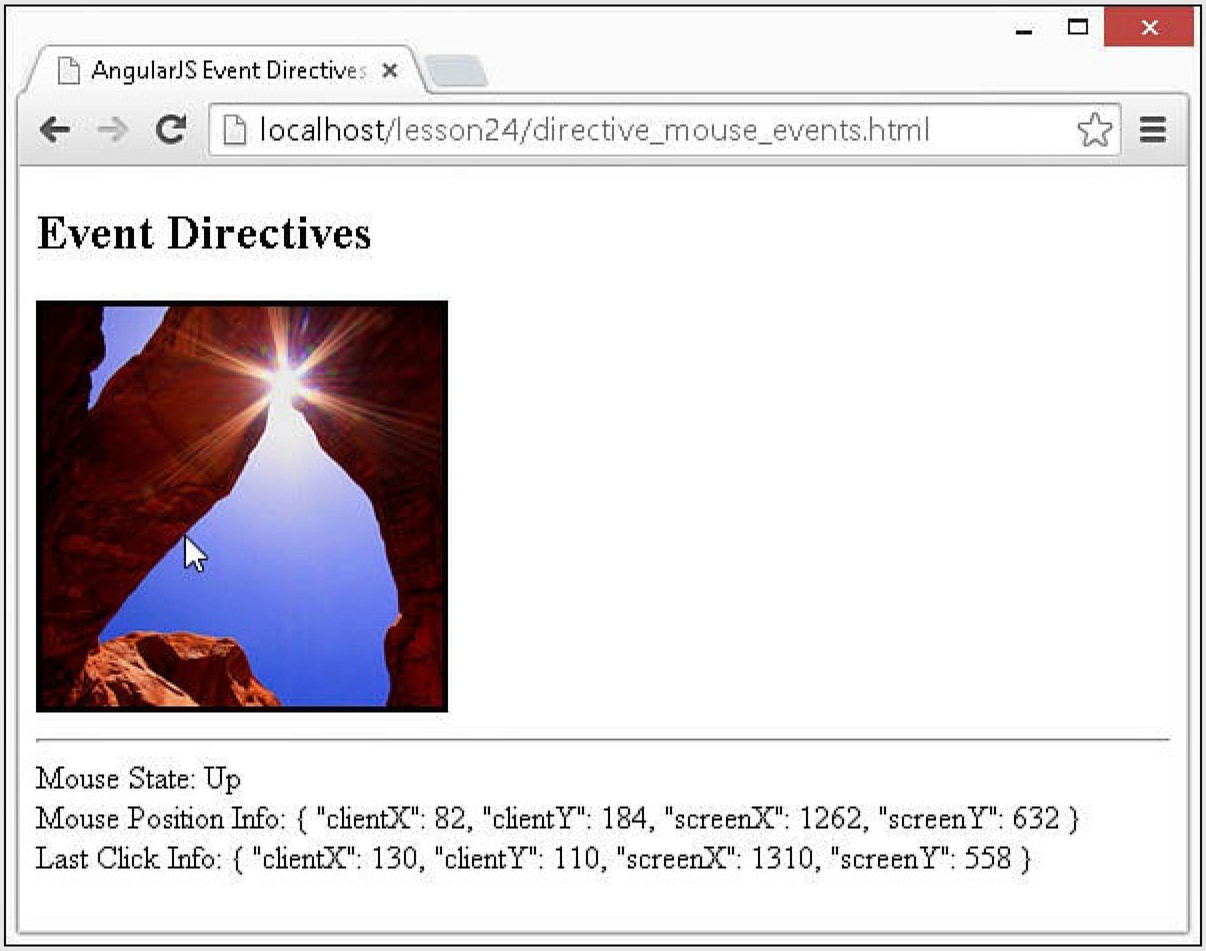
```
$scope.mouseMove = function(event){
  $scope.mouseInfo.clientX = event.clientX;
  $scope.mouseInfo.clientY = event.clientY;
  $scope.mouseInfo.screenX = event.screenX;
  $scope.mouseInfo.screenY = event.screenY;
};
```

6. The following lines of code in Listing 24.14 apply the ng-mouseenter, ng-mouseleave, ng-mouseclick, ng-mousedown, and ng-mouseup directives to an <img> element and set the value of mouseState in the scope when the event is triggered. They also bind the ng-click and ng-mousemove events to mouseClick($event) and mouseMove($event) in the scope. Notice that $event is used to pass the browser's mouse event to the handlers:

Click here to view code image

```
    <img
        src="/images/img3.jpg"
        ng-mouseenter="mouseState='Entered'"
        ng-mouseleave="mouseState='Left'"
        ng-mouseclick="mouseState='Clicked'"
        ng-mousedown="mouseState='Down'"
        ng-mouseup="mouseState='Up'"
        ng-click="mouseClick($event)"
        ng-mousemove="mouseMove($event)"></img><hr>
```

7. Open the directive_mouse_events.html file in a browser and try moving over and clicking the image. Figure 24.6 shows the AngularJS application working. Notice that the mouse state changes as you enter, leave, and click the image. Also notice that the position information is updated as you move over and click the image.


AngularJS Event Directives
localhost/lesson24/directive_mouse_events.html

Event Directives

Mouse State: Up
Mouse Position Info: { "clientX": 82, "clientY": 184, "screenX": 1262, "screenY": 632 }
Last Click Info: { "clientX": 130, "clientY": 110, "screenX": 1310, "screenY": 558 }

FIGURE 24.6 Implementing mouse click and movement event directives in AngularJS template views.

LISTING 24.13 directive_mouse_events.js Implementing a Controller with Scope Data and Event Handlers to Support Mouse Click and Movement Events from the View

Click here to view code image

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.mouseInfo = {};
    $scope.lastClickInfo = {};
    $scope.mouseClick = function(event){
      $scope.lastClickInfo.clientX = event.clientX;
      $scope.lastClickInfo.clientY = event.clientY;
      $scope.lastClickInfo.screenX = event.screenX;
      $scope.lastClickInfo.screenY = event.screenY;
    };
    $scope.mouseMove = function(event){
      $scope.mouseInfo.clientX = event.clientX;
      $scope.mouseInfo.clientY = event.clientY;
      $scope.mouseInfo.screenX = event.screenX;
      $scope.mouseInfo.screenY = event.screenY;
    };
  });
```

LISTING 24.14 directive_mouse_events.html An AngularJS Template That Implements the `ngClick` and Other Mouse Click and Move Event Directives

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Event Directives</title>
  <style>
    img {
      border: 3px ridge black;
      height: 200px; width: 200px;
      display: inline-block;
    }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h2>Event Directives</h2>
    <img
        src="/images/img3.jpg"
        ng-mouseenter="mouseState='Entered'"
        ng-mouseleave="mouseState='Left'"
        ng-mouseclick="mouseState='Clicked'"
        ng-mousedown="mouseState='Down'"
        ng-mouseup="mouseState='Up'"
        ng-click="mouseClick($event)"
        ng-mousemove="mouseMove($event)"></img><hr>
    Mouse State: {{mouseState}}<br/>
    Mouse Position Info: {{mouseInfo|json}}<br/>
    Last Click Info: {{lastClickInfo|json}}<br/>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/directive_mouse_events.js"></script>
</body>
</html>
```

Summary

AngularJS directives extend the behavior of HTML. You can apply directives to AngularJS templates as HTML elements, attributes, and classes. You define the functionality of directives by using JavaScript code. AngularJS provides several built-in directives that interact with form elements, bind data in the scope to the view, and interact with browser events. For example, `ngModel` binds the value of a form element directly to the scope. When the scope value changes, so does the value displayed by the element, and vice versa.

Q&A

Q. Is it possible to override the behavior of built-in AngularJS directives?

A. Not directly; however, it is possible to create your own directive with the same name, and it will also be executed along with the built-in directive. Typically, it is better to create your own directives with a different name.

Q. Why do the AngularJS directives use the ng prefix?

A. Because it is short and sounds like “angular.”

Workshop

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

Quiz

1. What does the AngularJS directive `ng-required` do?
2. What does the AngularJS directive `ng-repeat` do?
3. Which AngularJS directive would you use to bind an element to the mouse click event?
4. Which AngularJS directive would you use to bind a form element to the focus

gained event?

Quiz Answers

1. It creates a Boolean value for `<input>` elements in a form.
2. Allows you to dynamically add multiple HTML elements based on an array of JavaScript objects in the scope.
3. `ng-click`
4. `ng-focus`

Exercises

1. Modify the code in Listing 24.8 so that the odd weekdays are highlighted instead of the even.
2. Modify the code in Listings 24.9 and 24.10 so that the value for the input in the InputData object is lowercase when the input is not in focus but in uppercase when it is.

Lesson 25. Creating Your Own Custom Directives to Extend HTML

What You'll Learn in This Lesson:

- How to build a custom directive
- Ways to configure the scopes in the custom directives
- How to modify the DOM within a custom directive
- Using template partials to inject HTML elements into custom directives
- How to implement nested custom directives that interact with each other

As with many other features of AngularJS, you can extend directive functionality by creating your own custom directives. Custom directives enable you to extend the functionality of HTML by implementing the behavior of elements yourself. If you have code that needs to manipulate the DOM, you should make this happen by using a custom directive. As with the built-in directive, custom directives provide the capability to interact with form elements, bind data in the scope to the view, and interact with browser events.

This lesson discusses the design and implementation of custom directives. You will also see a couple of basic examples of custom directives that extend the capability of HTML.

Understanding Custom Directive Definitions

You implement custom directives by calling the `directive()` method on a `Module` object. The `directive()` method accepts the name of a directive as the first parameter and a provider function that returns an object containing the necessary instructions to build the directive object. For example, the following is a basic definition for a directive:

Click here to view code image

```
angular.module('myApp', []).
  directive('myDirective', function() {
    return {
      template: 'Name: {{name}} Score: {{score}}'
    };
  });
```

Table 25.1 provides a list of the properties you can apply to the object returned by the directive definition as `template` is returned in the preceding code.

Property	Description
`template`	Enables you to define the AngularJS template text that is inserted into the directive's element.
`templateUrl`	Same as `template` except that you specify a URL at the server, and the partial template is downloaded and inserted into the directive's element.
`restrict`	Enables you to specify whether the directive applies to an HTML element, an attribute, or both.
`type`	This is set to a string that represents the document type used by the markup. This is useful for templates that have a non-HTML root node; for example, SVG or MathML. The default value is `html`. ▶ **`html`**: All root template nodes are HTML and don't need to be wrapped. Root nodes can also be top-level elements such as `<svg>` or `<math>`. ▶ **`svg`**: The template contains only SVG content and must be wrapped in an `<svg>` node before processing. ▶ **`math`**: The template contains only MathML content and must be wrapped in a `<math>` node before processing.
`multiElement`	Tells the compiler to collect DOM nodes between nodes with the attributes `directive-name-start` and `directive-name-end`, and group them together as the directive elements. You should use this option only on directives that are not strictly behavioral, such as `ngClick`, and that do not manipulate or replace child nodes, such as `ngInclude`.
`priority`	Specifies a priority number for this directive. Directives are compiled in order based on their priority, starting with the directive with the highest-priority value. Prelink functions are executed in the same order starting at highest priority, but postlink functions are executed in reverse, starting with the lowest-priority directive.

`terminal`	This is a Boolean that tells the compiler to stop and not compile directives that have a lower-priority number. Any directives that have a lower-priority number than the one with terminal set to `true` will not be compiled.
`transclude`	Enables you to specify whether the directive has access to scopes outside the internal scope. This enables you to wrap the contents of an element into a new element generated when the directive is compiled and linked.
`scope`	Enables you to define the scope for the directive. The scope can share the parent scope, inherit from it, or have its own isolate scope.
`compile`	Enables you to specify a compile function that has access to the DOM element attributes. The compile function enables you to define prelink and postlink functions that can interact with and manipulate the AngularJS template DOM.
`link`	Similar to the compile function, but also provides access to the scope. The link function will be executed only if the compile function is not present. From the linking function, you can register DOM event listeners as well as manipulating the DOM. The link function is executed after the template has been cloned. Typically, you should put most of the template logic inside of the linking function.
`controller`	Enables you to define a controller within the directive to manage the directive scope and view.
`controllerAs`	Specifies an alias for the controller so that it can be referenced at the directive template. The directive needs to define a scope for this configuration to be used. This option is useful in the case in which the directive is used as component.
`bindToController`	When an isolate scope is used for a component and `controllerAs` is used, `bindToController` allows a component to have its properties bound to the controller, rather than to scope. When the controller is instantiated, the initial values of the isolate scope bindings are already available.
`require`	Enables you to specify other directives that are required to implement this directive. Providers for those directives must be available for an instance of this directive to be created.
`template`	Enables you to define the AngularJS template text that is inserted into the directive's element.

TABLE 25.1 Directive Definition Properties That Define AngularJS Directive Functionality

The following sections discuss the directive options in more detail.

Defining the Directive View Template

You can include AngularJS template code to build view components that will be displayed in the HTML element that contains the directive. You can add template code directly by using the `template` property, as in this example:

Click here to view code image

```
directive('myDirective', function() {
  return {
    template: 'Name: {{name}} Score: {{score}}'
  };
});
```

You can specify a root element in the custom template—but only one element. This element acts as the root element for any child element defined in the AngularJS template to be placed inside. Also, if you are using the `transclude` flag, the element should include `ngTransclude`. For example:

Click here to view code image

```
directive('myDirective', function() {
  return {
    transclude: true,
    template: '<div ng-transclude></div>'
  };
});
```

You can also use the `templateUrl` property to specify a URL of an AngularJS template located on the web server, as in this example:

Click here to view code image

```
directive('myDirective', function() {
  return {
    templateUrl: '/myDirective.html'
  };
});
```

The template URL can contain any standard AngularJS template code. You can therefore make your directives as simple or as complex as you need them to be.

Restricting Directive Behavior

You can apply a directive as an HTML element, an attribute, or both. The `restrict` property enables you to limit how your custom directive can be applied. The `restrict` property can be set to the following:

- **A:** Applied as an attribute name. For example:

```
<my-directive></my-directive>
```

- **E:** Applied as an element name. For example:

Click here to view code image

```
<div my-directive="expression"></div>
```

- **C:** Applied as a class. For example:

Click here to view code image

```
<div class="my-directive: expression;"></div>
```

- **M:** Applied as a comment. For example:

Click here to view code image

```
<!-- directive: my-directive expression -->
```

- **AEC:** Applied as an attribute, an element, or a class name. You can also use other combinations, such as `AE` or `AC`.

For example, you can apply the following directive as an attribute or an element:

Click here to view code image

```
directive('myDirective', function() {
  return {
    restrict: 'AE',
    templateUrl: '/myDirective.html'
  };
});
```

The following shows how to implement the directive as both an element and an attribute. Notice that the camelCase name is replaced by one with hyphens:

```
<my-directive></my-directive>
<div my-directive></div>
```

Adding a Controller to a Directive

You can add a custom controller to a directive by using the `controller` property of the directive definition. This enables you to provide controller support for the directive template. For example, the following code adds a simple controller that sets up a scope value and function:

Click here to view code image

```
directive('myDirective', function() {
  return {
    scope: {title: '='},
    controller: function ($scope){
      $scope.title = "new";
      $scope.myFunction = function(){
      });
    }
    };
  });
```

You can also use the `require` option to ensure that a controller is available to the directive. The `require` option uses the `require:'^controller'` syntax to instruct the injector service to look in parent contexts until it finds the controller. The following is an example of requiring the `myController` controller in a directive:

Click here to view code image

```
  directive('myDirective', function() {
    return {
      require: '^myController'
    };
  });
```

When you add the `require` option, the specified controller is passed as the fourth parameter of the `link()` function. For example:

Click here to view code image

```
  directive('myDirective', function() {
    return {
      require: '^myController',
      link: function(scope, element, attrs, injectedMyController){
            }
    };
  });
```

The `link` function will be executed after the directive is compiled and the DOM is build. You can then use the `link` function to manipulate elements of the DOM within the directive.

You can also require multiple controllers using the `require` option, in which case, an array of controllers is passed to the `link()` function. For example:

Click here to view code image

```
   directive('myDirective', function() {
    return {
      require: ['^myControllerA', '^myControllerB'],
      link: function(scope, element, attrs, requiredControllers){
              var controllerA = requiredControllers[0];
              var controller = requiredControllers[1];
            }
    };
  });
```

If you specify the name of another directive in the `require` option, the controller for that directive is linked. For example:

Click here to view code image

```
  directive('myDirective', function() {
```

```
  return {
    require: '^myOtherDirective',
    link: function(scope, element, attrs, otherDirectiveController){
          }
  };
});
```

Configuring the Directive Scope

Directives share the scope with the parent by default. This is typically adequate for most needs. The biggest downside is that you might not want to include all the custom directive properties in the parent scope, especially if the parent scope is the root scope.

To solve that problem, you can define a separate scope for the directive using the `scope` property. The following sections describe how to add an inherited scope and an isolate scope.

Adding an Inherited Scope

The simplest method to add a scope to a directive is to create one that inherits from the parent scope. The advantage is that you have a scope separate from the parent to add additional values to, but the disadvantage is that the custom directives can still modify values in the parent scope.

To create an inherited scope for the custom directive, set the `scope` property of the directive to `true`. For example:

Click here to view code image

```
directive('myDirective', function() {
  return {
    scope: true
  };
});
```

Adding an Isolate Scope

At times, you might want to separate the scope inside a directive from the scope outside the directive. Doing so prevents the possibility of the directive changing values in the scope of the parent controller. The directive definition enables you to specify a `scope` property that creates an isolate scope. An isolate scope isolates the directive scope from the outer scope to prevent the directive from accessing the outer scope and the controller in the outer scope from altering the directive scope. For example, the following isolates the scope of the directive from the outside scope:

Click here to view code image

```
directive('myDirective', function() {
  return {
    scope: { },
    templateUrl: '/myDirective.html'
    };
  });
```

Using this code, the directive has a completely empty isolate scope. However, you might want to still map some items in the outer scope to the directive's inner scope. You can use the following prefixes to attribute names to make local scope variables available in the directive's scope:

- **@:** Binds a local scope string to the value of the DOM attribute. The value of the attribute will be available inside the directive scope.
- **=:** Creates a bidirectional binding between the local `scope` property and the directive `scope` property.
- **&:** Binds a function in the local scope to the directive scope.

If no attribute name follows the prefix, the name of the directive property is used. For example:

```
title: '@'
```

is the same as:

```
title: '@title'
```

The following code shows how to implement each of the methods to map local values into a directive's isolate scope:

Click here to view code image

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.title="myApplication";
    $scope.myFunc = function(){
      console.log("out");
    };
  }).
  directive('myDirective', function() {
    return {
      scope: {title: '=', newFunc:"&myFunc", info: '@'},
      template: '<div ng-click="newFunc()">{{title}}: {{info}}</div>'
    };
  });
```

The following code shows how to define the directive in the AngularJS template to provide the necessary attributes to map the properties:

```
<div my-directive
     my-func="myFunc()"
     title="title"
     info="SomeString"></div>
```

Transcluding Elements

Transcluding can be kind of a difficult concept to pick up on at first. Basically, the idea is that you can keep the contents of the custom directive defined in an AngularJS template and bind them to the scope. The way this works is that the linking function for the directive receives a transclusion function that is prebound to the current scope. Then elements inside the directive have access to the scope outside the directive.

You can set the `transclude` option to the following values:

- **true:** Transcludes the content of the directive.
- **'element':** Transcludes the whole element, including any directives defined at lower priorities.

You must also include the `ngTransclude` directive in elements inside your directive template. The following is an example of implementing `transclude` to access the `title` variable in the controller scope from the `myDirective` directive template:

Click here to view code image

```
angular.module('myApp', []).
  directive('myDirective', function() {
    return {
      transclude: true,
      scope: {},
      template: '<div ng-transclude>{{title}}</div>'
    };
  }).
  controller('myController', function($scope) {
    $scope.title="myApplication";
  });
```

Manipulating the DOM with a Link Function

When the AngularJS HTML compiler encounters a directive, it runs the directive's compile function, which returns the `link()` function. The `link()` function is added to the list of AngularJS directives. After all directives have been compiled, the HTML compiler calls the `link()` functions in order, based on priority.

If you want to modify the DOM inside a custom directive, you should use a `link()` function. The `link()` function accepts the `scope`, `element`, `attributes`, `controller`, and `transclude` function associated with the directive, enabling you to manipulate the DOM directly within the directive. The transclude function is a handle that is bound to the transclusion scope.

Inside the `link()` function, you handle the `$destroy` event on the directive element and clean up anything necessary. The `link()` function is also responsible for registering DOM listeners to handle browser events.

The `link()` function uses the following syntax:

Click here to view code image

```
link: function(scope, element, attributes, [controller], [transclude])
```

The `scope` parameter is the scope of the directive, `element` is the element where the directive will be inserted, `attributes` lists the attributes declared on the element, and `controller` is the controller specified by the `require` option. The `transclude` parameter is a handle to the transclude function.

The transclude function provides access to the element created when the transclusion of the contents of the original element occurs. To access the transcluded element with the inherited scope, you can call the transclude function. This allows you to access and manipulate the transcluded element within the link function. For example:

Click here to view code image

```
link: function link(scope, elem, attr, controller, transcludeFn){
        var transcludedElement = transcludeFn();
      }
```

You can also access the clone of the transcluded element by specifying a `clone` parameter. For example:

Click here to view code image

```
link: function link(scope, elem, attr, controller, transcludeFn){
        transcludeFn(function(clone){
          //access clone here . . .
        });
      }
```

You can also access the clone of the transcluded element with a different scope by applying a `scope` parameter as well as the `clone` parameter. For example:

Click here to view code image

```
link: function link(scope, elem, attr, controller, transcludeFn){
        transcludeFn(scope, function(clone){
          //access clone here . . .
        });
      }
```

The following directive shows the implementation of a basic `link()` function that sets a scope variable, appends data to the DOM element, implements a `$destroy` event handler, and adds a `$watch` to the scope:

Click here to view code image

```
directive('myDirective', function() {
  return {
    scope: {title: '='},
    require: '^otherDirective',
    link: function link(scope, elem, attr, controller, transclude){
      scope.title = "new";
      elem.append("Linked");
```

```
        elem.on('$destroy', function() {
          //cleanup code
        });
        scope.$watch('title', function(newVal){
          //watch code
        });
      }
    };
```

The `link` property can also be set to an object that includes pre and post properties that specify prelink and postlink functions. In the preceding example, where link is set to a function, the function is executed as a postlink function, meaning that it is executed after the child elements are already linked, whereas the prelinked function is executed before the child elements are linked. Therefore, you should do DOM manipulation only in the postlink function. In fact, it is quite rare to need to include the prelink function.

The following shows an example of the syntax for including both the prelink and the postlink functions:

Click here to view code image

```
directive('myDirective', function() {
  return {
  link: {
    pre: function preLink(scope, elem, attr, controller){
           //prelink code
         },
    post: function postLink(scope, elem, attr, controller){
           //postlink code
         },
  }
};
```

Manipulating the DOM with a Compile Function

The compile function is very similar to the link function with one major advantage, but several drawbacks. The advantage and the main reason to use the compile function is performance. The compile method is executed only once when compiling the template, whereas the link function is executed each time the element is linked; for example, if you are applying multiple directives inside an `ng-repeat` loop or when the model changes. If you are doing a large number of DOM manipulations, that can be a big deal.

These are the limitations of the compile function:

- Any manipulations are applied before cloning takes place. That means that when the custom directive is used inside an `ng-repeat`, any DOM manipulations will be applied to *all* the custom directives generated.
- The `compile()` method cannot handle directives that recursively use themselves in their own templates or `compile()` functions because that would result in an infinite loop.

- The `compile()` method does not have access to the scope.
- The `transclude` function has been deprecated and removed from the compile function so you cannot link to the transcluded elements.

The syntax for the `compile()` method is very similar to that for the `link()` method. You can specify a single postlink function such as this:

Click here to view code image

```
directive('myDirective', function() {
  return {
  compile: function compile(scope, elem, attr, controller){
             //postlink code
           }
};
```

You can also specify pre- and postlink functions using an object as shown here:

Click here to view code image

```
directive('myDirective', function() {
  return {
  compile: {
    pre: function preLink(elem, attr){
           //prelink code
         },
    post: function postLink(elem, attr){
           //postlink code
         },
  }
};
```

Implementing Custom Directives

The types of custom directives you can define are limitless, and this makes AngularJS extensible. Custom directives are the most complex portion of AngularJS to explain and to grasp. The best way to get started is to show you some examples of custom directives, to give you a feel for how to implement them and have them interact with each other.

Try it Yourself: Manipulating the DOM in Custom Directives

One of the most common tasks you will be performing in custom directives is manipulating the DOM. This should be the only place in your AngularJS apps that you manipulate the DOM. In this exercise, you build a basic custom directive that applies a box with title and footer around the elements that are contained within. This example is basic and gives you a chance to see how to use some of the mechanisms in AngularJS, such as setting values as attributes to the custom directive.

Use the following steps to build the application with a custom directive that manipulates the DOM:

1. Add the lesson25/directive_custom_dom.html and lesson25/js/directive_custom_dom.js files.

2. Add the code shown in Listing 25.1 and Listing 25.2 to the HTML and JavaScript files.

3. The following lines of code from Listing 25.1 define a simple application with a controller that contains only the scope variable `title` and a directive named `mybox`:

Click here to view code image

```
angular.module('myApp', [])
.controller('myController', function($scope) {
    $scope.title="myApplication";
  })
.directive('mybox', function() {
  return {
. . .
    };
  });
```

4. The following line of code in the directive enables transclusion:

```
    transclude: true,
```

5. The following line of code restricts the custom directive to element names only:

```
    restrict: 'E',
```

6. The following line of code defines an isolate scope accepting the parameters `title` and `bwidth` as strings:

Click here to view code image

```
    scope: {title: '@', bwidth: '@bwidth'},
```

7. The following lines define a `template` that adds the title bar and a `<div ng-transclude>` to store the transcluded content of the custom directive:

Click here to view code image

```
    template: '<div><span class="titleBar">{{title}}' +
              '</span><div ng-transclude></div></div>',
```

8. The following lines show the `link()` function that uses `append` to append a footer element. This could also have been done in the `template`; however, I wanted to illustrate that it can also be done in the `link()` function. Also note that the text of the footer element is coming from the title value of the parent scope using `scope.$parent.title`. The `link` function also adds a `border` and sets the `display` and `width` values based on the `bwidth` value in the scope:

Click here to view code image

```
    link: function (scope, elem, attr, controller, transclude){
        elem.append('<span class="footer">' + scope.$parent.title +
'</span>');
        elem.css('display', 'inline-block');
        elem.css('width', scope.bwidth);
      },
```

9. The code in Listing 25.2 implements an AngularJS template that sets up some CSS styles and then adds the `<mybox>` custom directive that was defined in Listing 25.1. Notice that the content of the directive varies from a string to an image to a paragraph.

10. Open directive_custom_dom.html in a browser. The results are shown in Figure 25.1. Notice that the `bwidth` attribute size determines the width of the box, and all the elements are surrounded by the same type of box.

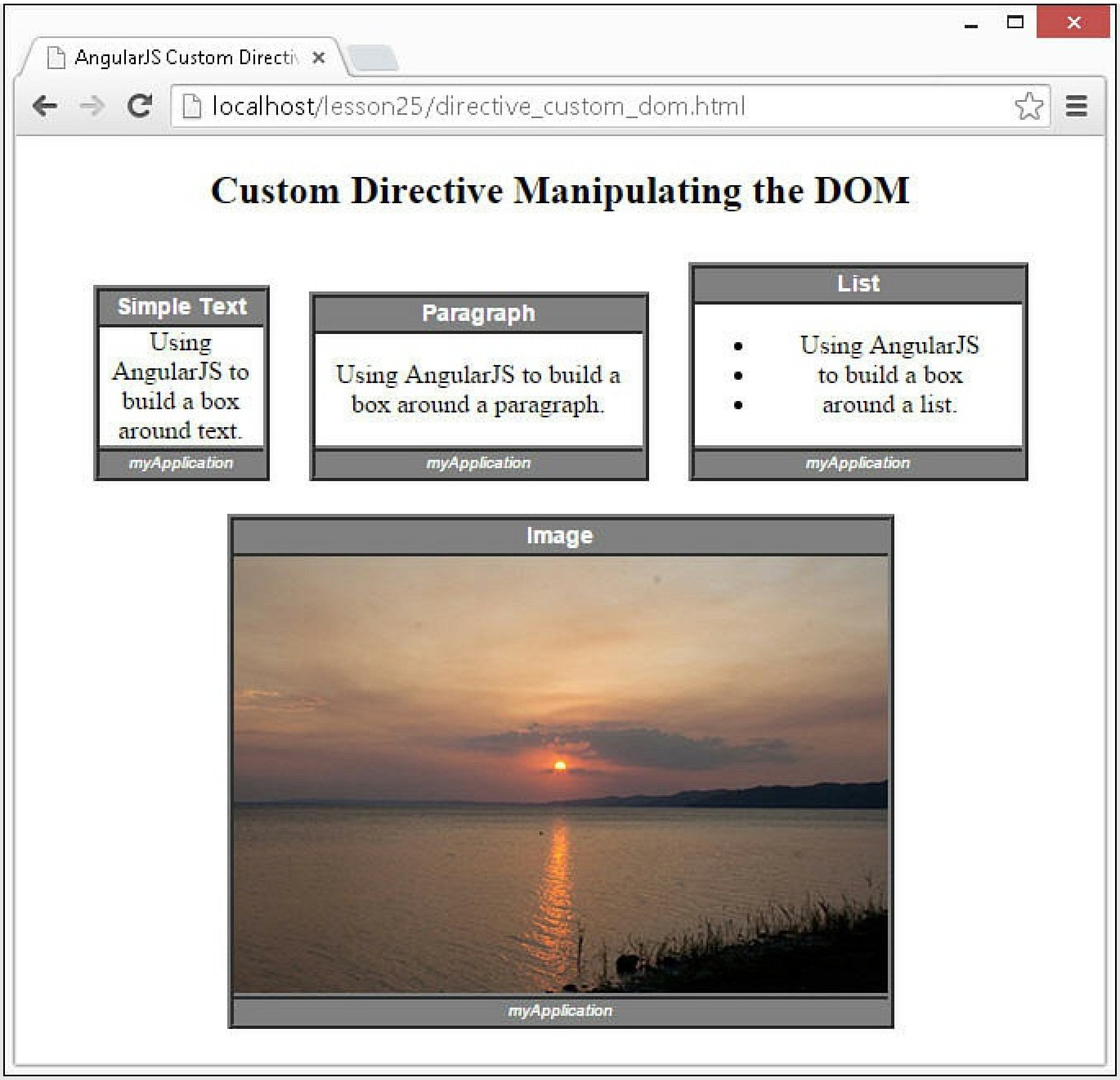

AngularJS Custom Directi
localhost/lesson25/directive_custom_dom.html
Custom Directive Manipulating the DOM
Simple Text
Using AngularJS to build a box around text.
myApplication
Paragraph
Using AngularJS to build a box around a paragraph.
myApplication
List
Using AngularJS to build a box around a list.
myApplication
Image
myApplication

FIGURE 25.1 Implementing custom directives that manipulate DOM elements in AngularJS template views.

LISTING 25.1 directive_custom_dom.js Implementing Custom Directives That Manipulate the DOM

Click here to view code image

```
angular.module('myApp', [])
.controller('myController', function($scope) {
    $scope.title="myApplication";
  })
.directive('mybox', function() {
  return {
    transclude: true,
    restrict: 'E',
    scope: {title: '@', bwidth: '@bwidth'},
    template: '<div><span class="titleBar">{{title}}' +
              '</span><div ng-transclude></div></div>',
    link: function (scope, elem, attr, controller, transclude){
        elem.append('<span class="footer">' + scope.$parent.title +
'</span>');
        elem.css('display', 'inline-block');
        elem.css('width', scope.bwidth);
      },
    };
  });
```

LISTING 25.2 directive_custom_dom.html An AngularJS Template That Utilizes a Custom Directive That Manipulates the DOM

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Directive</title>
  <style>
    * { text-align: center; }
    .titleBar { color: white; background-color: grey;
                font: bold 14px/18px arial; display: block;
                border-bottom: 4px ridge grey; }
    .footer {  color: white; background-color: grey;
               font: italic 10px/14px arial; display: block;
               border-top: 4px ridge grey; }
    mybox { border: 4px ridge grey; margin: 10px; }
    img { display: block; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h2>Custom Directive Manipulating the DOM</h2>
    <mybox title="Simple Text" bwidth="100px">
      Using AngularJS to build a box around text.
    </mybox>
    <mybox title="Paragraph" bwidth="200px">
      <p>Using AngularJS to build a box around a paragraph.</p>
    </mybox>
    <mybox title="List" bwidth="200px">
      <ul>
        <li>Using AngularJS</li>
        <li>to build a box</li>
        <li>around a list.
        </li>
      </ul>
    </mybox>
    <mybox title="Image" bwidth="400px">
      <img src="/images/sunset2.jpg" width="400px" />
    </mybox>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_custom_dom.js"></script>
</body>
</html>
```

Try it Yourself: Implementing Event Handlers in a Custom Directive

Another common use of custom directives is to implement event handlers to interact with mouse and keyboard events that are occurring in the custom elements. This enables you to provide enhanced user interactions to the custom elements.

In this example, you add mouse event handlers that enable you to use drag operations on images to resize and adjust the opacity. When the mouse is dragged left, the image shrinks; right, the image enlarges; up, the image fades; and down, the opacity increases. For this example, I have included the full version of jQuery by loading it in the AngularJS template. I chose to do so to use some capabilities like getting the image `width()` that is not available in jQuery lite.

Use the following steps to build the application with a custom directive that manipulates the DOM:

1. Add the lesson25/directive_custom_zoom.html and lesson25/js/directive_custom_zoom.js files.
2. Add the code shown in Listing 25.3 and Listing 25.4 to the HTML and JavaScript files.
3. The following lines of code from Listing 25.3 define a simple application with two directives, `zoomit` and `fadit`:

Click here to view code image

```
angular.module('myApp', [])
.directive('zoomit', function() {
  return {
    link: function (scope, elem, attr){
. . .
})
.directive('fadeit', function() {
  return {
    link: function (scope, elem, attr){
```

```
. . .
    }
  };
});
```

4. The following lines in the `link` function of the first directive, `zoomit`, listen on the `mousedown`, `mouseup`, `mouseleave`, and `mousemove` events. When the mouse button is pressed, the `dragging` variable is set to `true`, and when the mouse is released or the mouse leaves the element, `dragging` is set to `false`. Also note that in `mousedown`, the default event behavior is suppressed by `event.preventDefault()`. This is to eliminate any interaction conflicts with the default browser behavior while dragging:

Click here to view code image

```
      elem.on('mouseup', function(){
        dragging = false;
      });
      elem.on('mouseleave', function(){
        dragging = false;
      });
```

5. In the following lines in `zoomit`, define a `mousemove` handler to determine the position movement of the mouse and increment or decrement the image size accordingly. Notice that because the full version of jQuery is loaded, you were able to use the `width()` and `height()` functions to get and set the size of the image:

Click here to view code image

```
      elem.on('mousemove', function(event){
        if(dragging){
          var adjustment = null;
          if ( event.screenX > lastX+tolerance &&
              elem.width() < 300){
            adjustment = 1.1;
          } else if ( event.screenX < lastX-tolerance &&
              elem.width() > 100){
            adjustment = .9;
          }
          if(adjustment){
            //requires full jQuery library
            elem.width(elem.width()*adjustment);
            elem.height(elem.height()*adjustment);
            lastX = event.screenX;
          }
        }
      });
```

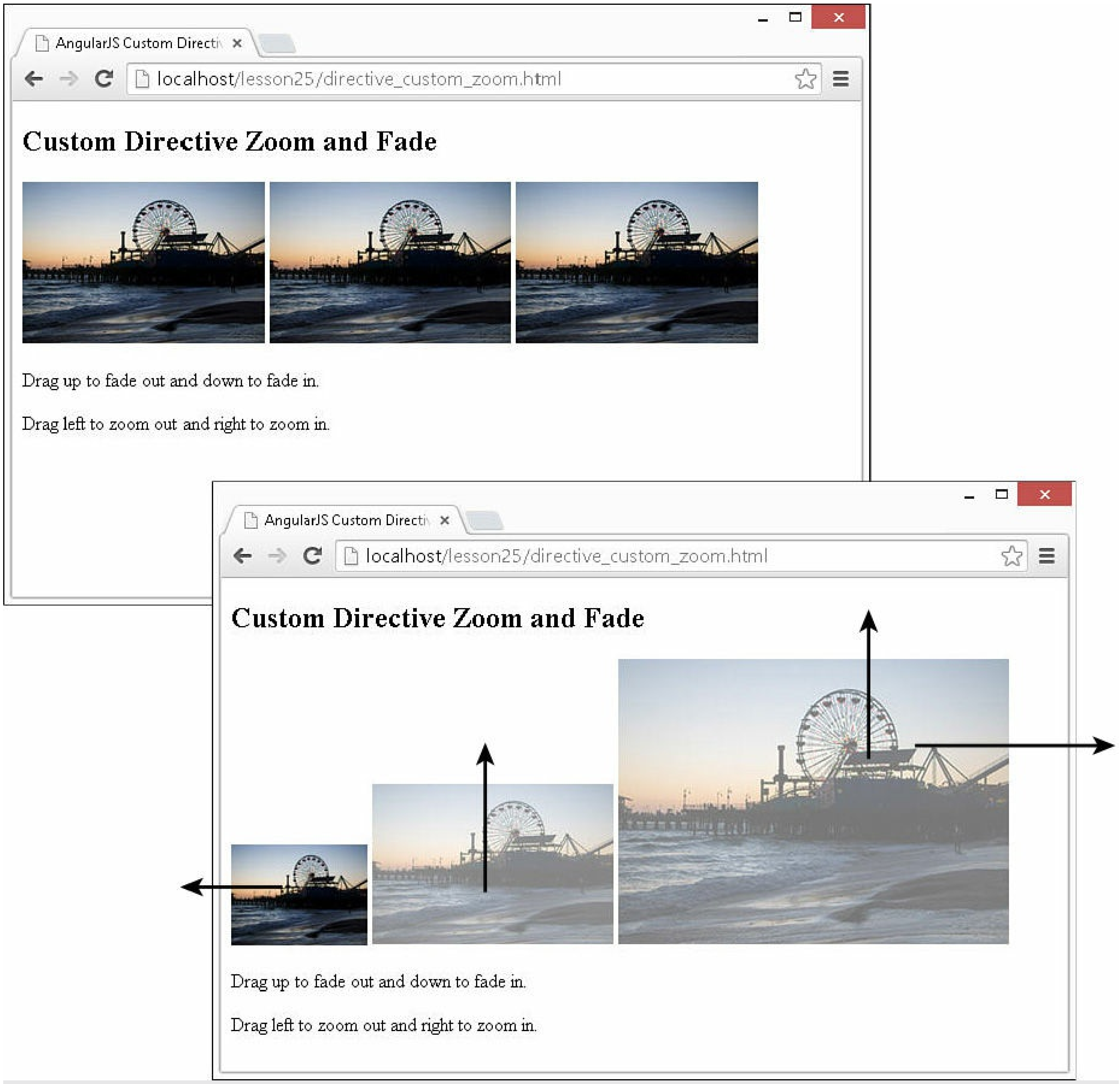
6. Look at the `fadeit` directive, and you can see that it is very similar to the zoomit directive, with the exception that the opacity value of the image is changed.

7. The fadeit and zoomit directives are used on the following <img> element from the AngularJS template shown in Listing 25.4. Notice that on the first image, the zoomit directive is added, on the second image the fadeit directive is added, and on the final image both are added. This shows you that multiple custom directives can be added to the same element:

Click here to view code image

```
<img src="/images/wheel.jpg" zoomit></img>
<img src="/images/wheel.jpg" fadeit></img>
<img src="/images/wheel.jpg" zoomit fadeit></img>
```

8. Open the directive_custom_zoom.js file in a browser. The results are shown in Figure 25.2. The first image is shrunk by dragging left, the second image is faded by dragging up, and the third image is expanded and faded.


AngularJS Custom Directi
localhost/lesson25/directive_custom_zoom.html

Custom Directive Zoom and Fade

Drag up to fade out and down to fade in.

Drag left to zoom out and right to zoom in.

AngularJS Custom Directi
localhost/lesson25/directive_custom_zoom.html

Custom Directive Zoom and Fade

Drag up to fade out and down to fade in.

Drag left to zoom out and right to zoom in.

FIGURE 25.2 Implementing custom directives that provide interactions with mouse events to manipulate DOM elements.

LISTING 25.3 directive_custom_zoom.js Implementing Custom Directives That Register with DOM Events

Click here to view code image

```
angular.module('myApp', [])
.directive('zoomit', function() {
  return {
    link: function (scope, elem, attr){
      var dragging = false;
      var tolerance = 10;
      var lastX = 0;
      elem.on('mousedown', function(event){
        lastX = event.screenX;
        event.preventDefault();
        dragging = true;
      });
      elem.on('mouseup', function(){
        dragging = false;
      });
      elem.on('mouseleave', function(){
        dragging = false;
      });
      elem.on('mousemove', function(event){
        if(dragging){
          var adjustment = null;
          if ( event.screenX > lastX+tolerance &&
              elem.width() < 300){
            adjustment = 1.1;
          } else if ( event.screenX < lastX-tolerance &&
              elem.width() > 100){
            adjustment = .9;
          }
          if(adjustment){
            //requires full jQuery library
            elem.width(elem.width()*adjustment);
            elem.height(elem.height()*adjustment);
            lastX = event.screenX;
          }
        }
      });
    }
  };
})
.directive('fadeit', function() {
  return {
    link: function (scope, elem, attr){
      var dragging = false;
      var tolerance = 10;
      var lastY = 0;
      elem.on('mousedown', function(event){
        lastY = event.screenY;
        event.preventDefault();
        dragging = true;
      });
      elem.on('mouseup', function(){
        dragging = false;
      });
      elem.on('mouseleave', function(){
        dragging = false;
      });
      elem.on('mousemove', function(event){
        if(dragging){
          var adjustment = null;
          var currentOpacity = parseFloat(elem.css("opacity"));
          if ( event.screenY > lastY+tolerance &&
              currentOpacity < 1){
            adjustment = 1.1;
          } else if ( event.screenY < lastY-tolerance &&
              currentOpacity > 0.5){
            adjustment = .9;
          }
          if(adjustment){
            //requires full jQuery library
            elem.css("opacity", currentOpacity*adjustment);
            lastY = event.screenY;
          }
        }
      });
    }
  };
});
```

LISTING 25.4 directive_custom_zoom.html An AngularJS Template That Utilizes a Custom Directive to Provide Interactions with Mouse Events

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Directive</title>
  <style>
    img { width: 200px; }
  </style>
</head>
<body>
  <h2>Custom Directive Zoom and Fade</h2>
  <img src="/images/wheel.jpg" zoomit></img>
  <img src="/images/wheel.jpg" fadeit></img>
  <img src="/images/wheel.jpg" zoomit fadeit></img>
  <p>Drag up to fade out and down to fade in.</p>
  <p>Drag left to zoom out and right to zoom in.</p>
  <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/directive_custom_zoom.js"></script>
</body>
</html>
```

Try it Yourself: Implementing Nested Directives

The final example illustrates how you can nest directives within each other and have them interact. Nesting directives is a great way to provide a parent context and container for custom elements that are related to each other. In this example, the outer directive `myPhotos` acts as a container that can contain child directives named `myPhoto`.

The idea is that you can add an `<my-photos></my-photos>` element and then add `<my-photo></my-photo>` elements within it. Each `myPhoto` element is added to the scope of the controller of `myPhotos`, allowing the photo title to be displayed and the `active` property of each `myPhoto` element to be easily toggled on and off.

Use the following steps to build the application with a custom directive that manipulates the DOM:

1. Add the lesson25/directive_custom_photos.html, my_photos.html, and lesson25/js/directive_custom_photos.js and files.
2. Add the code shown in Listing 25.5, Listing 25.6, and Listing 25.7 to the HTML and JavaScript files.
3. The following lines of code from Listing 25.5 implement the `myPhotos` directive. The `myPhotos` directive is designed to be a container for the `myPhoto` directive that is defined later. Notice that the following lines define a controller that provides the functionality for the `myPhotos` directive, including an `addPhoto()` function:

Click here to view code image

```
.directive('myPhotos', function() {
  return {
    restrict: 'E',
    transclude: true,
    scope: {},
    controller: function($scope) {
      var photos = $scope.photos = [];
      $scope.select = function(photo) {
        angular.forEach(photos, function(photo) {
          photo.selected = false;
        });
        photo.selected = true;
      };
      this.addPhoto = function(photo) {
        photos.push(photo);
      };
    },
    templateUrl: 'my_photos.html'
  };
})
```

4. The following line from the `myPhotos` directive specifies that

my_photos.html should be used as the template when rendering the directive:

Click here to view code image

```
    templateUrl: 'my_photos.html'
```

5. The following lines from Listing 25.7 implement a partial template loaded by the `myPhotos` directive. It generates a `<div>` container and then uses the `photos` array in the `myPhotos` scope to build a list of links bound to the `select()` function, using `ng-click`. `<div ng-transclude></div>` provides the container for the `myPhoto` child elements:

Click here to view code image

```
<div>
  <div  class="imgList" >
      <li ng-repeat="photo in photos"
            ng-class="{active:photo.selected}">
        <a href="" ng-click="select(photo)">{{photo.title}}</a>
      </li>
  </div>
  <div class="imgView" ng-transclude></div>
</div>
```

6. The following code from Listing 25.5 defines the myPhoto directive. Because the code uses `require:'^myPhotos'` in the `myPhoto` directive, you can also call the `addPhoto()` method from the `link()` function by using the `photosControl` handle to the `myPhotos` controller:

Click here to view code image

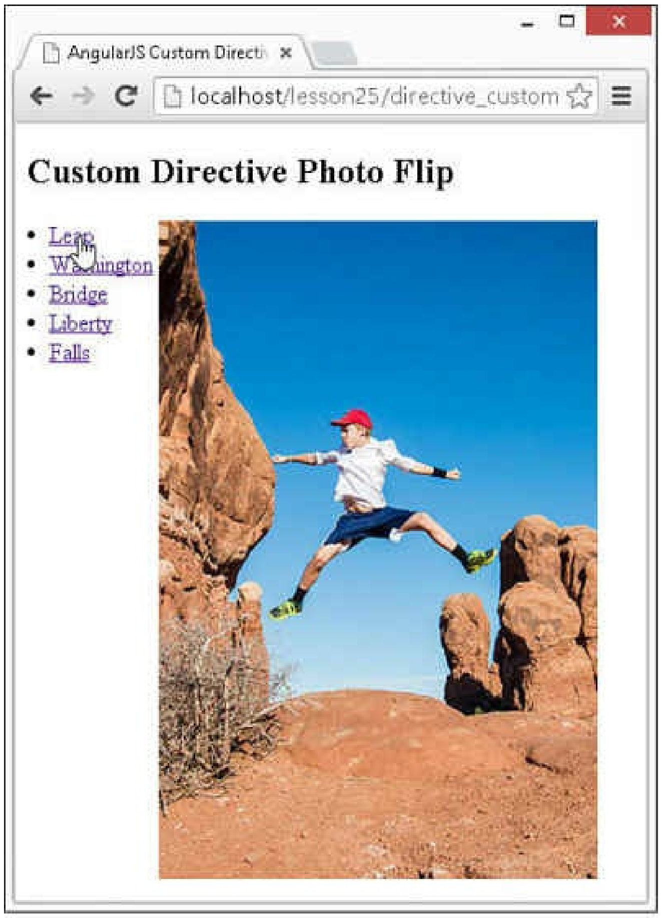
```
.directive('myPhoto', function() {
  return {
    require: '^myPhotos',
    restrict: 'E',
    transclude: true,
    scope: { title: '@'},
    link: function(scope, elem, attrs, photosControl) {
      photosControl.addPhoto(scope);
    },
    template: '<div ng-show="selected" ng-transclude></div>'
  };
});
```

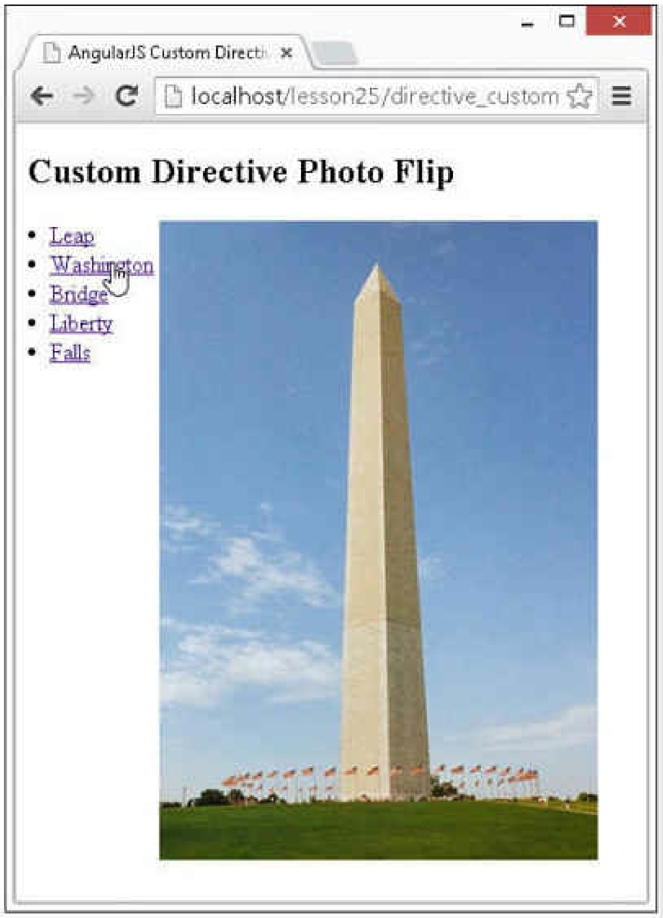
7. Listing 25.6 implements the `myPhotos` and `myPhoto` directives in an AngularJS template. The `myPhoto` directives are nested inside the `myPhotos` directive. Notice that the `title` attribute is set on each `myPhoto` directive. Using this methodology, shown next, you could add any number of `myPhoto` elements to the page and have them show up in the list:

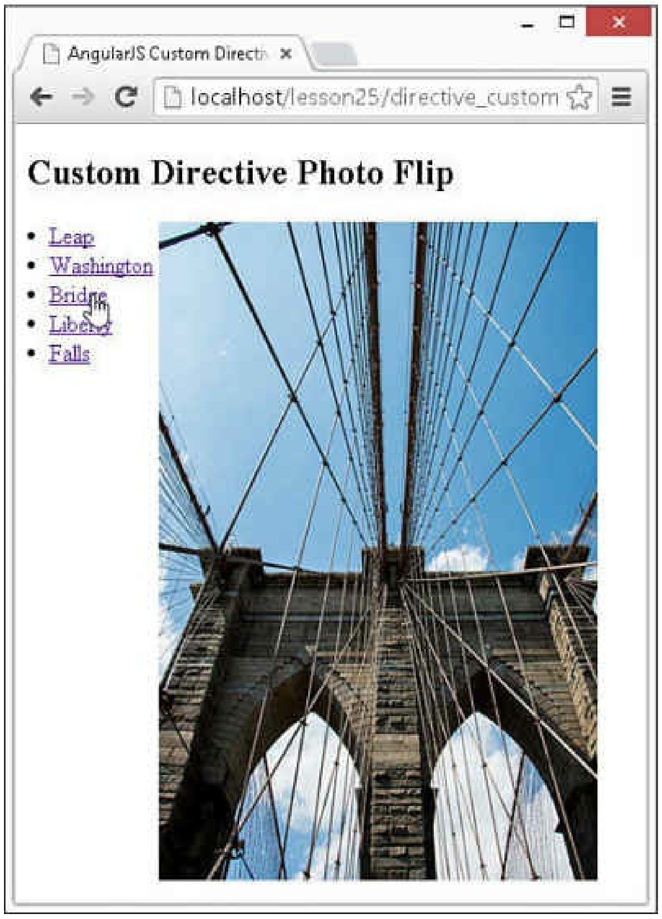
Click here to view code image

```
  <my-photos>
    <my-photo title="Leap">
      <img src="/images/power.jpg"/>
    </my-photo>
. . .
```

8. Open the directive_custom_photos.html file in a browser. Figure 25.3 shows how the images are shown and hidden each time a link is clicked.


AngularJS Custom Directi
localhost/lesson25/directive_custom
Custom Directive Photo Flip
Leap
Washington
Bridge
Liberty
Falls


AngularJS Custom Directi
localhost/lesson25/directive_custom
Custom Directive Photo Flip
Leap
Washington
Bridge
Liberty
Falls


AngularJS Custom Directi
localhost/lesson25/directive_custom
Custom Directive Photo Flip
Leap
Washington
Bridge
Liberty
Falls

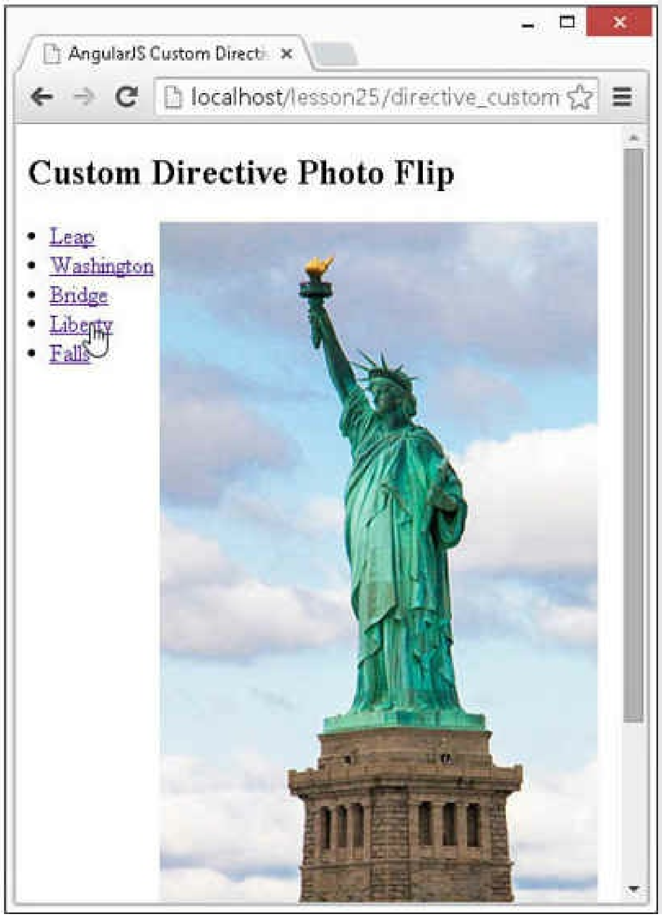

AngularJS Custom Directi
localhost/lesson25/directive_custom
Custom Directive Photo Flip
Leap
Washington
Bridge
Liberty
Falls

FIGURE 25.3 Implementing event directives in AngularJS template views.

LISTING 25.5 directive_custom_photos.js Implementing Custom Directives That Interact with Each Other

Click here to view code image

```
angular.module('myApp', [])
.directive('myPhotos', function() {
  return {
    restrict: 'E',
    transclude: true,
    scope: {},
    controller: function($scope) {
      var photos = $scope.photos = [];
      $scope.select = function(photo) {
        angular.forEach(photos, function(photo) {
          photo.selected = false;
        });
        photo.selected = true;
      };
      this.addPhoto = function(photo) {
        photos.push(photo);
      };
    },
    templateUrl: 'my_photos.html'
  };
})
.directive('myPhoto', function() {
  return {
    require: '^myPhotos',
    restrict: 'E',
    transclude: true,
    scope: { title: '@'},
    link: function(scope, elem, attrs, photosControl) {
      photosControl.addPhoto(scope);
    },
    template: '<div ng-show="selected" ng-transclude></div>'
  };
});
```

LISTING 25.6 directive_custom_photos.html An AngularJS Template That Implements Nested Custom Directives

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
```

```
  <title>AngularJS Custom Directive</title>
  <style>
    img {
      width: 300px }
    .imgView, .imgList {
      vertical-align: top; display:inline-block; }
  </style>
</head>
<body>
  <h2>Custom Directive Photo Flip</h2>
   <my-photos>
     <my-photo title="Leap">
       <img src="/images/power.jpg"/>
     </my-photo>
     <my-photo title="Washington">
       <img src="/images/washington.jpg"/>
     </my-photo>
     <my-photo title="Bridge">
       <img src="/images/bridge.jpg"/>
     </my-photo>
     <my-photo title="Liberty">
       <img src="/images/liberty.jpg"/>
     </my-photo>
     <my-photo title="Falls">
       <img src="/images/falls2.jpg"/>
     </my-photo>
   </my-photos>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/directive_custom_photos.js"></script>
</body>
</html>
```

LISTING 25.7 my_photos.html A Partial AngularJS Template That Provides the Root Element for the myPhotos Custom Directive

Click here to view code image

```
<div>
  <div  class="imgList" >
      <li ng-repeat="photo in photos"
            ng-class="{active:photo.selected}">
        <a href="" ng-click="select(photo)">{{photo.title}}</a>
      </li>
  </div>
  <div class="imgView" ng-transclude></div>
</div>
```

Summary

One of the most powerful features of AngularJS is the capability to create your own custom directives. Implementing a custom directive in code is simple using the `directive()` method on a `Module` object. However, directives can also be very complex because of the myriad ways they can be implemented. This lesson has given you a small taste of what can be done with custom directives in AngularJS. Try spending some time playing around with the examples and writing some of your own to familiarize yourself as much as possible with how they work.

Q&A

Q. Is there a way to ensure that the name I choose for a custom directive will not be used by someone else?

A. Currently there is no way to enforce that. The best way is to come up with a unique acronym to use as the prefix for your custom AngularJS directives.

Q. Is it better to implement custom directives as tag, attribute, class, or comment names?

A. The best practice is to use tag names or attribute names when it makes sense, rather than comment and class names.

Workshop

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

Quiz

1. True or false: When using the scope in a custom directive, the scope has to be shared with the parent scope.
2. True or false: The property `'type'` can be set to a string that is not an HTML root node.
3. What is the syntax to instruct the injector service to look in parent contexts for a controller when adding a controller to a directive?
4. Which function(s) should you use to manipulate the DOM in a custom directive?

Quiz Answers

1. False. The scope can share the parent scope, inherit from it, or be its own isolate scope.
2. True. `'type'` can be used not only with HTML, but with `svg` and `math` elements as well.

3. Use the `require: '^controllerName'` option in the directive.
4. The `link` and `compile` functions.

Exercises

1. Modify the example code in Listings 25.1 and 25.2. Add a `bheight` parameter similar to the `bwidth` parameter to set the height of the box as well as the width.
2. Modify the example code in Listings 25.3 and 25.4. Add a new directive named `movit` that changes the x and y positions of the element on the page. You will need to change the elements to be absolute positioned.
3. Modify the example code in Listing 25.6 to create a second `<my-photos>` block and include 10 or 15 additional images so that you can see how easy it is to reuse custom directives.

Lesson 26. Using Events to Interact with Data in the Model

What You'll Learn in This Lesson:

- How to use `$watch` to track scope variables
- How to use `$watchCollection` to track changes
- How to emit a custom event
- How to handle custom events
- How to implement custom events

Events are one of the most critical components in most AngularJS applications. Events enable users to interact with elements as well as the application to know when to perform certain tasks. This lesson discusses different types of events that you have and will be working with in your AngularJS applications.

Specifically, this lesson discusses four types of events, including browser events, user interaction events, scope-based events, and custom events. You have already been introduced to some of this in previous lessons. The reason this is positioned here is that the previous lessons give perspective to the discussion that follows.

Browser Events

Several events are triggered by the browser itself. In a way, these are also user-interaction events; however, I want to keep them separate for this discussion. The browser events include things like the ready event when the document is loaded and the resize event when the browser is resized.

These are useful to know when the view of the user is changing. You have already seen how to use the `.on()` function to add handlers for events; the question is where to put the handler. The best solutions I've seen involve a handler that is registered in the run block for the entire application. Any information can then be made available to subsequent components through the scope model or through a cache service.

User Interaction Events

You have already been exposed to user interaction events. These include mouse and keyboard events, as well as other events such as the focus and blur events. There are typically two places where you will implement interactions for user events. One is using the `ng` event directives, such as `ng-click`, and simple interactions in the view and controller code. The second place to add user event interactions is in the link function of custom directives.

You have already been exposed to both of these methods in previous lessons; in this lesson, you'll see that you have a choice of which one to use. The advantage of using the built-in `ng` event directives such as `ngClick` is that you do not have to add the complexity of creating a custom directive for simple requirements.

There are a couple of downsides to using the `ng` event directives, though. One is that you should not be doing DOM manipulation in the controller, which is where you can define handlers for the `ng` directives. Another downside is that you will have to implement the `ngClick` code in the template every time you want the functionality. For example, consider the following template code to add mouse event handlers to an element:

Click here to view code image

```
<span
  ng-mouseenter="mouseEntered(event)"
  ng-mouseleave="mouseLeft(event)"
  ng-click="clicked(event)">
<span>
```

The code isn't too bad, but what if there are several locations where you want the same functionality? If they fall in an `ng-repeat` block, it's not too bad, but otherwise it's a pain. A good rule to follow is that if you want to use the functionality in more than one place, and definitely if you will reuse it in multiple applications, you should define a custom directive that implements the handlers.

Adding $watches to Track Scope Change Events

Another common event that you will be using is triggered not by the browser, but by changes to the data in the model. This capability enables you to react to model changes without having to add code at every point where the values might change. This capability is useful because often the data in the model might be changing in various ways—user input, service updates, and so on.

Using $watch to Track a Scope Variable

To add the capability to handle changes to scope values, you need to add a `$watch` to the variable in the scope using the `$watch` functionality built in to AngularJS. The `$watch` function in the scope uses the following syntax:

Click here to view code image

```
$watch(watchExpression, listener, [objectEquality])
```

The watchExpression is the expression in the scope to watch. This expression is called on every `$digest()` and returns the value that will be watched. The listener defines a function that will be called when the value of the watchExpression changes to a new value. The listener will not be called if the watchExpression is changed to the value it is already set to. The objectEquality is a Boolean that when true will use the `angular.equals()` function to determine equality instead of the more strict `==!` operator. You should be careful when using objectEquality because on complex objects, it can result in increases in memory and performance usage.

The following shows an example of adding `$watch` on the scope variable score:

Click here to view code image

```
$scope.score = 0;
$scope.$watch('score', function(newValue, oldValue) {
  if(newValue > 10){
    $scope.status = 'win';
  }
});
```

Using $watchGroup to Track Multiple Scope Variables

AngularJS also provides the capability to watch an array of expressions using the `$watchGroup` method. The `$watchGroup` method works the same as `$watch` except that the first parameter is an array of expressions to watch. The listener will be passed an array with the new and old values for the watched variables. For example, if you wanted to watch the variables `score` and `time`, you would use this:

Click here to view code image

```
$scope.score = 0;
$scope.time = 0;
$scope.$watchGroup(['score', 'time'], function(newValues, oldValues) {
  if(newValues[0] > 10){
    $scope.status = 'win';
  } else if (newValues[1] > 5{
    $scope.status = 'times up';
});
```

Using $watchCollection to Track Changes to Properties of an Object in the Scope

You can also watch the properties of an object using the `$watchCollection` method. The `$watchCollection` method takes an object as the first parameter and watches the properties of the object. In the case of an array, the individual values of the array are watched. For example:

Click here to view code image

```
$scope.scores = [5, 10, 15, 20];
$scope.$watchGroup('scores', function(newValue, oldValue) {
  $scope.newScores = newValue;
});
```

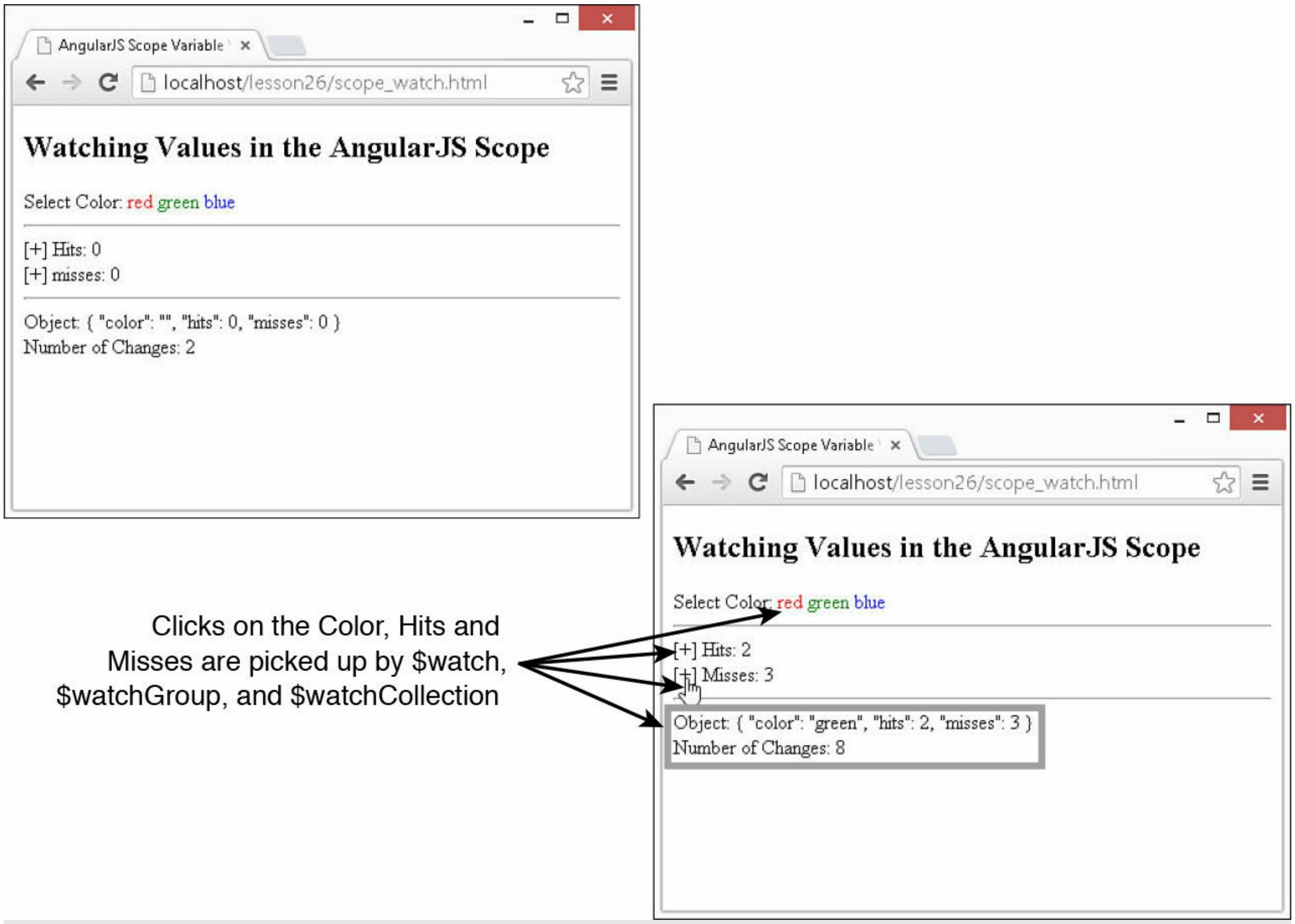
Try it Yourself: Implementing Watches in a Controller

In this example, you get a chance to implement a watch, watch group, and watch collection in a controller to track changes and update the view dynamically. The code in Listings 26.1 and 26.2 demonstrate a simple example that implements the `$watch`, `$watchGroup`, and `$watchCollection` methods.

The code in Listing 26.1 implements a controller that stores the values `myColor`, `hits`, and `misses`, as well as an object named `myObj`, in the scope. The code in Listing 26.2 implements an AngularJS template that provides the interaction to change the values and view the updated watch items.

Use the following steps to build the application that watches values in the scope:

1. Add the lesson26/scope_watch.html and lesson26/js/scope_watch.js files.
2. Add the code shown in Listing 26.1 and Listing 26.2 to the HTML and JavaScript files.
3. The code in Listing 26.1 first sets several scope values including `myColor`, `hits`, and `misses`, which will be watched. The controller also defines `setColor()`, `hit()`, and `miss()` functions that allow those values to be changed from the browser.
4. The following lines of code from Listing 26.1 implement a `$watch` on the `myColor` value in the scope so that each time `myColor` changes, the `myObj.color` value can also be adjusted.
5. The following lines of code from Listing 26.1 implement a `$watchGroup` on the `hits` and `misses` values in the scope so that each time they change, the value of `myObj.hits` and `myObj.misses` are also changed.
6. The following lines of code from Listing 26.1 implement a `$watchCollection` on the `myObj` object in the scope so that each time `myObj` is changed, the `changes` value in the scope is incremented to track the number of time `myColor`, `hits`, and `misses` change.
7. The code in Listing 26.2 implements an AngularJS template that enables the user to use the mouse to select the color and increment the `hits` and `misses` variables. The object and change values are displayed at the bottom, showing how the `$watch` methods detect and update changes to the scope.
8. Open the scope_watch.html file in a browser and change the color and increment the hits and misses to see the watched values in `myObj` and `changes` change. Figure 26.1 shows the rendered AngularJS web page.


AngularJS Scope Variable

localhost/lesson26/scope_watch.html

Watching Values in the AngularJS Scope

Select Color: red green blue

[+] Hits: 0
[+] misses: 0

Object: { "color": "", "hits": 0, "misses": 0 }
Number of Changes: 2

AngularJS Scope Variable

localhost/lesson26/scope_watch.html

Watching Values in the AngularJS Scope

Select Color: red green blue

[+] Hits: 2
[+] Misses: 3

Object: { "color": "green", "hits": 2, "misses": 3 }
Number of Changes: 8

Clicks on the Color, Hits and Misses are picked up by $watch, $watchGroup, and $watchCollection

FIGURE 26.1 Using `$watch()`, `$watchGroup()`, and `$watchCollection()` handlers to watch the value of scope variables.

LISTING 26.1 scope_watch.js Implementing `$watch()`, `$watchGroup()`, and `$watchCollection()` Handlers to Watch the Value of Scope Variables

[Click here to view code image]

```
angular.module('myApp', [])
.controller('myController', function ($scope) {
  $scope.mColors = ['red', 'green', 'blue'];
  $scope.myColor = '';
  $scope.hits = 0;
  $scope.misses = 0;
  $scope.changes = 0;
  $scope.myObj = {color: '', hits: '', misses: ''};
  $scope.setColor = function (color){
    $scope.myColor = color;
  };
  $scope.hit = function (){
    $scope.hits += 1;
  };
  $scope.miss = function (){
    $scope.misses += 1;
  };
  $scope.$watch('myColor', function (newValue, oldValue){
    $scope.myObj.color = newValue;
  });
  $scope.$watchGroup(['hits', 'misses'], function (newValue, oldValue){
    $scope.myObj.hits = newValue[0];
    $scope.myObj.misses = newValue[1];
  });
  $scope.$watchCollection('myObj', function (newValue, oldValue){
    $scope.changes += 1;
  });
});
```

LISTING 26.2 scope_watch.html HTML Template Code That Provides the View and Interactions with the Scope and Controller Defined in Listing 26.1

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Scope Variable Watch</title>
    <style>
      span { cursor: pointer; }
    </style>
  </head>
  <body>
    <h2>Watching Values in the AngularJS Scope</h2>
    <div ng-controller="myController">
      Select Color:
      <span ng-repeat="mColor in mColors">
        <span ng-style="{color: mColor}"
              ng-click="setColor(mColor)">
          {{mColor}}</span>
      </span><hr>
      <span ng-click="hit()">[+]</span>
      Hits: {{hits}}<br>
      <span ng-click="miss()">[+]</span>
      misses: {{misses}}<hr>
      Object: {{myObj|json}} <br>
      Number of Changes: {{changes}}
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
    <script src="js/scope_watch.js"></script>
  </body>
</html>
```

Emitting and Broadcasting Custom Events

A great feature of scopes is the capability to emit and broadcast events within the scope hierarchy. Events enable you to send notification to different levels in the scope that an event has occurred. Events can be anything you choose, such as a value changed or threshold reached. This is extremely useful in many situations, such as letting child scopes know that a value has changed in a parent scope, or vice versa.

Emitting a Custom Event to the Parent Scope Hierarchy

To emit an event from a scope, you use the `$emit()` method. This method sends an event upward through the parent scope hierarchy. Any ancestor scopes that have registered for the event are notified. The `$emit()` method uses the following syntax, where name is the event name and args is zero or more arguments that are passed to the event handler functions:

Click here to view code image

```
scope.$emit(name, [args, . . .])
```

Broadcasting a Custom Event to the Child Scope Hierarchy

You can also broadcast an event downward through the child scope hierarchy by using the `$broadcast()` method. Any descendent scopes that have registered for the event are notified. The `$broadcast()` method uses the following syntax, where name is the event name and args is zero or more arguments that are passed to the event handler functions:

Click here to view code image

```
scope.$broadcast(name, [args, . . .])
```

Handling Custom Events with a Listener

To handle an event that is emitted or broadcasted, you use the `$on()` method. The `$on()` method uses the following syntax, where name is the name of the event to listen for:

```
scope.$on(name, listener)
```

The listener parameter is a function that accepts the event as the first parameter and any arguments passed by the `$emit()` or `$broadcast()` method as subsequent parameters. The `event` object has the following properties:

- **targetScope:** The scope from which `$emit()` or `$broadcast()` was called.

- **currentScope:** The scope that is currently handling the event.
- **name:** The name of the event.
- **stopPropagation():** A function that stops the event from being propagated up or down the scope hierarchy.
- **preventDefault():** A function that prevents default behavior in a browser event but only executes your own custom code.
- **defaultPrevented:** A Boolean that is `true` if `event.preventDefault()` has been called.

Try it Yourself: Implementing Custom Events in Nested Controllers

In this example, you use custom events to communicate changes between two controllers, one nested within another. The code in Listing 26.3 and Listing 26.4 provide an AngularJS application and template that illustrates the use of `$emit()`, `$broadcast()`, and `$on()` to send and handle events up and down the scope hierarchy.

In the example, there is a list of characters stored in a parent scope. You can easily click on characters in the list to display character information contained in a child scope. You can also delete the character in the child scope, which removes it from the list in the parent scope.

Use the following steps to build the application with nested controllers and scopes:

1. Add the lesson26/scope_events.html and lesson26/js/scope_ events.js files.
2. Add the code shown in Listing 26.3 and Listing 26.4 to the HTML and JavaScript files.
3. The following lines of code in Listing 26.3 implement the parent scope controller called `Characters`. An array of character names is defined in the scope and the `currentName` value is set to the first element in that array:

Click here to view code image

```
controller('Characters', function($scope) {
  $scope.names = ['Frodo', 'Aragorn', 'Legolas', 'Gimli'];
  $scope.currentName = $scope.names[0];
...
}).
```

4. The following `changeName()` function changes the `currentName` value in the scope and then broadcasts a `CharacterChanged` event. Notice that `changeName()` uses the `this` keyword to access the `name` property. The `name` property comes from a dynamic child scope that was created because

the following directives were used to generate multiple elements in Listing 26.4:

Click here to view code image

```
  $scope.changeName = function() {
    $scope.currentName = this.name;
    $scope.$broadcast('CharacterChanged', this.name);
  };
```

5. The following `CharacterDeleted` event is handled by the `$scope.$on('CharacterDeleted')` event handler and removes the character from the list, which will be reflected in the web page:

Click here to view code image

```
  $scope.changeName = function() {
  $scope.$on('CharacterDeleted', function(event, removeName){
    var i = $scope.names.indexOf(removeName);
    $scope.names.splice(i, 1);
    $scope.currentName = $scope.names[0];
    $scope.$broadcast('CharacterChanged', $scope.currentName);
  });
```

6. The following lines of code in Listing 26.3 define a child scope controller named `Character`. The `CharacterChanged` event is handled by the `$scope.$on('Character-Changed')` event handler and sets the `currentInfo` value in the scope, which will update the page elements. The `deleteChar()` function removes the character from the `Character` scope and uses `$emit` to send a `CharacterDeleted` event up to the parent so that it can be handled in the `Characters` controller:

Click here to view code image

```
controller('Character', function($scope) {
  $scope.info = {'Frodo': { weapon: 'Sting',
                            race: 'Hobbit'},
                 'Aragorn': { weapon: 'Sword',
                              race: 'Man'},
                 'Legolas': { weapon: 'Bow',
                              race: 'Elf'},
                 'Gimli': { weapon: 'Axe',
                            race: 'Dwarf'}};
  $scope.currentInfo = $scope.info['Frodo'];
  $scope.$on('CharacterChanged', function(event, newCharacter){
    $scope.currentInfo = $scope.info[newCharacter];
  });
  $scope.deleteChar = function() {
    delete $scope.info[$scope.currentName];
    $scope.$emit('CharacterDeleted', $scope.currentName);
  };
});
```

7. The AngularJS template code in Listing 26.4 implements the nested `ng-controller` statements, which generate the scope hierarchy and display scope values for the characters. This code also includes some very basic CSS styling to make spans look like buttons and to position elements on the page.

8. Load the scope_events.html file in a browser. Figure 26.2 shows the resulting web page. As you click a character name, information about that character is displayed, and when you click the Delete button, the character is deleted from the buttons and the Info section.

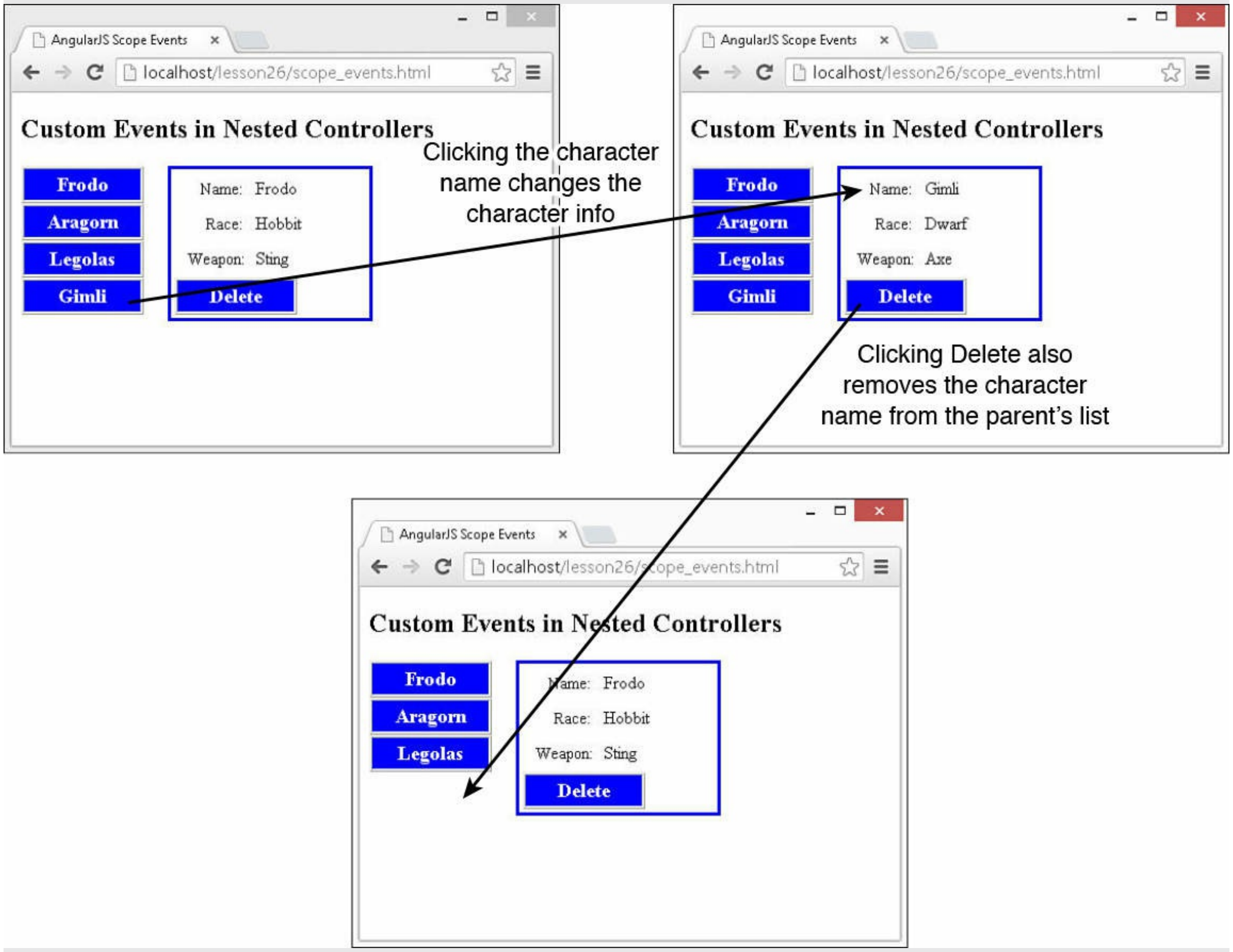

AngularJS Scope Events
localhost/lesson26/scope_events.html
Custom Events in Nested Controllers
Frodo
Aragorn
Legolas
Gimli
Name: Frodo
Race: Hobbit
Weapon: Sting
Delete
Clicking the character name changes the character info
Name: Gimli
Race: Dwarf
Weapon: Axe
Clicking Delete also removes the character name from the parent's list

FIGURE 26.2 Using `$broadcast()` and `$emit()` to send change and delete events through a scope hierarchy.

LISTING 26.3 scope_events.js Implementing `$emit()` and `$broadcast()` Events Within the Scope Hierarchy

Click here to view code image

```
angular.module('myApp', []).
  controller('Characters', function($scope) {
    $scope.names = ['Frodo', 'Aragorn', 'Legolas', 'Gimli'];
    $scope.currentName = $scope.names[0];
    $scope.changeName = function() {
      $scope.currentName = this.name;
      $scope.$broadcast('CharacterChanged', this.name);
    };
    $scope.$on('CharacterDeleted', function(event, removeName){
      var i = $scope.names.indexOf(removeName);
      $scope.names.splice(i, 1);
      $scope.currentName = $scope.names[0];
      $scope.$broadcast('CharacterChanged', $scope.currentName);
    });
  }).
  controller('Character', function($scope) {
    $scope.info = {'Frodo': { weapon: 'Sting',
                              race: 'Hobbit'},
                   'Aragorn': { weapon: 'Sword',
                                race: 'Man'},
                   'Legolas': { weapon: 'Bow',
                                race: 'Elf'},
                   'Gimli': { weapon: 'Axe',
                              race: 'Dwarf'}};
    $scope.currentInfo = $scope.info['Frodo'];
    $scope.$on('CharacterChanged', function(event, newCharacter){
      $scope.currentInfo = $scope.info[newCharacter];
    });
    $scope.deleteChar = function() {
      delete $scope.info[$scope.currentName];
      $scope.$emit('CharacterDeleted', $scope.currentName);
    };
  });
```

LISTING 26.4 scope_events.html HTML Template Code That Renders the Scope Hierarchy for Listing 26.3 Controllers

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Scope Events</title>
    <style>
      span{
        padding: 3px; border: 3px ridge;
        cursor: pointer; width: 100px; display: inline-block;
        font: bold 18px/22px Georgia; text-align: center;
        color: white; background-color: blue }
      label{
        padding: 2px; margin: 5px 10px; font: 15px bold;
        display: inline-block; width: 50px; text-align: right; }
      .lList {
        vertical-align: top;
        display: inline-block; width: 130px; }
      .cInfo {
        display: inline-block; width: 175px;
        border: 3px blue ridge; padding: 3px; }
    </style>
  </head>
  <body>
    <h2>Custom Events in Nested Controllers</h2>
    <div ng-controller="Characters">
      <div class="lList">
          <span ng-repeat="name in names"
                 ng-click="changeName()">{{name}}
            </span>
      </div>
      <div class="cInfo">
            <div ng-controller="Character">
              <label>Name: </label>{{currentName}}<br>
              <label>Race: </label>{{currentInfo.race}}<br>
              <label>Weapon: </label>{{currentInfo.weapon}}<br>
              <span ng-click="deleteChar()">Delete</span>
            </div>
      </div>
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
    <script src="js/scope_events.js"></script>
  </body>
</html>
```

Summary

The capability to manage events is one of the most critical components in most AngularJS applications. You can use events in AngularJS applications to provide user interaction with elements as well as components of the application communicating with each other to know when to perform certain tasks. This lesson started off with a brief discussion about browser and user interaction events and how they relate to the overall application architecture.

Next, the lesson covered using the `$watch`, `$watchGroup`, and `$watchCollection` methods to watch values in the scope. Using watches allows you to act on changes to the scope values without having to place code in every location where those values might change.

Scopes are organized into hierarchies, and the root scope is defined at that application level. Each instance of a controller also gets an instance of a child scope. In this lesson, you learned how to emit or broadcast events from within a scope and then implement handlers that listen for those events and get executed when they are triggered.

Q&A

Q. When do watches get processed by AngularJS?

A. AngularJS uses a digest cycle that is basically a loop that checks for changes to variables being watched by the scope.

Q. Does that mean that all my scope values are being watched?

A. No, that could be a terrible performance drain. Only the variables you have specified by the watch functions will be checked.

Workshop

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

Quiz

1. What is the best way to watch multiple values in the scope at the same time?
2. Do events triggered by `$emit` go upward or downward through the scope hierarchy?
3. Do events triggered by `$broadcast` go upward or downward through the scope hierarchy?
4. What is the function called that stops the default behavior in a browser event?

Quiz Answers

1. Use `$watchGroup`
2. Upward
3. Downward
4. `preventDefault()`

Exercises

1. Modify the code in Listings 26.1 and 26.2 and add a ratio value that represents the number of hits divided by the number of hits plus the number of misses. Modify the current watches to update the ratio each time hit or miss is pressed.
2. Modify the code in Listings 26.3 and 26.4. Add a Duplicate button to the web page next to the Delete button. When the user clicks the Duplicate button, create a second copy of that character in the characters list. You will also need to add an

event and handler to notify the parent controller that another character has been added and should show up in the character names list.

Lesson 27. Implementing AngularJS Services in Web Applications

What You'll Learn in This Lesson:

- How to send HTTP GET and PUT requests to the web server from AngularJS
- How to implement browser alerts
- How to implement timers in your AngularJS code
- How to manipulate cookies from AngularJS
- How to use the `$animate` service

One of the most fundamental components of AngularJS functionality is its built-in services. Services provide task-based functionality to your applications. Think about a service as a chunk of reusable code that performs one or more related tasks. AngularJS provides several built-in services and also enables you to create your own customized services.

This lesson introduces the AngularJS services. You will see and implement some of the built-in services, such as `$http` for web server communication, `$cookieStore` for storing and retrieving browser cookies, and `$animate` to provide animation capabilities.

Understanding AngularJS Services

AngularJS services are singleton objects, which means only one instance is ever created. The intent of a service is to provide a concise bit of code that performs specific tasks. A service can be as simple as providing a value definition or as complex as providing full HTTP communication to a web server.

A service provides a container for reusable functionality that is readily available to AngularJS applications. Services are defined and registered with the dependency injection mechanism in AngularJS. This enables you to inject services into modules, controllers, and other services.

By the Way

Lesson 21, "Understanding AngularJS Application Dynamics," discusses dependency injection. You should read that lesson, if you haven't already, before continuing with this one.

Using the Built-In Services

AngularJS provides several built-in services. These are automatically registered with the dependency injector; therefore, you can easily incorporate them into your AngularJS applications by using dependency injection.

Table 27.1 describes some of the most common built-in services to give you an idea of what is available. The following sections cover some of these services in more detail.

Service	Description
`$anchorScroll`	Provides the capability to scroll to a page anchor specified in `$location.hash()` based on the rules defined in the HTML5 spec.
`$animate`	Provides animation hooks to link into both CSS- and JavaScript-based animations.
`$cacheFactory`	Provides the capability to put key/value pairs into an object cache, where they can be retrieved later by other code components using the same service.
`$compile`	Provides the capability to compile an HTML string or a DOM object into a template and produce a template function that can link the scope and template together.
`$cookies`	Provides read and write access to the browser's cookies.
`$document`	Specifies a jQuery-wrapped reference to the browser's `window.document` element.
`$exceptionHandler`	Specifies a handler to which uncaught exceptions in AngularJS expressions are delegated.
`$http`	Provides a simple-to-use functionality to send HTTP requests to the web server or another service.

`$ingerpolate`	Compiles a string with markup into an interpolation function.
`$interval`	Provides access to a browser's `window.setInterval` functionality.
`$locale`	Provides localization rules that are consumed by various AngularJS components.
`$location`	Provides the capability to interact with a browser's `window.location` object.
`$log`	Provides a simple logging service.
`$parse`	Parses an AngularJS expression string into a JavaScript function.
`$q`	Provides a promise/deferred implementation service.
`$resource`	Enables you to create an object that can interact with a RESTful server-side data source.
`$rootElement`	Provides access to the root element in the AngularJS application.
`$rootScope`	Provides access to the root scope for the AngularJS application.
`$route`	Provides deep-linking URL support for controllers and views by watching the `$location.url()` and mapping the path to existing route definitions.
`$routeParams`	Provides a service that enables you to retrieve the current set of parameters in the route.
`$sanitize`	Provides a service that can be used to sanitize input by parsing the HTML into tokens.
`$sce`	Provides strict contextual escaping functionality when handling data from untrusted sources.
`$swipe`	Provides a service that makes implementing device swipe types of directives easier.
`$templateCache`	Provides the capability to read templates from a web server into a cache for later use.
`$timeout`	Provides access to a browser's `window.setTimeout` functionality.
`$window`	Specifies a jQuery-wrapped reference to the browser's `window` element.

TABLE 27.1 Common Services That Are Built In to AngularJS

Sending HTTP GET and PUT Requests with the $http Service

The `$http` service enables you to directly interact with the web server from your AngularJS code. The `$http` service uses the browser's `XMLHttpRequest` object underneath, but from the context of the AngularJS framework.

There are two ways to use the `$http` service. The simplest is to use one of the following built-in shortcut methods that correspond to standard HTTP requests:

- `delete(url, [config])`
- `get(url, [config])`
- `head(url, [config])`
- `jsonp(url, [config])`
- `post(url, data, [config])`
- `put(url, data, [config])`
- `patch(url, data, [config])`

Configuring the `$http` Request

In these methods, the `url` parameter is the URL of the web request. The optional `config` parameter is a JavaScript object that specifies the options to use when implementing the request. Table 27.2 lists the properties you can set in the `config` parameter.

Property	Description
`method`	An HTTP method, such as `GET` or `POST`.
`url`	The URL of the resource that is being requested.
`params`	Parameters to be sent. This can be a string in the format `?key1=value1&key2=value2&...` or it can be an object, in which case it is turned into a JSON string.
`data`	Data to be sent as the request message data. By default, data is posted to the server as JSON.
`headers`	Headers to send with the request. You can specify an object containing the header names to be sent as properties. If a property in the object has a null value, the header is not sent.
`xsrfHeaderName`	The name of the HTTP header to populate with the `XSRF` token.
`xsrfCookieName`	The name of the cookie containing the `XSRF` token.
`transformRequest`	A function that is called to transform/serialize the request headers and body. The function accepts the body data as the first parameter and a getter function to get the headers by name as the second. For example: `function(data, getHeader)`

`transformResponse`	A function that is called to transform/deserialize the response headers and body. The function accepts the body data as the first parameter and a getter function to get the headers by name as the second. For example: `function(data, getHeader)`
`cache`	A Boolean that, when `true`, indicates that a default `$http` cache is used to cache `GET` responses; otherwise, if a cache instance is built with `$cacheFactory`, that cache is used for caching. If `false` and there is no `$cacheFactory` built, the responses are not cached.
`timeout`	The timeout, in milliseconds, when the request should be aborted.
`withCredentials`	A Boolean that, when `true`, indicates that the `withCredentials` flag on the `XHR` object is set.
`responseType`	The type of response to expect, such as `json` or `text`.

TABLE 27.2 Properties That Can Be Defined in the config Parameter for $http Service Requests

You can also specify the request, URL, and data by sending the `config` parameter directly to the `$http(config)` method. For example, the following two lines are the same:

Click here to view code image

```
$http.get('/myUrl');
$http({method: 'GET', url:'/myUrl'});
```

Implementing the $http Response Callback Functions

When you call a request method by using the `$http` object, you get back an object with the promise methods `success()` and `error()`. You can pass to these methods a callback function that is called if the request is successful or if it fails. These methods accept the following parameters:

- **data:** Response data.
- **status:** Response status.
- **header:** Response header.
- **config:** Request configuration.

The following is a simple example of implementing the `success()` and `error()` methods on a `get()` request:

Click here to view code image

```
$http({method: 'GET', url: '/myUrl'}).
  success(function(data, status, headers, config) {
    // handle success
  }).
  error(function(data, status, headers, config) {
    // handle failure
  });
```

Try it Yourself: Implementing a Simple HTTP Server and Using the `$http` Service to Access It

In this example, you implement an AngularJS application that uses the `$http` service to access and update data on a web server. The purpose of this exercise is to familiarize you with using the `$http.get()` and `$http.post()` requests.

The code in Listings 27.1 through 27.3 implements a simple Node.js web service and AngularJS application that accesses it. The web server contains a simple JavaScript object with items and count to mimic the stock of a store. The web application enables a user to tell the server to restock the store, and buy and use items. The example is very rudimentary so that the code is easy to follow, but it incorporates `GET` and `POST` requests as well as error-handling examples.

Use the following steps to build the custom webserver with `POST` and `GET` request handlers and build an AngularJS application that accesses the server:

1. Add the lesson27/service_http.html and lesson28/js/service_http.js files.
2. Add the code shown in Listing 27.2 and Listing 27.3 to the HTML and JavaScript files.
3. For this example, we need to have the web server provide a simulated database access service. Listing 27.5 implements the Node.js web server that handles the following `GET` and `POST` routes to get and set a data on the server:
   - **/reset/data:** A `GET` route that reinitializes the items available in the store.
   - **/buy/item:** A `POST` route that decrements the item count and returns the store data (typically it would not need to return the full data, but for this simple example it makes the code cleaner). If the item is out of stock, a 400 error is returned.
   - **/set/user:** A `POST` route that accepts a user object in the body of the request and updates the object on the server to simulate storing a user object.
   - **/set/data:** A `POST` route that accepts an array of objects in the body of the request and updates the data variable to simulate storing database data. Typically, you would never store all the table data at once, but for

simplicity of this example, this is how it is.

You don’t necessarily need to pay a lot of attention to the code in Listing 27.1 other than understanding the routes that it provides so that you can follow the interactions in the AngularJS application defined in Listing 27.2. The server is very rudimentary and doesn’t handle errors but is enough to illustrate using `$http` in AngularJS.

By the Way

You will need to stop the normal `server.js` HTTP server if it is running before starting `service_db_server.js` from Listing 27.1 (don’t forget to stop `service_server.js` and reload `server.js` when you are done with this exercise). Also, you will want to place the `service_server.js` file from Listing 27.1 in the parent folder to the `service_http.html` in Listing 27.3 for the paths to match up properly in the Node.js static routes. The structure should look similar to this:

Click here to view code image

```
./service_server.js
./lesson28/service_http.html
./lesson28/js/service_http.js
```

LISTING 27.1 service_server.js Implementing an Express Server That Supports GET and POST Routes for an AngularJS Controller

Click here to view code image

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use('/', express.static('./'));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
function initStore(){
  var items = ['eggs', 'toast', 'bacon', 'juice'];
  var storeObj = {};
  for (var itemIDX in items){
    storeObj[items[itemIDX]] =
      Math.floor(Math.random() * 5 + 1);
  }
  return storeObj;
}
var storeItems = initStore();
app.get('/reset/data', function(req, res){
  storeItems = initStore();
  res.json(storeItems);
});
app.post('/buy/item', function(req, res){
  if (storeItems[req.body.item] > 0){
    storeItems[req.body.item] =
      storeItems[req.body.item] - 1;
    res.json(storeItems);
  }else {
    res.json(400, { msg: 'Sorry ' + req.body.item +
                         ' is out of stock.' });
  }
});
app.listen(80);
```

4. Add the code in Listing 27.2 that implements the AngularJS application and controller. Notice that the `buyItem()` method calls the `/buy/item POST` route on the server and places the results in the scope variable `$scope.storeItems`. If an error occurs, the `$scope.status` variable is set to the `msg` value in the error response object. The `resetStore()` method calls the `/reset/data GET` route on the server and updates `$scope.storeItems` with the successful response.

LISTING 27.2 service_http.js Implementing an AngularJS Controller That Interacts with the Web Server Using the `$http` Service

Click here to view code image

```
angular.module('myApp', []).
  controller('myController', ['$scope', '$http',
                              function($scope, $http) {
    $scope.storeItems = {};
    $scope.kitchenItems = {};
    $scope.status = "";
    $scope.resetStore = function(){
      $scope.status = "";
      $http.get('/reset/data')
           .success(function(data, status, headers, config) {
              $scope.storeItems = data;
            })
           .error(function(data, status, headers, config) {
              $scope.status = data;
            });
    };
    $scope.buyItem = function(buyItem){
      $http.post('/buy/item', {item:buyItem})
           .success(function(data, status, headers, config) {
              $scope.storeItems = data;
              if($scope.kitchenItems.hasOwnProperty(buyItem)){
                $scope.kitchenItems[buyItem] += 1;
              } else {
                $scope.kitchenItems[buyItem] = 1;
              }
              $scope.status = "Purchased " + buyItem;
            })
           .error(function(data, status, headers, config) {
              $scope.status = data.msg;
            });
    };
    $scope.useItem = function(useItem){
      if($scope.kitchenItems[useItem] > 0){
        $scope.kitchenItems[useItem] -= 1;
      }
    };
  }]);
```

5. Add the code in Listing 27.3 that implements an AngularJS template that includes the `Restock Store` button, `status` message on error, and a list of store items.

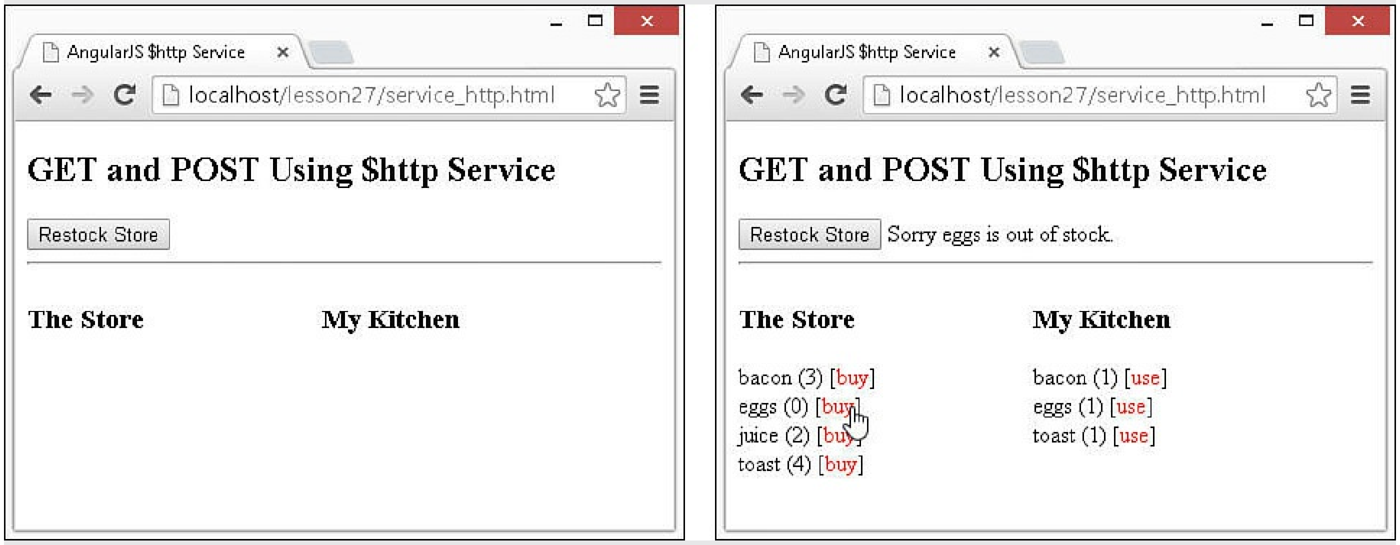
LISTING 27.3 service_http.html An AngularJS Template That Implements Directives That Are Linked to Web Server Data

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS $http Service</title>
  <style>
    span {
      color:red; cursor: pointer; }
    .myList {
      display: inline-block; width: 200px;
      vertical-align: top; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h2>GET and POST Using $http Service</h2>
    <input type="button" ng-click="resetStore()"
           value="Restock Store"/>
    {{status}}
    <hr>
    <div class="myList">
         <h3>The Store</h3>
         <div ng-repeat="(item, count) in storeItems">
           {{item}} ({{count}})
           [<span ng-click="buyItem(item)">buy</span>]
         </div>
    </div>
    <div class="myList">
```

```
          <h3>My Kitchen</h3>
          <div ng-repeat="(item, count) in kitchenItems">
            {{item}} ({{count}})
            [<span ng-click="useItem(item)">use</span>]
          </div>
        </div>
      </div>
      <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
      <script src="js/service_http.js"></script>
    </body>
    </html>
```

6. Open the service_http.html from Listing 27.3 in a browser. You should be able to restock the store, buy and use items, and see the count change accordingly. Figure 27.1 shows how the item counts get adjusted when items are bought and used, and the out-of-stock error message is shown when the user tries to buy an out-of-stock item.


AngularJS $http Service
localhost/lesson27/service_http.html

GET and POST Using $http Service

Restock Store

The Store   My Kitchen

AngularJS $http Service
localhost/lesson27/service_http.html

GET and POST Using $http Service

Restock Store Sorry eggs is out of stock.

The Store
bacon (3) [buy]
eggs (0) [buy]
juice (2) [buy]
toast (4) [buy]

My Kitchen
bacon (1) [use]
eggs (1) [use]
toast (1) [use]

FIGURE 27.1 Implementing the `$http` service to allow AngularJS controllers to interact with the web server.

Using the **$cacheFactory** Service

The `$cacheFactory` service provides a very handy repository for temporarily storing data as key/value pairs. Because `$cacheFactory` is a service, it is available to multiple controllers and other AngularJS components.

When creating the `$cacheFactory` service, you can specify an `options` object that contains the `capacity` property—for example, `{capacity: 5}`. By adding this `capacity` setting, you limit the maximum number of elements in the cache to five. When a new item is added, the oldest item is removed. If no capacity is specified, the cache continues to grow.

Listing 27.4 illustrates a basic example of implementing `$cacheFactory` in a `Module` object and then accessing it from two different controllers. The `factory()` method creates and injectable `$cacheFactory` service object that is instantiated and used in the `myController` and `myController2` controllers.

LISTING 27.4 service_cache.js Implementing a $cacheFactory Service in an AngularJS Application

Click here to view code image

```
var app = angular.module('myApp', []);
app.factory('MyCache', function($cacheFactory) {
  return $cacheFactory('myCache', {capacity:5});
});
app.controller('myController', ['$scope', 'MyCache',
                                function($scope, cache) {
    cache.put('myValue', 55);
  }]);
app.controller('myController2', ['$scope', 'MyCache',
                                 function($scope, cache) {
  $scope.value = cache.get('myValue');
}]);
```

Implementing Browser Alerts Using the $window Service

The `$window` service provides a jQuery wrapper for a browser's `window` object, allowing you to access the `window` object as you normally would from JavaScript. To illustrate this, the following code pops up a browser alert, using the `alert()` method on the `window` object. The message of the alert gets data from the `$window.screen.availWidth` and `$window.screen.availHeight` properties of the browser's `window` object:

Click here to view code image

```
var app = angular.module('myApp', []);
app.controller('myController', ['$scope', '$window',
                                function($scope, window) {
    window.alert("Your Screen is: \n" +
        window.screen.availWidth + "X" + window.screen.availHeight);
  }]);
```

Interacting with Browser Cookies Using the $cookieStore Service

AngularJS provides a couple of services for getting and setting cookies: `$cookie` and `$cookieStore`. Cookies provide temporary storage in a browser and persist even when the user leaves the web page or closes the browser.

The $cookie service enables you to get and change string cookie values by using dot notation. For example, the following code retrieves the value of a cookie with the name appCookie and then changes it:

Click here to view code image

```
var cookie = $cookies.appCookie;
$cookies.appCookie = 'New Value';
```

The $cookieStore service provides get(), put(), and remove() functions to get, set, and remove cookies. A nice feature of the $cookieStore service is that it serializes JavaScript object values to a JSON string before setting them, and then it deserializes them back to objects when getting them.

To use the $cookie and $cookieStore services, you need to do three things. First, you load the angular-cookies.js library in the template after angular.js but before application.js. For example:

Click here to view code image

```
<script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
<script src="http://code.angularjs.org/1.3.0/angular-cookies.min.js">
</script>
```

By the Way

You can also download the angular-cookies.js file from the AngularJS website at http://code.angularjs.org/<*version*>/, where <*version*> is the version of AngularJS that you are using. You might need to download the angular-cookies.min.js.map file as well, depending on which version of AngularJS you are using.

Second, you add ngCookies to the required list in your application Module definition. For example:

Click here to view code image

```
var app = angular.module('myApp', ['ngCookies']);
```

Third, you inject the $cookies or $cookieStore service into your controller. For example:

Click here to view code image

```
app.controller('myController', ['$scope', '$cookieStore',
                                function($scope, cookieStore) {
}]);
```

Try it Yourself: Using the $cookieStore Service

In this example, you use the `$cookie` service to set and clear cookies in the browser from your AngularJS application. Listing 27.5 loads `ngCookies` in the application, injects `$cookieStore` into the controller, and then uses the `get()`, `put()`, and `remove()` methods to interact with a cookie named `myAppCookie`.

Listing 27.6 implements a set of radio buttons that tie to the `favCookie` value in the model and use `ng-change` to call `setCookie()` when the values of the buttons change.

Use the following steps to build the AngularJS application that implements cookies:

1. Add the lesson27/service_cookie.html and lesson27/js/service_cookie.js files.
2. Add the code shown in Listing 27.5 and Listing 27.6 to the HTML and JavaScript files.
3. The following lines of code from Listing 27.5 implement the controller that is injected with the `$cookieStore` service:

Click here to view code image

```
app.controller('myController', ['$scope', '$cookieStore',
                                function($scope, cookieStore) {
```

4. The following lines of code implement the `setCookie()` function that calls `cookieStore.remove()` to remove the cookie from the browser and `cookieStore.put()` to set the cookie in the browser. Notice that the name of the cookie is `myAppCookie` and the value for `flavor` comes from the `favCookie` value in the scope:

Click here to view code image

```
    $scope.setCookie = function(){
      if ($scope.favCookie === 'None'){
        cookieStore.remove('myAppCookie');
      }else{
        cookieStore.put('myAppCookie', {flavor:$scope.favCookie});
      }
      $scope.myFavCookie = cookieStore.get('myAppCookie');
    };
```

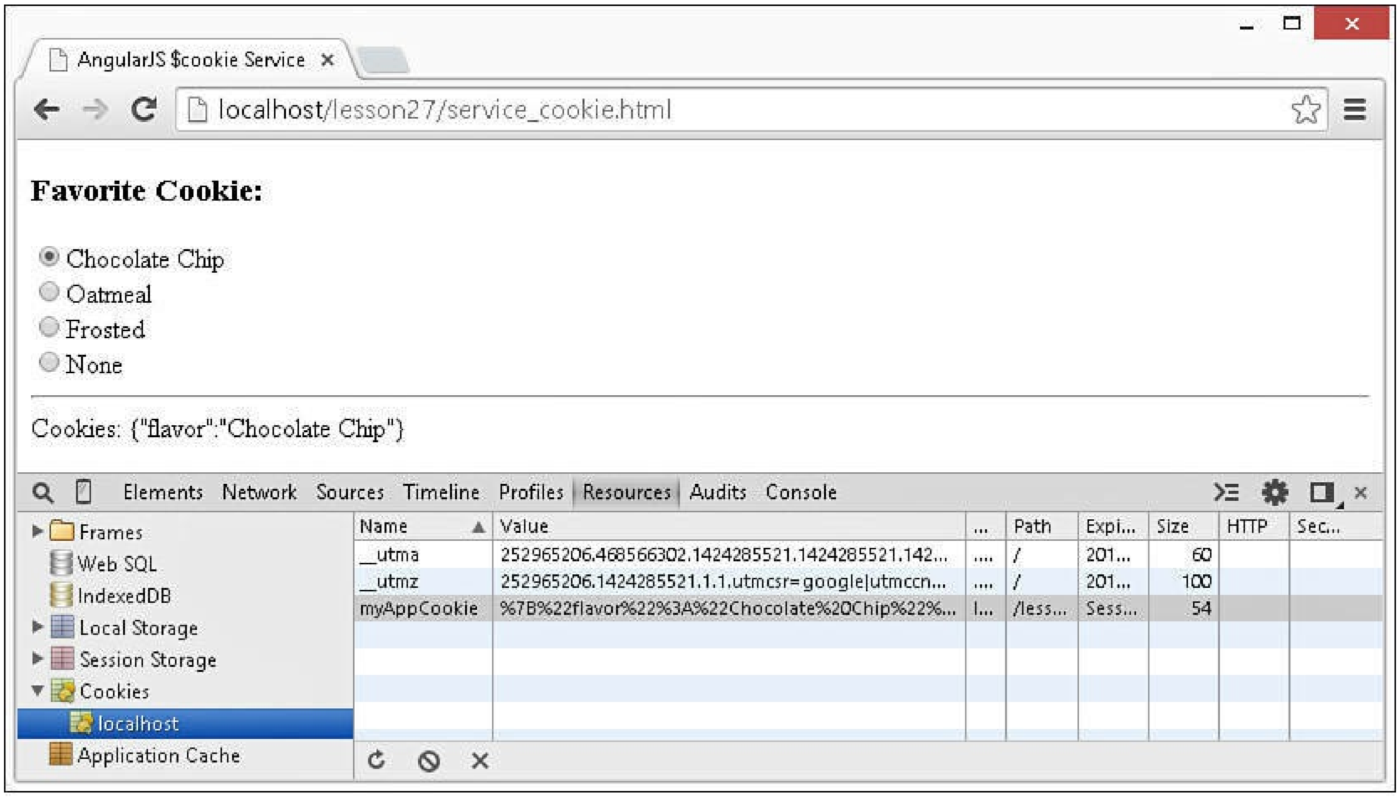
5. The following lines of code implement the `initCookieValue()` function that uses `cookieStore.get()` to retrieve the value of the cookie if it is already stored in the browser and then sets the `favCookie` and `myFavCookie` value accordingly:

Click here to view code image

```
    $scope.initCookieValue = function(){
      var cookie = cookieStore.get('myAppCookie');
      if (cookie){
        $scope.favCookie = cookie.flavor;
      } else {
        $scope.favCookie = 'None';
      }
      $scope.myFavCookie = $scope.favCookie;
    };
    $scope.initCookieValue();
```

6. The code in Listing 27.6 implements the AngularJS template with a radio button set that has its value bound to the `favCookie` using the `ng-model="favCookie"` directive.

7. Open the server_cookie.html file in a web browser to set and clear the cookies. In Chrome, you can use the developer tools Resources, Cookies, localhost to view the value of `myAppCookie`, as shown in Figure 27.2. To refresh the value of the cookies in developer tools, click the Cookies menu and then click the localhost menu again.

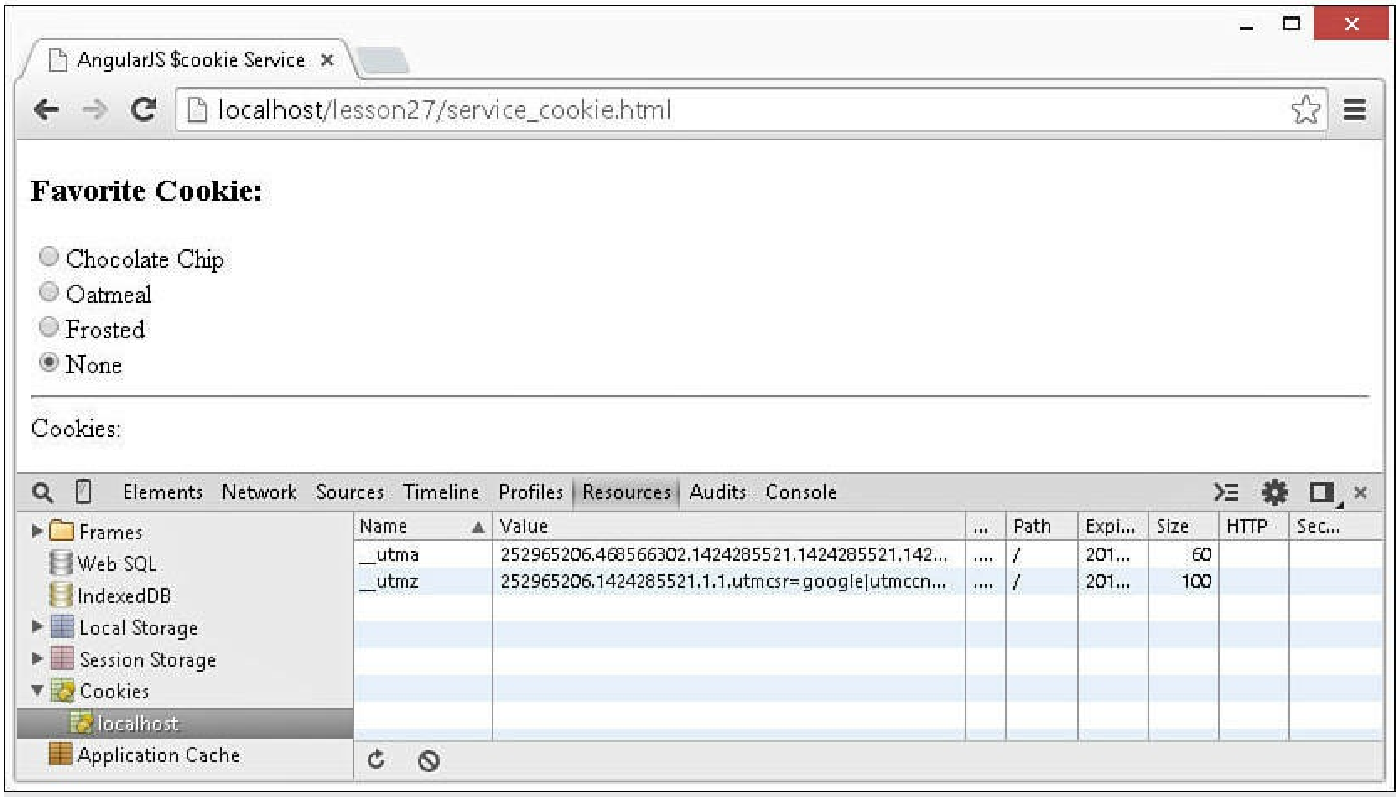

AngularJS $cookie Service
localhost/lesson27/service_cookie.html
Elements Network Sources Timeline Profiles Resources Audits Console
Frames
Web SQL
IndexedDB
Local Storage
Session Storage
Cookies
localhost
Application Cache

Favorite Cookie:

- [✓ Chocolate Chip]
- Oatmeal
- Frosted
- None

Cookies: {"flavor":"Chocolate Chip"}

Name	Value	...	Path	Expi...	Size	HTTP	Sec...
__utma	252965206.468566302.1424285521.1424285521.142...	....	/	201...	60		
__utmz	252965206.1424285521.1.1.utmcsr=google|utmccn...	....	/	201...	100		
myAppCookie	%7B%22flavor%22%3A%22Chocolate%20Chip%22%...	l...	/less...	Sess...	54		


AngularJS $cookie Service
localhost/lesson27/service_cookie.html
Elements Network Sources Timeline Profiles Resources Audits Console
Frames
Web SQL
IndexedDB
Local Storage
Session Storage
Cookies
localhost
Application Cache

Favorite Cookie:

- Chocolate Chip
- Oatmeal
- Frosted
- [✓ None]

Cookies:

Name	Value	...	Path	Expi...	Size	HTTP	Sec...
__utma	252965206.468566302.1424285521.1424285521.142...	....	/	201...	60		
__utmz	252965206.1424285521.1.1.utmcsr=google|utmccn...	....	/	201...	100		

FIGURE 27.2 Implementing the `$cookieStore` service to allow AngularJS controllers to interact with the browser cookies.

LISTING 27.5 service_cookie.js Implementing an AngularJS Controller That

Interacts with Browser Cookies by Using the $cookieStore Service

Click here to view code image

```
var app = angular.module('myApp', ['ngCookies']);
app.controller('myController', ['$scope', '$cookieStore',
                                function($scope, cookieStore) {
    $scope.setCookie = function(){
      if ($scope.favCookie === 'None'){
        cookieStore.remove('myAppCookie');
      }else{
        cookieStore.put('myAppCookie', {flavor:$scope.favCookie});
      }
      $scope.myFavCookie = cookieStore.get('myAppCookie');
    };
    $scope.initCookieValue = function(){
      var cookie = cookieStore.get('myAppCookie');
      if (cookie){
        $scope.favCookie = cookie.flavor;
      } else {
        $scope.favCookie = 'None';
      }
      $scope.myFavCookie = $scope.favCookie;
    };
    $scope.initCookieValue();
  }]);
```

LISTING 27.6 service_cookie.html An AngularJS Template That Implements Radio Buttons to Set a Cookie Value

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS $cookie Service</title>
</head>
<body>
  <div ng-controller="myController">
    <h3>Favorite Cookie:</h3>
    <input type="radio" value="Chocolate Chip" ng-model="favCookie"
           ng-change="setCookie()">Chocolate Chip</input><br>
    <input type="radio" value="Oatmeal" ng-model="favCookie"
           ng-change="setCookie()">Oatmeal</input><br>
    <input type="radio" value="Frosted" ng-model="favCookie"
           ng-change="setCookie()">Frosted</input><br>
    <input type="radio" value="None" ng-model="favCookie"
           ng-change="setCookie()">None</input>
    <hr>Cookies: {{myFavCookie}}
  </div>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
  <script src="http://code.angularjs.org/1.3.0/angular-cookies.min.js">
</script>
  <script src="js/service_cookie.js"></script>
</body>
</html>
```

Implementing Timers with **$interval** and **$timeout** Services

The AngularJS `$interval` and `$timeout` services enable you to delay execution of code for an amount of time. These services interact with the JavaScript `setInterval` and `setTimeout` functionality, but within the AngularJS framework.

The `$interval` and `$timeout` services use the following syntax:

Click here to view code image

```
$interval(callback, delay, [count], [invokeApply]);
$timeout(callback, delay, [invokeApply]);
```

The parameters are described here:

- **callback:** Is executed when the delay has expired.
- **delay:** Specifies the number of milliseconds to wait before the callback function is executed.
- **count:** Indicates the number of times to repeat the interval.
- **invokeApply:** Is a Boolean that, if `true`, causes the function to execute only in the `$apply()` block of the AngularJS event cycle. The default is `true`.

When you call the `$interval()` and `$timeout()` methods, they return a promise object that you can use to cancel the timeout or interval. To cancel an existing `$timeout` or `$interval`, call the `cancel()` method. For example:

Click here to view code image

```
var myInterval = $interval(function(){$scope.seconds++;}, 1000, 10, true);
. . .
$interval.cancel(myInterval);
```

If you create timeouts or intervals by using `$timeout` or `$interval`, you must explicitly destroy them by using `cancel()` when the `scope` or `elements` directives are destroyed. The easiest way to do this is by adding a listener to the `$destroy` event. For example:

Click here to view code image

```
$scope.$on('$destroy', function(){
  $scope.cancel(myInterval);
});
```

Using the **$animate** Service

The `$animate` service provides animation detection hooks you can use when performing enter, leave, and move DOM operations, as well as `addClass` and `removeClass` operations. You can use these hooks either through CSS class names or through the `$animate` service in AngularJS.

To implement animation, you need to add a directive that supports animation to the element that you want to animate. Table 27.3 lists the directives that support animation and the types of animation events that they support.

Directive	Description
`ngRepeat`	Supports `enter`, `leave`, and `move` events.
`ngView`	Supports `enter` and `leave` events.
`ngInclude`	Supports `enter` and `leave` events.
`ngSwitch`	Supports `enter` and `leave` events.
`ngIf`	Supports `enter` and `leave` events.
`ngClass`	Supports `addClass` and `removeClass` events.
`ngShow`	Supports `addClass` and `removeClass` events.
`ngHide`	Supports `addClass` and `removeClass` events.

TABLE 27.3 AngularJS Directives That Support Animation

Implementing Animation in CSS

To implement animation in CSS, you need to include the `ngClass` directive in the element that you want to animate. AngularJS uses the `ngClass` value as a root name for additional CSS classes that will be added to and removed from the element during animation.

An animation event is called on an element with an `ngClass` directive defined. Table 27.4 lists the additional classes that are added and removed during the animation process.

Class	Description
`ng-animate`	Added when an event is triggered.
`ng-animate-active`	Added when animation starts and triggers CSS transitions.
`<super>-ng-move`	Added when `move` events are triggered.
`<super>-ng-move-active`	Added when move animation starts and triggers CSS transitions.
`<super>-ng-leave`	Added when `leave` events are triggered.
`<super>-ng-leave-active`	Added when leave animation starts and triggers CSS transitions.
`<super>-ng-enter`	Added when `enter` events are triggered.
`<super>-ng-enter-active`	Added when enter animation starts and triggers CSS transitions.
`<super>-ng-add`	Added when `addClass` events are triggered.
`<super>-ng-add-active`	Added when add class animation starts and triggers CSS transitions.
`<super>-ng-remove`	Added when `removeClass` events are triggered.
`<super>-ng-remove-active`	Added when remove class animation starts and triggers CSS transitions.

TABLE 27.4 AngularJS Directives That Are Automatically Added and Removed During Animation

To implement CSS-based animations, you need to add the appropriate CSS transition code for the additional classes listed in Table 27.4. To illustrate this, the following snippet implements add class and remove class transitions for a user-defined class named `.img-fade` that animates changing the `opacity` of the image to `.1` for a 2-second duration:

Click here to view code image

```
.img-fade-add, .img-fade-remove {
  -webkit-transition:all ease 2s;
  -moz-transition:all ease 2s;
  -o-transition:all ease 2s;
  transition:all ease 2s;
}
.img-fade, .img-fade-add.img-fade-add-active {
   opacity:.1;
}
```

Notice that the transitions are added to the `.img-fade-add` and `.img-fade-remove` classes, but the actual class definition is applied to `.img-fade`. You also need the class definition `.img-fade-add.img-fade-add-active` to set the ending state for the transition.

Implementing Animation in JavaScript

Implementing AngularJS CSS animation is very simple, but you can also implement animation in JavaScript using jQuery. JavaScript animations provide more direct control over your animations. Also, JavaScript animations do not require a browser to support CSS3.

To implement animation in JavaScript, you need to include the jQuery library in your template before the `angular.js` library is loaded. For example:

Click here to view code image

```
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
```

By the Way

Including the full jQuery library is necessary if you want to be able to utilize the full features of jQuery animation. If you decide to include the jQuery library, make certain that it is loaded before the AngularJS library in your HTML code.

You also need to include the `ngAnimate` dependency in your application `Module` object definition. For example:

Click here to view code image

```
var app = angular.module('myApp', ['ngAnimate']);
```

You can then use the `animate()` method on your `Module` object to implement animations. The `animate()` method returns an object that provides functions for the `enter`, `leave`, `move`, `addClass`, and `removeClass` events that you want to handle. These functions are passed the element to be animated as the first parameter. You can then use the jQuery `animate()` method to animate an element.

The jQuery `animate()` method uses the following syntax, in which cssProperties is an object of CSS attribute changes, duration is specified in milliseconds, easing is the easing method, and callback is the function to execute when the animation completes:

Click here to view code image

```
animate(cssProperties, [duration], [easing], [callback])
```

For example, the following code animates adding the `fadeClass` class to an element by setting `opacity` to `0`:

Click here to view code image

```
app.animation('.fadeClass', function() {
  return {
    addClass : function(element, className, done) {
      jQuery(element).animate({ opacity: 0}, 3000);
    },
  };
});
```

Try it Yourself: Animating Elements Using AngularJS

In this exercise, you get a chance to implement animation using AngularJS with CSS as well as jQuery. The purpose of this exercise is to show you how each method works.

Listings 27.7, 27.8, and 27.9 implement a basic animation example that applies a fade-in/out animation to an image, using the JavaScript method, and uses CSS transition animation to animate resizing the image.

Use the following steps to build the AngularJS application that applies the animations:

1. Add the lesson27/service_animation.html, lesson27/js/service_animation.js, and lesson27/js/service_animation.css files.
2. Add the code shown in Listing 27.7, Listing 27.8, and Listing 27.9 to the HTML, JavaScript, and CSS files.
3. The following line of code in Listing 27.7 injects the AngularJS animation service into the application so that it can be used:

Click here to view code image

```
var app = angular.module('myApp', ['ngAnimate']);
```

4. The following lines of code from Listing 27.7 implement the fade-in and fade-out animations using the `animation()` method on the application. Notice that the same class `.fadeOut` is used to apply both the fade-in and the fade-out animations by hooking into the `addClass` and `removeClass` events and then using a `jQuery.animate()` method on the opacity of the element:

Click here to view code image

```
app.animation('.fadeOut', function() {
  return {
    enter : function(element, parentElement, afterElement, doneCallback) {},
    leave : function(element, doneCallback) {},
    move : function(element, parentElement, afterElement, doneCallback) {},
    addClass : function(element, className, done) {
      jQuery(element).animate({ opacity: 0}, 3000);
    },
  removeClass : function(element, className, done) {
   jQuery(element).animate({ opacity: 1}, 3000);
  }
 };
```

5. The following lines of CSS code from Listing 27.9 implement the animation of shrinking and growing the image by applying CSS animation to the AngularJS class transitions of `-add` and `-add-active` when the `shrink` and `grow` classes are assigned to the element:

Click here to view code image

```
.shrink-add, .grow-add {
 -webkit-transition:all ease 2.5s;
 -moz-transition:all ease 2.5s;
 -o-transition:all ease 2.5s;
 transition:all ease 2.5s;
}
.shrink,
.shrink-add.shrink-add-active {
 width:100px;
}
.start-class,
.grow,
.grow-add.grow-add-active {
 width:400px;
}
```

6. Listing 27.8 implements the AngularJS template that supports the animation. Notice that line 5 loads the jQuery library to support the JavaScript animation code. Also, line 6 loads the `animate.css` script that contains the transition animations shown in Listing 27.9. The buttons add and remove the appropriate classes to initiate the animations.

7. Load the service_animate.html file in a browser and use the buttons to see the animations, as shown in Figure 27.3.

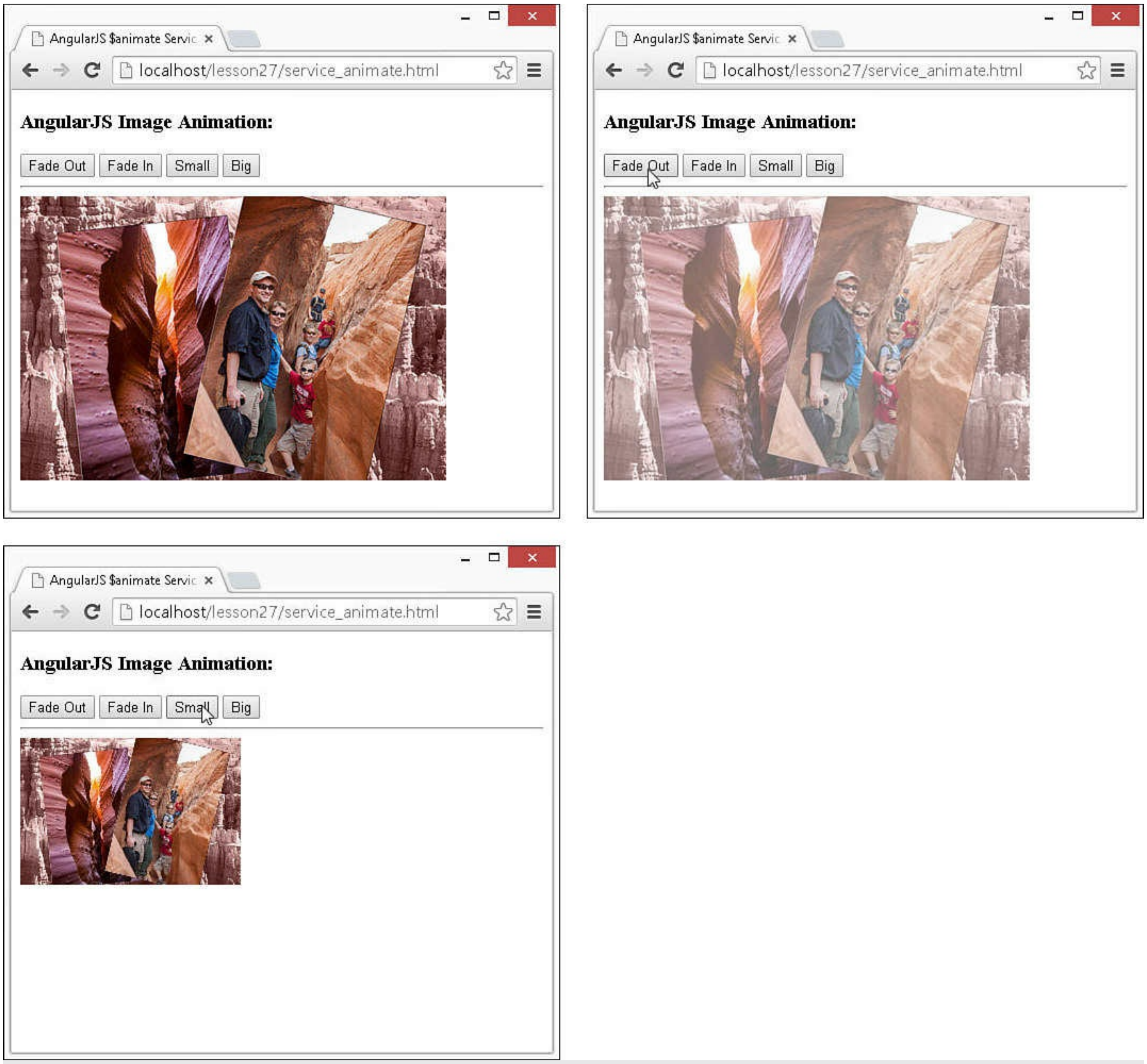

AngularJS $animate Servic
localhost/lesson27/service_animate.html
AngularJS Image Animation:
Fade Out
Fade In
Small
Big

FIGURE 27.3 Implementing the `$animation` service in both CSS and JavaScript to animate fading and resizing an image.

LISTING 27.7 service_animate.js Implementing an AngularJS Controller That Implements jQuery Animation Using the `$animation` Service

[Click here to view code image]()

```
var app = angular.module('myApp', ['ngAnimate']);
app.controller('myController', function($scope ) {
  $scope.myImgClass = 'start-class';
  });
app.animation('.fadeOut', function() {
  return {
    enter : function(element, parentElement, afterElement,
doneCallback) {},
    leave : function(element, doneCallback) {},
    move : function(element, parentElement, afterElement, doneCallback)
{},
    addClass : function(element, className, done) {
      jQuery(element).animate({ opacity: 0}, 3000);
    },
    removeClass : function(element, className, done) {
      jQuery(element).animate({ opacity: 1}, 3000);
    }
  };
});
```

LISTING 27.8 service_animate.html An AngularJS Template That Implements Buttons That Change the Class on an Image to Animate Fading and Resizing

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS $animate Service</title>
  <link rel="stylesheet" href="css/animate.css">
</head>
<body>
  <div ng-controller="myController">
    <h3>AngularJS Image Animation:</h3>
    <input type="button"
           ng-click="myImgClass='fadeOut'" value="Fade Out"/>
    <input type="button"
           ng-click="myImgClass=''" value="Fade In"/>
    <input type="button"
           ng-click="myImgClass='shrink'" value="Small"/>
    <input type="button"
           ng-click="myImgClass='grow'" value="Big"/>
    <hr>
    <img ng-class="myImgClass" src="/images/canyon.jpg" />
  </div>
  <script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
  <script src="http://code.angularjs.org/1.3.0/angular-animate.min.js">
</script>
  <script src="js/service_animate.js"></script>
</body>
</html>
```

By the Way

If you are loading AngularJS from your own web server, you can also download the `angular-animate.js` file from the AngularJS website at http://code.angularjs.org/<*version*>/, where <*version*> is the version of AngularJS that you are using. You might need to download the `angular-animate.min.js.map` file as well, depending on which version of AngularJS you are using.

LISTING 27.9 animate.css CSS Code That Provides Transition Effects for the Various Class Stages of the AngularJS Animation Code

Click here to view code image

```
.shrink-add, .grow-add {
  -webkit-transition:all ease 2.5s;
  -moz-transition:all ease 2.5s;
  -o-transition:all ease 2.5s;
  transition:all ease 2.5s;
}
.shrink,
.shrink-add.shrink-add-active {
  width:100px;
}
.start-class,
.grow,
.grow-add.grow-add-active {
  width:400px;
}
img{
  width:100px;
}
```

Using the $location Service

The `$location` service provides a wrapper to the JavaScript `window.location` object. This makes the URL accessible in your AngularJS application. Not only can you get information about the URL, but you also can modify it, changing the location with a new URL or navigating to a specific hash tag.

To add the `$location` service to a controller or service, you need to inject it using the standard dependency injection methods. For example:

Click here to view code image

```
app.controller('myController', ['$scope', '$location',
                                function($scope, location) {
    . . .
  }]);
```

Table 27.5 lists the methods that can be called on the `$location` service and describes their implementation.

Method	Description
`absUrl()`	Returns the full URL that was passed to the browser.
`url([url])`	Returns the URL that was passed to the browser if called with no parameters. For example: `location.url()` Adding a *`url`* parameter to the `url()` method will set the current relative URL for the location. For example: `location.url("/new/path?new=query")`
`protocol()`	Returns the protocol that was used in the current URL.
`host()`	Returns the host that was used in the current URL.
`port()`	Returns the port that was used in the current URL.
`path([path])`	Returns the protocol that was used in the current URL if no parameters are passed. For example: `location.path()` Adding a *`path`* parameter to the `path()` method will set the current relative path for the location. For example: `location.path("/new/path ")`
`search([search], [paramValue])`	Returns a JavaScript object containing the query parameters passed as part of the URL if no *`search`* or *`paramValue`* parameter is passed to the `search()` function. For example: `location.search()` If you pass *`search`* and *`paramValue`* parameters, the value of the parameter named by `search` will be changed in the URL to the value specified by *`paramValue`*. For example: `location.search("param1", "newValue")`
`hash()`	Returns the hash tag that was used in the current URL.
`replace()`	When you call this on the `$location` service object, all subsequent changes to the location will replace the history record instead of adding a new one.

TABLE 27.5 Methods Available on the AngularJS $location Service Object

Try it Yourself: Implementing the $location Service in an AngularJS Application

In this example, you get a chance to implement the `$location` service in an AngularJS application. The code in Listings 27.10 and 27.11 implement a simple example of using the `$location` service to access and change the path, hash, and search elements in the URL passed to the browser.

Use the following steps to build the AngularJS application that dynamically manipulates the URL elements:

1. Add the lesson27/service_location.html and lesson27/js/service_location.js files.
2. Add the code shown in Listing 27.10 and Listing 27.11 to the HTML and JavaScript files.
3. The following code from Listing 27.10 injects the `$location` service into the controller:

Click here to view code image

```
app.controller('myController', ['$scope', '$location',
                                function($scope, location) {
```

4. The `updateLocationInfo()` function in the controller sets values in the scope for elements of the current URL location. These values are represented in the browser by the template defined in Listing 27.11.
5. The following function `changePath` changes the `path` value in the URL to `/new/path` and updates the location information:

Click here to view code image

```
  $scope.changePath = function(){
    location.path("/new/path");
    $scope.updateLocationInfo();
  };
```

6. The following function `changeHash` changes the `hash` value in the URL to `newHash` and updates the location information:

Click here to view code image

```
  $scope.changeHash = function(){
    location.hash("newHash");
    $scope.updateLocationInfo();
  };
```

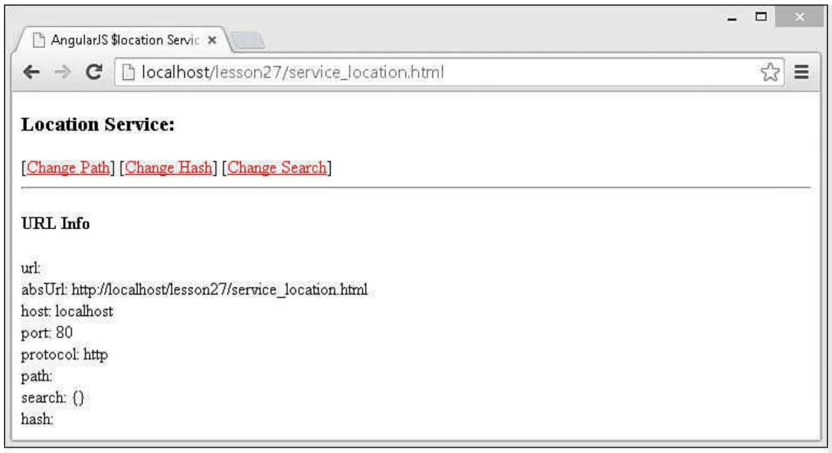
7. The following function `changeSearch` changes the path value in the URL to `?p1=newA` and updates the location information:

Click here to view code image

```
  $scope.changeSearch = function(){
    location.search("p1", "newA");
    $scope.updateLocationInfo();
  };
```

8. The code in Listing 27.11 implements an AngularJS template that displays the captured `$location` service information and provides links to change the `path`, `hash`, and `search` properties.

9. Load the service_location.html file in a browser and click the Change Path, Change Hash, and Change Search links. Figure 27.4 shows the web page in action. Notice that when the links are clicked, the `path`, `hash`, and `search` values in the URL displayed in the browser change.

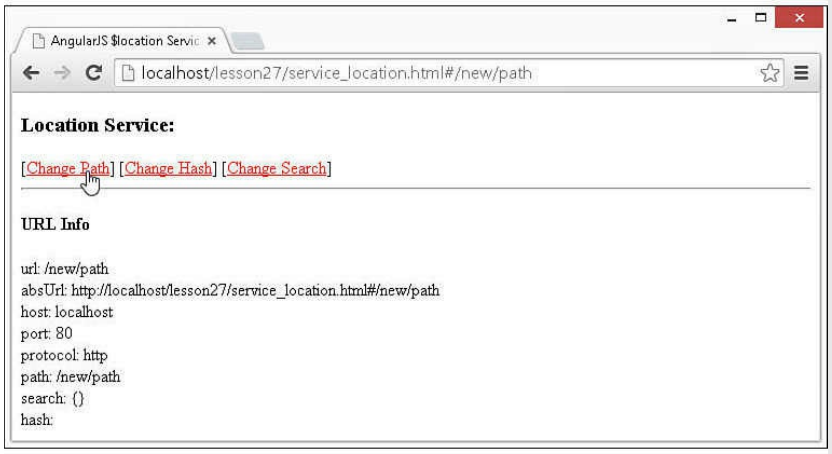

AngularJS $location Servic
localhost/lesson27/service_location.html

Location Service:

[Change Path] [Change Hash] [Change Search]

URL Info

```
url:
absUrl: http://localhost/lesson27/service_location.html
host: localhost
port: 80
protocol: http
path:
search: {}
hash:
```

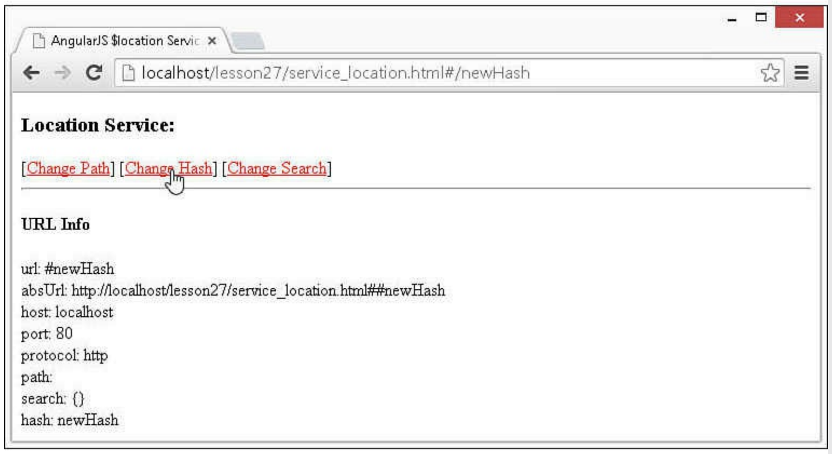

AngularJS $location Servic
localhost/lesson27/service_location.html#/new/path

Location Service:

[Change Path] [Change Hash] [Change Search]

URL Info

```
url: /new/path
absUrl: http://localhost/lesson27/service_location.html#/new/path
host: localhost
port: 80
protocol: http
path: /new/path
search: {}
hash:
```

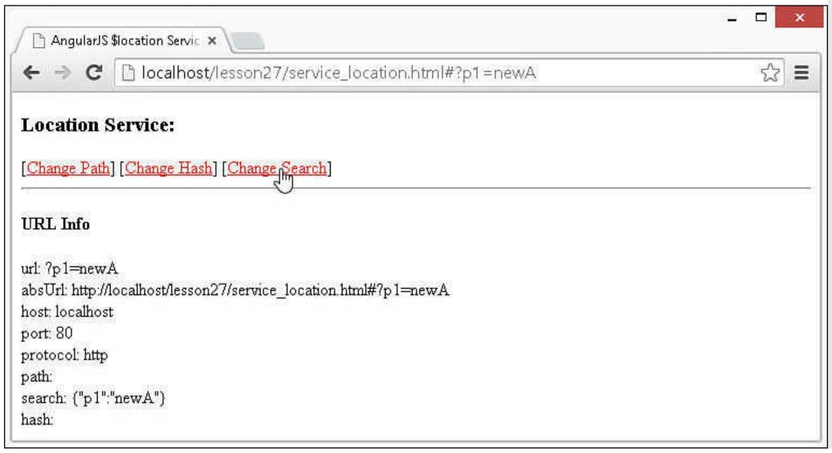

AngularJS $location Servic
localhost/lesson27/service_location.html#/newHash

Location Service:

[Change Path] [Change Hash] [Change Search]

URL Info

```
url: #newHash
absUrl: http://localhost/lesson27/service_location.html##newHash
host: localhost
port: 80
protocol: http
path:
search: {}
hash: newHash
```


AngularJS $location Servic
localhost/lesson27/service_location.html#?p1=newA

Location Service:

[Change Path] [Change Hash] [Change Search]

URL Info

```
url: ?p1=newA
absUrl: http://localhost/lesson27/service_location.html#?p1=newA
host: localhost
port: 80
protocol: http
path:
search: {"p1":"newA"}
hash:
```

FIGURE 27.4 Implementing the `$location` service in an AngularJS application to interact with the browser location.

LISTING 27.10 service_location.js An AngularJS Application That Implements a Controller to Gather Information from the `$location` Service and Provides Functions to Change the `path`, `search`, and `hash` Values

Click here to view code image

```
var app = angular.module('myApp', []);
app.controller('myController', ['$scope', '$location',
                                function($scope, location) {
  $scope.updateLocationInfo = function() {
    $scope.url = location.url();
    $scope.absUrl = location.absUrl();
    $scope.host = location.host();
    $scope.port = location.port();
    $scope.protocol = location.protocol();
    $scope.path = location.path();
    $scope.search = location.search();
    $scope.hash = location.hash();
  };
  $scope.changePath = function(){
    location.path("/new/path");
    $scope.updateLocationInfo();
  };
  $scope.changeHash = function(){
    location.hash("newHash");
    $scope.updateLocationInfo();
  };
  $scope.changeSearch = function(){
    location.search("p1", "newA");
    $scope.updateLocationInfo();
  };
  $scope.updateLocationInfo();
}]);
```

LISTING 27.11 service_location.html An AngularJS Template That Displays Information Gathered from the `$location` Service and Provides Links to Change the `path`, `search`, and `hash` Values

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS $location Service</title>
  <style>
```

```
    span {
      color: red; text-decoration: underline;
      cursor: pointer; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h3>Location Service:</h3>
    [<span ng-click="changePath()">Change Path</span>]
    [<span ng-click="changeHash()">Change Hash</span>]
    [<span ng-click="changeSearch()">Change Search</span>]
    <hr>
    <h4>URL Info</h4>
    url: {{url}}<br>
    absUrl: {{absUrl}}<br>
    host: {{host}}<br>
    port: {{port}}<br>
    protocol: {{protocol}}<br>
    path: {{path}}<br>
    search: {{search}}<br>
    hash: {{hash}}<br>
  </div>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
  <script src="js/service_location.js"></script>
</body>
</html>
```

Using the **$q** Service to Provide Deferred Responses

An extremely useful service provided by AngularJS is the `$q` service. The `$q` service is a promise/deferred response implementation. Because not all services can respond immediately to a request, there is a need to defer the response until the service is ready to respond. That is where the `$q` service comes in. The idea is that you can make a request, and rather than getting the response directly, you will get a promise that the service will respond. The requesting application can then assign a callback function that should be executed when the deferred request completes successfully or fails.

To utilize the `$q` service for deferred responses, you first need to create a deferred object using the following syntax:

```
var deferred = $q.defer();
```

After you have a deferred object, you can pass the promise around by accessing the `promise` attribute. For example, the following line returns the `promise` to the calling application:

Click here to view code image

```
function makeDeferredRequest(){
     var deferred = $q.defer();
     return deferred.promise;
   }
```

The requesting application can then call the `then()` method on the `promise` object to register `successCallback`, `errorCallback`, and `notifyCallback` functions using the following syntax:

Click here to view code image

```
   promise.then(successCallback, [errorCallback], [notifyCallback])
```

The following shows a sample implementation of the `then()` function:

Click here to view code image

```
   var promise = makeDeferredRequest();
   promise.then(
     function successCallback(value){
       //handle success
     },
     function errorCallback(value){
       //handle error
     },
     function notifyCallback(value){
       //handle notify
     },
```

From the deferred service side, you can use the methods described in Table 27.6 to handle notifying the requesting application of the status of the request.

Service	Description
`$resolve(value)`	Executes the `successCallback` function on the promise object and passes the `value` specified to it.
`$reject(reason)`	Executes the `errorCallback` function on the promise object and passes the `reason` specified to it.
`$notify(value)`	Executes the `notifyCallback` function on the promise object and passes the `value` specified to it.

TABLE 27.6 Methods Available on a Deferred Object of the $q Service

The "Implementing a Database Access Service" section of Lesson 28, "Creating Your Own Custom AngularJS Services," shows a good example of using the `$q` service to handle the deferred responses to remote database requests.

Summary

AngularJS services are singleton objects that you can register with the dependency injector. Controllers and other AngularJS components, including other services, can consume them. AngularJS provides much of the back-end functionality in the way of services, such as `$http`, which enables you to easily integrate web server communication into your AngularJS applications.

In this lesson, you also learned about several of the built-in services, such as `$cookieStore`, `$q`, `$window`, `$location`, `$animate`, and `$cacheFactory`. These and other AngularJS services can be used to easily inject functionality into your controllers, directives, and custom services.

Q&A

Q. Are the built-in services provided by AngularJS run as singletons?

A. Yes, but they are lazy singletons, meaning that they get instantiated only if a component depends on it.

Workshop

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

Quiz

1. What does the `$rootElement` AngularJS service do?
2. Which AngularJS service should I use to send and receive requests from the server?
3. Which AngularJS service should you use to get the query string from URL?
4. Which AngularJS server allows you to handle deferred actions for asynchronous requests?

Quiz Answers

1. `$rootElement` provides access to the root element of the AngularJS application.
2. `$http`
3. `$location`
4. `$q`

Exercises

1. Modify the code in Listings 27.5 and 27.6 to add and manage a second cookie for favorite cakes. You should implement the same style of radio button group as for cookie.

2. Modify the code in Listings 27.7, 27.8, and 27.9 to animate moving the image. You will need to change the position of the image to absolute and add a new class called `.slideRight`. Then implement an animation similar to the one for `.fadeOut` that uses jQuery to animate the position instead of the opacity.

Lesson 28. Creating Your Own Custom AngularJS Services

What You'll Learn in This Lesson:

- What methods are available to create AngularJS custom services
- How to use the factory provider to define a custom service
- How to use the service provider to define a custom service
- How to build an application using all four types of custom services
- How to implement an asynchronous database access service

AngularJS provides a lot of functionality in built-in services; however, you will also need to be able to implement your own custom services that provide your own specific functionality. You should implement a custom service anytime you need to provide task-based functionality to your applications.

When implementing custom services, you need to think about the service as a chunk of reusable code that performs one or more related tasks. Then you can design and group them together into custom modules that can easily be consumed by several AngularJS applications.

This lesson introduces the AngularJS custom services. Then the lesson provides several examples of custom AngularJS service implementation to provide you with a clearer understanding of how to design and build your own.

Understanding Custom AngularJS Services

AngularJS enables you to create your own custom services to provide functionality in AngularJS components that require it. As you saw in the previous lesson, the built-in AngularJS services provide a variety of functionality for AngularJS applications. Using custom services, you can customize, enhance, and extend that functionality in many ways.

There are four main types of services that you will likely be implementing in your code: `value`, `constant`, `factory`, and `service`. The following sections cover these services.

Defining a `value` Service

You use the very simple `value` service to define a single value that you can inject as a service provider. The `value` method uses the following syntax, where *name* is the service name and *object* is any JavaScript object you want to provide:

```
value(name, object)
```

For example:

Click here to view code image

```
var app = angular.module('myApp', []);
app.value('myValue', {color:'blue', value:'17'});
```

Defining a **constant** Service

The `constant` service is basically the same as the `value` service, except that `constant` services are available in the configuration phase of building the `Module` object, whereas `value` services are not. The `constant` method uses the following syntax, where name is the service name and object is any JavaScript object you want to provide:

```
constant(name, object)
```

For example:

Click here to view code image

```
var app = angular.module('myApp', []);
app.constant('myConst', "Constant String");
```

Using a Factory Provider to Build a **factory** Service

The `factory` method provides the capability to implement functionality into a service. It can also be dependent on other service providers, enabling you to build up compartmentalized code. The `factory` method uses the following syntax, where name is the service name and factoryProvider is a provider function that builds the factory service:

Click here to view code image

```
factory(name, factoryProvider)
```

Unlike the `value` or `constant` service, you can inject the `factory` method with other services, and it returns the service object with the appropriate functionality. The functionality can be a complex JavaScript service, a value, or a simple function. For example, the following code implements a `factory` service that returns a function that adds two numbers:

Click here to view code image

```
var app = angular.module('myApp', []);
app.constant('myConst', 10);
app.factory('multiplier', ['myConst', function (myConst) {
  return function(value) { return value + myConst; };
}]);
```

Using an Object to Define a **service** Service

The `service` method provides the capability to implement functionality into a server. The `service` method has the same functionality as the `factory` method; however, the `service` method is implemented differently than the `factory` method. Choosing one over the other is a matter of semantics.

The `service` method accepts a constructor function as the second argument and uses it to create a new instance of an object. The `service` method uses the following syntax, where **name** is the service name and **constructor** is a constructor function:

```
service(name, constructor)
```

The `service` method can also accept dependency injection. The following code implements a basic service method that provides an `add()` function and a `multiply()` function:

Click here to view code image

```
var app = angular.module('myApp', []);
app.constant('myConst', 10);
function ConstMathObj(myConst) {
  this.add = function(value){ return value + myConst; };
  this.multiply = function(value){ return value * myConst; };
}
app.service('constMath', ['myConst', ConstMathObj] );
```

Notice that the `ConstMathObj` constructor is created first, and then the `service()` method calls it and uses dependency injection to insert the `myConst` service.

Integrating Custom Services into Your AngularJS Applications

As you begin implementing AngularJS services for your applications, you will find that some will be very simplistic and others will become very complex. The complexity of the service typically reflects the complexity of the underlying data and functionality that it provides. The purpose of this section is to provide you with some basic examples of different types of custom services to illustrate how they can be implemented and utilized.

Each of the following sections contains an example to illustrate different aspects of custom services. The first is designed to show you how to implement the different types of services. The second shows you the reusability of services, and the third shows you a different look into service interactions.

Try it Yourself: Implementing a Simple Application That Uses All Four Types of Services

In this example, you build `constant`, `value`, `factory`, and `service` services. The purpose is to learn how each can be implemented, as well as how to use multiple types of services in your applications.

The code in Listing 28.1 shows an example of integrating `value`, `constant`, `factory`, and `service` services into a single module. The code in Listing 28.2 implements the AngularJS template that loads and runs the application.

Use the following steps to build the application that uses all four types of services:

1. Add the lesson28/service_custom_censor.html and lesson28/js/service_custom_censor.js files.

2. Add the code shown in Listing 28.1 and Listing 28.2 to the HTML and JavaScript files.

3. The following line in Listing 28.1 creates a simple `value` service named `censorWords` that contains the array of words to be censored:

Click here to view code image

```
app.value('censorWords', ["can't", "quit", "fail"]);
```

4. The following line creates a `constant` service named repString that contains the replacement string for censored words:

Click here to view code image

```
app.constant('repString', "****");
```

5. The following lines of code implement a `factory` service named `censorF` that is injected with `censorWords` and `repString`. This service returns a function that accepts an input string and uses replace to replace any words in the `censorWords` list with the value of `repString`:

Click here to view code image

```
app.factory('censorF', ['censorWords', 'repString',
                        function (cWords, repString) {
  return function(inString) {
    var outString = inString;
    for(i in cWords){
      var regex = new RegExp(cWords[i], "ig");
      outString = outString.replace(regex, repString);
    }
    return outString;
  };
}]);
```

6. The following lines of code implement a `service` service named `censorS` by first defining a function named `CensorObj` and then injecting it, along with `censorWords` and `repString`, into the `app.service()` method.

Similar to the `factory` service, this service also accepts a string and returns the censored result:

Click here to view code image

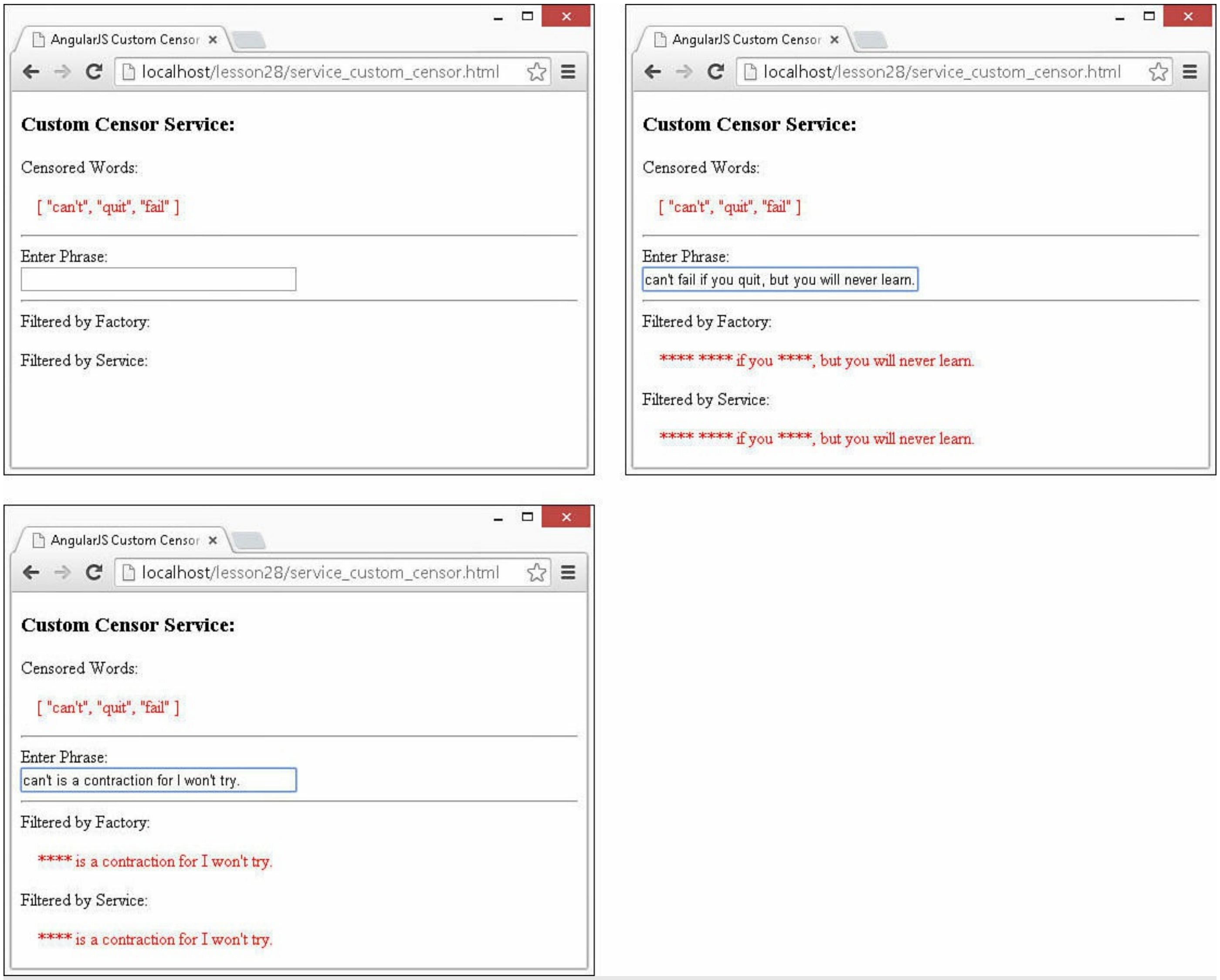
```
15 function CensorObj(cWords, repString) {
16   this.censor = function(inString){
17     var outString = inString;
18     for(i in cWords){
19       var regex = new RegExp(cWords[i], "ig");
20       outString = outString.replace(regex, repString);
21     }
22     return outString;
23   };
24   this.censoredWords = function(){
25     return cWords;
26   };
27 }
28 app.service('censorS', ['censorWords', 'repString', CensorObj]);
```

7. The controller is defined in lines 27–37 and is injected with `censorF` and `censorS`. Notice that in the following lines, `censorF` and `censorS` can be called directly from within the controller:

Click here to view code image

```
36     $scope.censoredByFactory = censorF(newValue);
37     $scope.censoredByService = censorS.censor(newValue);
```

8. Open the service_custom_censor.html file from Listing 28.2 in a browser to load the AngularJS template that displays the censored words and provides a text input to type in a phrase. The phrase is displayed twice, once censored by `censorF` and once by `censorS`. Figure 28.1 shows the AngularJS application in action.


AngularJS Custom Censor

localhost/lesson28/service_custom_censor.html

Custom Censor Service:

Censored Words:

["can't", "quit", "fail"]

Enter Phrase:

Filtered by Factory:

Filtered by Service:

AngularJS Custom Censor

localhost/lesson28/service_custom_censor.html

Custom Censor Service:

Censored Words:

["can't", "quit", "fail"]

Enter Phrase:

can't fail if you quit, but you will never learn.

Filtered by Factory:

**** **** if you ****, but you will never learn.

Filtered by Service:

**** **** if you ****, but you will never learn.

AngularJS Custom Censor

localhost/lesson28/service_custom_censor.html

Custom Censor Service:

Censored Words:

["can't", "quit", "fail"]

Enter Phrase:

can't is a contraction for I won't try.

Filtered by Factory:

**** is a contraction for I won't try.

Filtered by Service:

**** is a contraction for I won't try.

FIGURE 28.1 Using multiple custom services in an AngularJS controller to censor words in a phrase.

LISTING 28.1 service_custom_censor.js Implementing and Consuming Multiple Custom Services in an AngularJS Controller

Click here to view code image

```
var app = angular.module('myApp', []);
app.value('censorWords', ["can't", "quit", "fail"]);
app.constant('repString', "****");
app.factory('censorF', ['censorWords', 'repString',
                        function (cWords, repString) {
  return function(inString) {
    var outString = inString;
    for(i in cWords){
      var regex = new RegExp(cWords[i], "ig");
      outString = outString.replace(regex, repString);
    }
    return outString;
  };
}]);
function CensorObj(cWords, repString) {
  this.censor = function(inString){
    var outString = inString;
    for(i in cWords){
      var regex = new RegExp(cWords[i], "ig");
      outString = outString.replace(regex, repString);
    }
    return outString;
  };
  this.censoredWords = function(){
    return cWords;
  };
}
app.service('censorS', ['censorWords', 'repString', CensorObj]);
app.controller('myController', ['$scope', 'censorF', 'censorS',
                                function($scope, censorF, censorS) {
  $scope.censoredWords = censorS.censoredWords();
  $scope.inPhrase = "";
  $scope.censoredByFactory = censorF("");
  $scope.censoredByService = censorS.censor("");;
  $scope.$watch('inPhrase', function(newValue, oldValue){
    $scope.censoredByFactory = censorF(newValue);
    $scope.censoredByService = censorS.censor(newValue);
  });
}]);
```

LISTING 28.2 service_custom_censor.html AngularJS Template That Illustrates the Interaction of Multiple Custom Services in an AngularJS Controller

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Censor Service</title>
  <style>
    p { color: red; margin-left: 15px; }
    input { width: 250px; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h3>Custom Censor Service:</h3>
    Censored Words:<br>
    <p>{{censoredWords|json}}</p>
    <hr>
    Enter Phrase:<br>
    <input type="text" ng-model="inPhrase" /><hr>
    Filtered by Factory:
    <p>{{censoredByFactory}}</p>
    Filtered by Service:
    <p>{{censoredByService}}</p>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/service_custom_censor.js"></script>
</body>
</html>
```

Try it Yourself: Implementing Simple Time Service

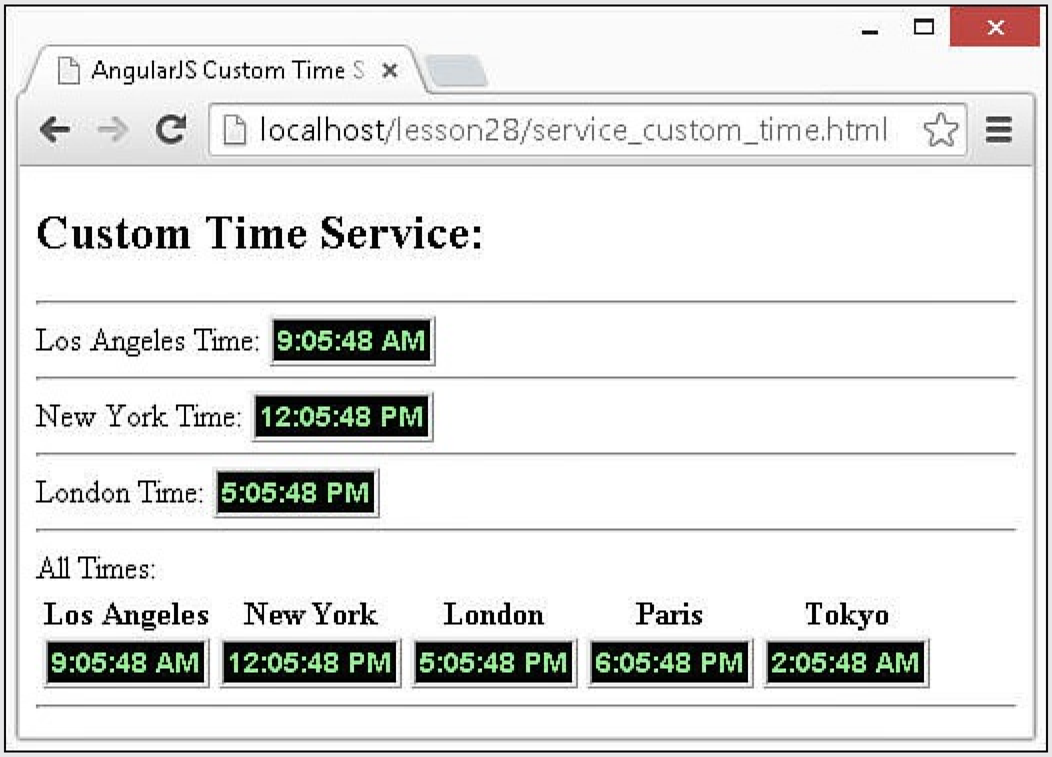
In this example, you will build a simple time service that generates a local time object for different cities. Then, in the AngularJS template, you will use the time service in multiple controllers. The purpose is to give you a chance to see how easy it is to reuse an AngularJS service.

The code in Listing 28.3 implements a custom service named `TimeService` using the function `TimeService()` to generate the service that is used in multiple controllers to display different times. The code in Listing 28.4 implements the AngularJS template that loads and runs the application.

Use the following steps to build the application that implements the custom time service and controllers:

1. Add the lesson28/service_custom_time.html and lesson28/js/service_custom_time.js files.
2. Add the code shown in Listing 28.3 and Listing 28.4 to the HTML and JavaScript files.
3. The following lines of code in Listing 28.3 implement a basic time service named `TimeService` that uses a `cities` object that defines the UTC offset to get the time. Notice that the `getTZDate()` method accepts a city name and then calculates the data based off of the UTC time and offset. The `getCitIes()` function returns a list of the cities supported by the service.
4. The rest of the application in Listing 28.3 implements several controllers, including `LAController`, `NYController`, `LondonController`, and `TimeController`. These controllers are injected with the `TimeService` service and use it to set the current time for a city or, in the case of `TimeController`, all cities.
5. Open the service_custom_time.html file from Listing 28.4 in a browser. This implements an AngularJS template that displays the time for the `LAController`, `NYController`, `LondonController`, and

TimeController controllers. In the case of TimeController, all times are displayed in a table using ng-repeat. Figure 28.2 shows the resulting AngularJS application web page. Notice the different times represented.



Custom Time Service:

Los Angeles Time: 9:05:48 AM

New York Time: 12:05:48 PM

London Time: 5:05:48 PM

All Times:

Los Angeles	New York	London	Paris	Tokyo
9:05:48 AM	12:05:48 PM	5:05:48 PM	6:05:48 PM	2:05:48 AM

FIGURE 28.2 Using a custom AngularJS service in multiple controllers to display the time for different cities.

LISTING 28.3 service_custom_time.js Implementing and Consuming a Custom AngularJS Service in Multiple Controllers

Click here to view code image

```
var app = angular.module('myApp', []);
function TimeService() {
  var cities = { 'Los Angeles': -8,
                 'New York': -5,
                 'London': 0,
                 'Paris': 1,
                 'Tokyo': 9 };
  this.getTZDate = function(city){
    var localDate = new Date();
    var utcTime = localDate.getTime() +
                  localDate.getTimezoneOffset() *
                  60*1000;
    return new Date(utcTime +
                     (60*60*1000 *
                      cities[city]));
  };
  this.getCities = function(){
    var cList = [];
    for (var key in cities){
      cList.push(key);
    }
    return cList;
  };
}
app.service('TimeService', [TimeService]);
app.controller('LAController', ['$scope', 'TimeService',
                                function($scope, timeS) {
  $scope.myTime = timeS.getTZDate("Los Angeles").toLocaleTimeString();
}]);
app.controller('NYController', ['$scope', 'TimeService',
                                function($scope, timeS) {
  $scope.myTime = timeS.getTZDate("New York").toLocaleTimeString();
}]);
app.controller('LondonController', ['$scope', 'TimeService',
                                    function($scope, timeS) {
  $scope.myTime = timeS.getTZDate("London").toLocaleTimeString();
}]);
app.controller('TimeController', ['$scope', 'TimeService',
                                  function($scope, timeS) {
  $scope.cities = timeS.getCities();
  $scope.getTime = function(cityName){
    return timeS.getTZDate(cityName).toLocaleTimeString();
  };
}]);
```

LISTING 28.4 service_custom_time.html AngularJS Template That Illustrates Injecting a Custom AngularJS Service into Multiple Controllers

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Time Service</title>
  <style>
    span {
      color: lightgreen; background-color: black;
      border: 3px ridge; padding: 2px;
      font: 14px/18px arial, serif; }
  </style>
</head>
<body>
  <h2>Custom Time Service:</h2><hr>
  <div ng-controller="LAController">
    Los Angeles Time:
    <span>{{myTime}}</span>
  </div><hr>
  <div ng-controller="NYController">
```

```
    New York Time:
    <span>{{myTime}}</span>
  </div><hr>
  <div ng-controller="LondonController">
    London Time:
    <span>{{myTime}}</span>
  </div><hr>
  <div ng-controller="TimeController">
    All Times:
    <table>
    <tr>
         <th ng-repeat="city in cities">
           {{city}}
         </th>
    </tr>
    <tr>
         <td ng-repeat="city in cities">
           <span>{{getTime(city)}}</span>
         </td>
    </tr>
    </table>
  </div><hr>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/service_custom_time.js"></script>
</body>
</html>
```

Try it Yourself: Implementing a Database Access Service

In this example, you build an intermediate database access service that uses the `$http` service to connect to a simulated server-side database service. The purpose of this exercise is to illustrate the use of built-in services alongside custom services. This also gives a good example of utilizing the `$q` service.

Use the following steps to build the custom web server with database simulation, implement the database access server, and build an AngularJS application that uses the service:

1. Add the lesson28/service_custom_db.html, lesson28/js/service_custom_db.js, and lesson28/js/service_custom_db_access.js files.
2. Add the code shown in Listings 28.6, 28.7, and 28.8 to the HTML and JavaScript files.
3. For this example, we need to have the web server provide a simulated database access service. Listing 28.5 implements the Node.js web server that handles the following `GET` and `POST` routes to get and set a user object and an array of table data to simulate making requests to a remote database service:

- **/get/user:** A `GET` route that returns the JSON version of a user object.
- **/get/data:** A `GET` route that returns the JSON version of an array of table data.
- **/set/user:** A `POST` route that accepts a user object in the body of the request and updates the object on the server to simulate storing a user object.
- **/set/data:** A `POST` route that accepts an array of objects in the body of the request and updates the data variable to simulate storing database data. Typically, you would never store all the table data at once, but for simplicity in this example, this is how it is.

You don't necessarily need to pay a lot of attention to the code in Listing 28.5 other than understanding the routes that it provides so that you can follow the interactions in the AngularJS application defined in Listings 28.6, 28.7, and 28.8. The server is very rudimentary, doesn't handle errors, and dynamically generates data to simulate the database.

By the Way

You will need to stop the normal `server.js` HTTP server if it is running before starting `service_db_server.js` from Listing 28.5 (don't forget to stop `service_db_server.js` and reload `server.js` when you are done with this exercise). Also, you should place the `service_db_server.js` file from Listing 28.5 in the parent folder to the `service_db_access.html` in Listing 28.6 for the paths to match up properly in the Node.js static routes. The structure should look similar to this:

Click here to view code image

```
./service_db_server.js
./lesson28/service_custom_db.html
./lesson28/js/service_custom_db_access.js
./lesson28/js/service_custom_db.js
```

LISTING 28.5 service_db_server.js Implementing a Node.js Express Server That Supports GET and POST Routes to Simulate a Database Service for the AngularJS Controller

Click here to view code image

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use('/', express.static('./'));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
var user = {
             first: 'Christopher',
             last: 'Columbus',
             username: 'cc1492',
             title: 'Admiral',
             home: 'Genoa'
           };
var data = [];
function r(min, max){
  var n = Math.floor(Math.random() * (max - min + 1)) + min;
  if (n<10){ return '0' + n; }
  else { return n; }
}
function p(start, end, total, current){
  return Math.floor((end-start)*(current/total)) + start;
}
function d(plusDays){
  var start = new Date(1492, 7, 3);
  var current = new Date(1492, 7, 3);
  current.setDate(start.getDate()+plusDays);
  return current.toDateString();
}
function makeData(){
  var t = 70;
  for (var x=0; x < t; x++){
    var entry = {
      day: d(x),
      time: r(0, 23) + ':' + r(0, 59),
      longitude: p(37, 25, t, x) + '\u00B0 '+ r(0,59) + ' N',
      latitude: p(6, 77, t, x) + '\u00B0 '+ r(0,59) + ' W'
    };
    data.push(entry);
  }
}
makeData();
app.get('/get/user', function(req, res){
  res.json(user);
});
app.get('/get/data', function(req, res){
  res.json(data);
});
app.post('/set/user', function(req, res){
  console.log(req.body.userData);
  user = req.body.userData;
  res.json({ data: user, status: "User Updated." });
});
app.post('/set/data', function(req, res){
  data = req.body.data;
  res.json({ data: data, status: "Data Updated." });
});
```

```
app.listen(80);
```

4. Add the code in Listing 28.6 that implements a module named `dbAccess` and a custom service named `DBService`. The `DBAccessObj()` function that creates the service object provides the `getUserData()` and `updateUser()` methods to retrieve and update the user object from the server using `$http` requests. The `getData` and `updateData()` methods provide similar functionality for the table data. Notice how the `$q` service is used to defer the response to the `$http` requests since the request will not return immediately.

LISTING 28.6 service_custom_db_access.js Implementing a Custom AngularJS Service That Utilizes the `$http` and `$q` Services to Provide Interaction with Data Stored on the Server

Click here to view code image

```
var app = angular.module('dbAccess', []);
function DBAccessObj($http, $q) {
  this.getUserData = function(){
    var deferred = $q.defer();
    $http.get('/get/user')
    .success(function(response, status, headers, config) {
      deferred.resolve(response);
    });
    return deferred.promise;
  };
  this.updateUser = function(userInfo){
    var deferred = $q.defer();
    $http.post('/set/user', { userData: userInfo}).
    success(function(response, status, headers, config) {
      deferred.resolve(response);
    });
    return deferred.promise;
  };
  this.getData = function(){
    var deferred = $q.defer();
    $http.get('/get/data')
    .success(function(response, status, headers, config) {
      deferred.resolve(response);
    });
    return deferred.promise;
  };
  this.updateData = function(data){
    var deferred = $q.defer();
    $http.post('/set/data', { data: data}).
    success(function(response, status, headers, config) {
      deferred.resolve(response);
    });
    return deferred.promise;
  };
}
app.service('DBService', ['$http', '$q', DBAccessObj]);
```

5. Add the code in Listing 28.7 that implements the application module. Notice that on line 1, the `dbAccess` module is injected into the `myApp` module to provide access to the `DBService` service. `DBService` is injected into the controller on line 2 and then used on lines 6, 12, 18, and 23 to make calls to get and set data from the server and assign it to the `$scope.userInfo` and `$scope.data` values in the scope. Notice how the `$q` service `then()` function is used to handle the deferred responses. For simplicity, only the `successCallback` function is implemented. Normally you would also want to implement an `errorCallback` function.

LISTING 28.7 service_custom_db.js Implementing an AngularJS Application That Injects the Module and Service from Listing 28.6 to Utilize the Database Access Service

Click here to view code image

```
01 var app = angular.module('myApp', ['dbAccess']);
02 app.controller('myController', ['$scope', 'DBService',
03                                 function($scope, db) {
04   $scope.status = "";
05   $scope.getUser = function(){
06     db.getUserData().then(function(response){
07       $scope.userInfo = response;
08       $scope.status = "User Data Retrieve.";
09     });
10   };
11   $scope.getData = function(){
12     db.getData().then(function(response){
13       $scope.data = response;
14       $scope.status = "User Data Retrieve.";
15     });
16   };
17   $scope.updateUser = function(){
18     db.updateUser($scope.userInfo).then(function(response){
19       $scope.status = response.status;
20     });
21   };
22   $scope.updateData = function(){
23     db.updateData($scope.data).then(function(response){
24       $scope.status = response.status;
25     });
26   };
```

```
  $scope.getUser();
  $scope.getData();
}]);
```

6. Add the code in Listing 28.8 that implements an AngularJS template that displays the user info in text inputs and binds the values directly to the scope. There are also two input buttons that call `updateUser()` to update the user info on the server and `getUser()` to refresh the scope data from the server. Similarly, two input buttons call `updateData()` and `getData()` to provide the same functionality for updating and refreshing the table data from the model.

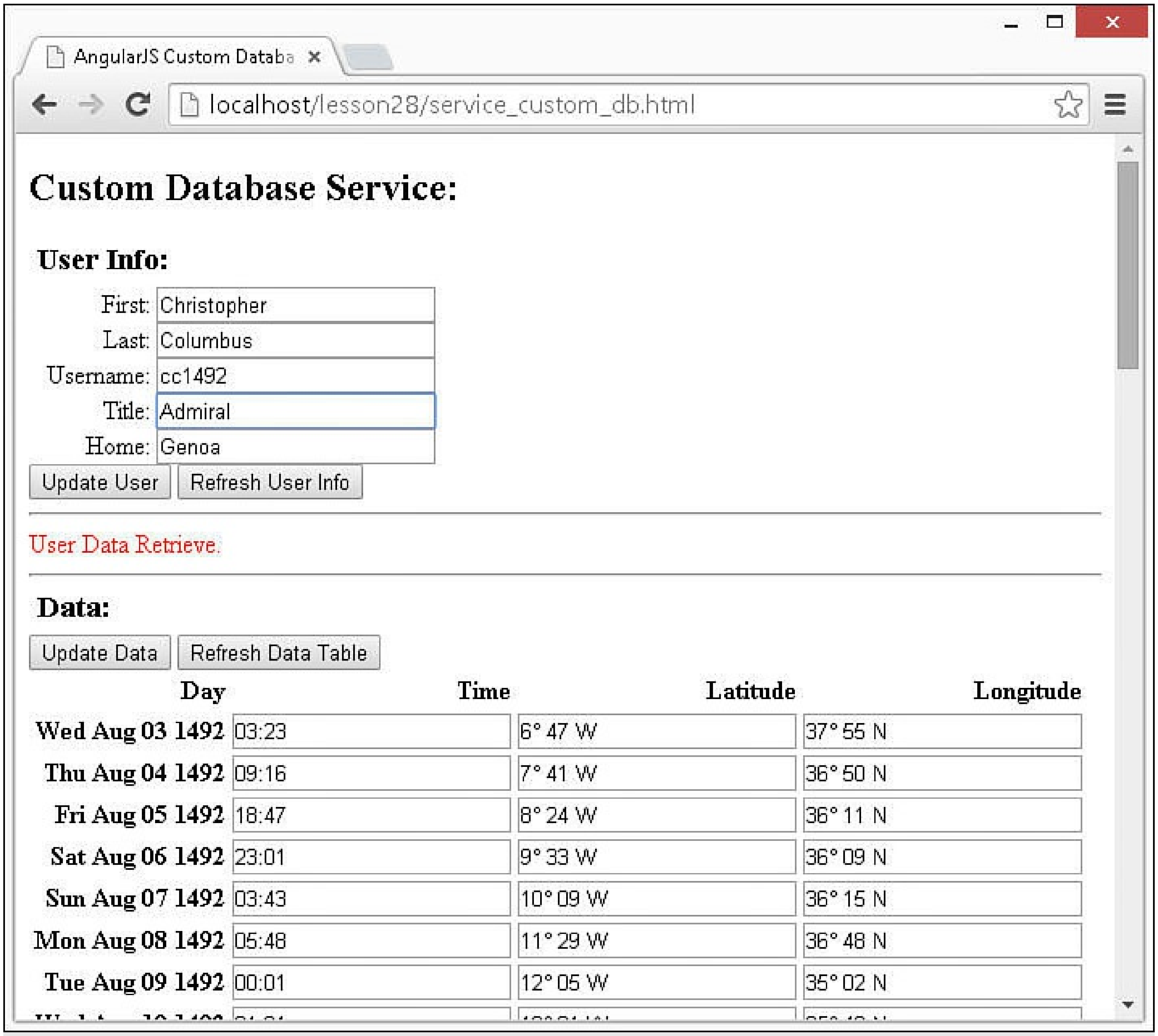
LISTING 28.8 service_custom_db.html AngularJS Template That Uses a Series of `<input>` Elements to Display and Update Data Retrieved from the Server

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Database Service</title>
     <style>
       label {
         display: inline-block; width: 75px; text-align: right; }
       td, tr {
         width: 125px; text-align: right; }
       p {
         color: red; font: italic 12px/14px; margin: 0px;}
       h3 {
         margin: 5px; }
     </style>
</head>
<body>
  <h2>Custom Database Service:</h2>
  <div ng-controller="myController">
    <h3>User Info:</h3>
    <label>First:</label>
      <input type="text" ng-model="userInfo.first" /><br>
    <label>Last:</label>
      <input type="text" ng-model="userInfo.last" /><br>
    <label>Username:</label>
      <input type="text" ng-model="userInfo.username" /><br>
    <label>Title:</label>
      <input type="text" ng-model="userInfo.title" /><br>
    <label>Home:</label>
      <input type="text" ng-model="userInfo.home" /><br>
    <input type= button ng-click="updateUser()" value="Update User" />
    <input type= button ng-click="getUser()" value="Refresh User Info"
/>
```

```
    <hr>
    <p>{{status}}</p>
    <hr>
    <h3>Data:</h3>
    <input type= button ng-click="updateData()" value="Update Data" />
    <input type= button ng-click="getData()" value="Refresh Data Table"
/><br>
    <table>
      <tr><th>Day</th><th>Time</th><th>Latitude</th><th>Longitude</th>
</tr>
      <tr ng-repeat="datum in data">
        <th>{{datum.day}}</th>
        <td><input type="text" ng-model="datum.time" /></td>
        <td><input type="text" ng-model="datum.latitude" /></td>
        <td><input type="text" ng-model="datum.longitude" /></td>
      </tr>
    </table>
    <hr>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/service_custom_db_access.js"></script>
  <script src="js/service_custom_db.js"></script>
</body>
</html>
```

7. Load service_custom_db.html from Listing 28.8 in a browser. Figure 28.3 shows the rendered AngularJS web application working. When you click the Update User or Update Data buttons, the values are changed on the server. That means that you can reload the web page and even exit the browser and come back, and the values will still be the updated versions. If you change values in the data or user area without clicking the Update User or Update Data buttons, you can click Refresh User Info or Refresh Data Table to retrieve the old values from the server.



Custom Database Service:

User Info:

First:	Christopher
Last:	Columbus
Username:	cc1492
Title:	Admiral
Home:	Genoa

User Data Retrieve.

Data:

Day	Time	Latitude	Longitude
Wed Aug 03 1492	03:23	6° 47 W	37° 55 N
Thu Aug 04 1492	09:16	7° 41 W	36° 50 N
Fri Aug 05 1492	18:47	8° 24 W	36° 11 N
Sat Aug 06 1492	23:01	9° 33 W	36° 09 N
Sun Aug 07 1492	03:43	10° 09 W	36° 15 N
Mon Aug 08 1492	05:48	11° 29 W	36° 48 N
Tue Aug 09 1492	00:01	12° 05 W	35° 02 N

FIGURE 28.3 Using custom AngularJS services that implement `$http` to provide access to retrieve and update data on the server.

Summary

AngularJS custom services are singleton objects that you can register with the dependency injector. After they're registered with the dependency injector, controllers, directives, and other AngularJS components, including other services, can consume them. AngularJS provides several methods for creating custom services, with varying levels of complexity. The `value` and `constant` methods create simple services. On the other hand, the `factory` and `service` methods enable you to create much more complex services.

This lesson focused on the tools that enable you to implement your own custom

AngularJS services when you need to provide task-based functionality to your applications. You learned about the four methods or types of custom AngularJS services, including `value`, `constant`, `factory`, and `service`.

This lesson showed examples of implementing each of the types of custom AngularJS services. You also saw an example of implementing a custom AngularJS service in multiple controllers. The final example in this lesson showed you how to implement a standalone custom AngularJS service that interacts with the server using `$http` and how to inject and use it in another module.

Q&A

Q. Is it possible to mock AngularJS services in my unit tests?

A. Yes. You can mock the service providers using something like the Jasmine test-driven development framework.

Q. What is the difference between a value service and a constant service?

A. They are almost identical in behavior. The only real difference is that value services cannot be injected into the configuration block of a module.

Workshop

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

Quiz

1. Which AngularJS service provider would you use if you wanted to define the initial value for a single variable?
2. How many types of service providers are there?
3. True or false: Services can be used only once within your module.

Quiz Answers

1. The `value` service provider.
2. Value, constant, factory, and service.
3. False. Services can be injected into multiple components of your module.

Exercises

1. Modify the code in Listings 28.1 and 28.2 to add the capability for the user to add a new word to the censored words list. Add a function to the `CensorObj` object

that allows the user to add words to the `cWords` list. Add a function to the controller that will call that method to add a new word to censor on. Finally, add a text input and button to the template that hooks into the scope.

2. Modify the code in Listings 28.3 and 28.4 to add local time to the time service and display it on the page. You will need to add a function to `TimeObj` that gets the local time and returns it. Then add a controller for the local time and implement the controller in the template like any other city.

Lesson 29. Creating Rich Web Application Components the AngularJS Way

What You'll Learn in This Lesson:

- Ways to implement dragging and dropping using custom AngularJS directives
- How to use nested AngularJS directives to build a tabbed panel application
- Using custom AngularJS directives to implement expandable and collapsible containers
- How to use simple AngularJS code to implement a star ratings view

The previous AngularJS lessons of this book have been directed toward teaching you the mechanics and basic implementation of the different components of AngularJS applications. You've learned about scope/model, views, controllers, directives, and services. This lesson switches gears a bit and provides some examples to help solidify how things are done in AngularJS.

AngularJS expects a lot more structure than normal JavaScript or even jQuery requires. However, it still provides a lot of flexibility within the framework. Consequently, it is a good idea to look at as many different angles of doing things in AngularJS as possible.

That is why this entire lesson is devoted to a series of examples of implementing AngularJS in various ways. The examples in this lesson are not polished—some more than others—however, they do provide different looks at implementing custom directives and utilizing the built-in directives. The purpose is not to provide you with instantly reusable code, but to give you some different views and a basic framework that you can build on as you design your own implementations.

Try it Yourself: Building a Tabbed View

In this example, you build two custom AngularJS directives, one that acts as a tab group and the other that acts as the individual panes in the tabbed group. The objective of this example is to give you a look at nesting custom directives inside each other, as well as some communication between the two.

Use the following steps to build the tabbed view AngularJS application:

1. Create the following folder and file structure in your code location. The server.js file and images folders were created in Lesson 1, "Introduction to Dynamic Web Programming." You will need to get the images from the book's website or supply your own:
   - **./server.js:** Node.js web server that serves the static project files.
   - **./images:** Folder that contains the images used in the examples.

- **./lesson29:** Project folder.
- **./lesson29/tabbable.html:** AngularJS template for the project.
- **./lesson29/tabs.html:** AngularJS partial template for the tabbed group.
- **./lesson29/panel.html:** AngularJS partial template for each individual panel in the tabbed group.
- **./lesson29/js/tabbable.js:** AngularJS application supporting the custom tabs directives.

2. Add the code in Listing 29.1 to the `tabbable.js` file. This is an AngularJS application that defines the two directives, `myTabs` and `myPane`. Note that the `directive` functions simply return an object that defines the directive. This object is used by AngularJS to create an instance of the directive when one is encountered in the HTML.

   Also note that the HTML used inside the templates comes from the partial files using the `templateUrl` option in the directive definition. Also note that the transclude option is used, which enables us to keep the contents for the `myPane` elements in the AngularJS template.

   Communication between the two directives is made possible by requiring the `myTabs` directive in the definition for `myPane`. This causes the controller defined in `myTabs` to be passed in to the link function of `myPane`. Note that on line 30, we are able to call `addPane()` to add the scope for the `myPane` directive to a list in the `myTabs` directive. The visible tab is changed using the `select()` method in the `myTabs` controller function.

LISTING 29.1 tabbable.js AngularJS Application That Defines Two Custom Directives That Can Be Nested to Provide a Tabbed Panel View

Click here to view code image

```
01 var app = angular.module('myApp', []);
02 app.directive('myTabs', function() {
03   return {
04     restrict: 'E',
05     transclude: true,
06     scope: {},
07     controller: function($scope) {
08       var panes = $scope.panes = [];
09       $scope.select = function(pane) {
10         angular.forEach(panes, function(pane) {
11           pane.selected = false;
12         });
```

```
        pane.selected = true;
      };
      this.addPane = function(pane) {
        if (panes.length == 0) {
          $scope.select(pane);
        }
        panes.push(pane);
      };
    },
    templateUrl: 'tabs.html'
  };
});
app.directive('myPane', function() {
  return { require: '^myTabs', restrict: 'E',
    templateUrl: 'pane.html',
    transclude: true, scope: { title: '@' },
    link: function(scope, element, attrs, tabsCtrl) {
      tabsCtrl.addPane(scope);
    }
  };
});
```

3. Add the code in Listing 29.2 to the table.html file. This is an AngularJS partial template that acts as the replacement for the `myTabs` directive. Notice that the `panes` value in the scope is used to add the tabs as `<span>` elements to the top of the view. The `panes` array is built as each `myPane` element is compiled and linked into the template.

LISTING 29.2 tabs.html AngularJS Partial Template That Contains the Template Code to Build the Tabs Container

Click here to view code image

```
<div class="tabbable">
  <div class="tabs">
    <span class="tab" ng-repeat="pane in panes"
        ng-class="{activeTab:pane.selected}"
        ng-click="select(pane)">{{pane.title}}
    </span>
  </div>
  <div class="tabcontent" ng-transclude></div>
</div>?
```

4. Add the code in Listing 29.3 to the pane.html file. This contains the AngularJS partial template that acts as the replacement for the `myPane` directive. Notice that we use `ng-show` to show and hide the panes as they are clicked. The `ng-transclude` attribute ensures that the contents defined in the `myPane` element are included in the rendered view.

LISTING 29.3 pane.html AngularJS Partial Template That Contains the Template Code to Build the Individual Panes of the Tabbed Container

```
<div class="pane"
     ng-show="selected"
     ng-transclude>
</div>
```

5. Add the code in Listing 29.4 to the tabbable.html file. This is the AngularJS template that supports the `myTabs` and `myPane` directives. Note the naming structure of `my-tabs` and `my-pane` for the elements needed in the template. For this example, only images are placed inside the `myPane` element. This could just as easily be a complex series of elements, such as a form or table.

LISTING 29.4 tabbable.html AngularJS Template That Implements the myTabs and myPane Custom Directives to Create a Tabbed View

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>Tab and Tab Pane Directives</title>
  <style>
    .tab{
      display:inline-block; width:100px;
      border-radius: .5em .5em 0 0; border:1px solid black;
      text-align:center; font: 15px/28px Helvetica, sans-serif;
      background-image: linear-gradient(#CCCCCC, #EEEEEE);
      cursor: pointer; }
    .activeTab{
      border-bottom: none;
      background-image: linear-gradient(#66CCFF, #CCFFFF); }
    .pane{
      border:1px solid black; background-color: #CCFFFF;
      height:300px; width:400px;
      padding:10px;  margin-top:-2px;
      overflow: scroll; }
  </style>
</head>
<body>
  <h2>AngularJS Custom Tabs</h2>
  <my-tabs>
    <my-pane title="Canyon">
      <img src="/images/canyon.jpg" />
    </my-pane>
    <my-pane title="Lake">
      <img src="/images/lake.jpg" />
    </my-pane>
    <my-pane title="Sunset">
      <img src="/images/jump.jpg" />
    </my-pane>
  </my-tabs>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
  <script src="js/tabbable.js"></script>
</body>
</html>
```

6. Load the tabbable.html file in a browser. Figure 29.1 shows the working web page. Notice that as the tab for each pane is clicked, the content changes.

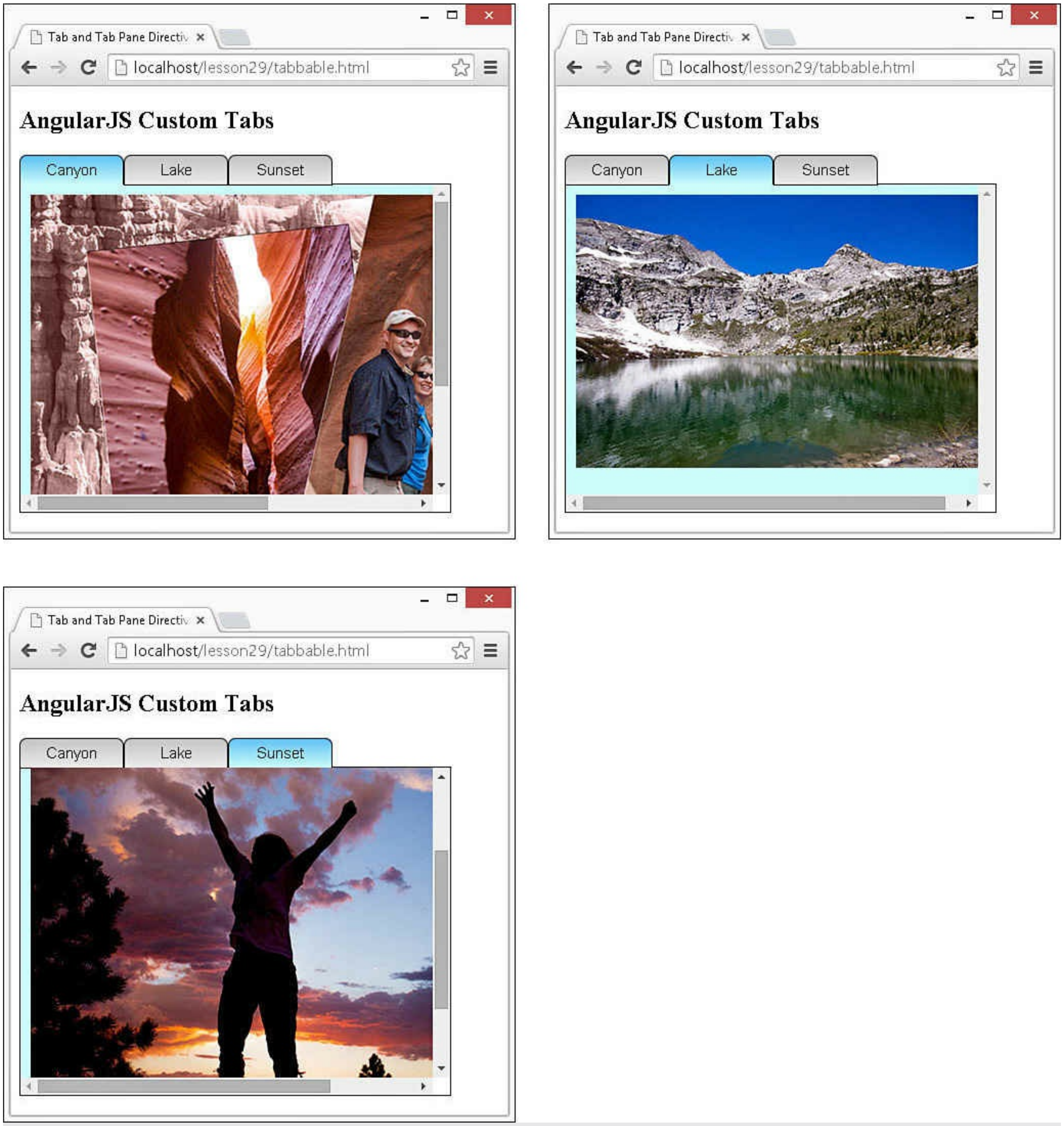


FIGURE 29.1 Implementing nested custom AngularJS directives to build a tabbed pane view.

Try it Yourself: Implementing Draggable and Droppable Elements

In this example, you use custom AngularJS directives to implement a set of draggable elements containing words that can be dragged onto a set of droppable image elements. When the word is dropped on an image, it is appended to a list of words that appear below the image.

The purpose of this exercise is to show you an example of using the HTML5 drag and drop events. The example uses only the events, and the actual drag and drop functionality is built using the AngularJS mechanisms. The reason is to illustrate using AngularJS (plus, the HTML5 drag and drop is not well implemented and needs to be revised). Another thing illustrated in this example is appending new elements to existing ones in an AngularJS directive.

Use the following steps to build the drag/drop AngularJS application:

1. Create the following folder and file structure in your code:
   - **./lesson29:** Project folder.
   - **./lesson29/dragdrop.html:** AngularJS template for the project.
   - **./lesson29/js/dragdrop.js:** AngularJS application supporting the custom drag and drop directives.
2. Add the code in Listing 29.5 to the `dragdrop.js` file. This is an AngularJS application that defines two new custom AngularJS directives, `dragit` and `dropit`. Notice that in the parent scope, the `dragStatus` and `dropStatus` variables are defined; these are updated in the custom directives. This is possible because no isolate scope is defined in the directives, so they share the parent controller scope.

   Notice that inside the `dragit` directive, the HTML5 draggable attribute is added to the `dragit` element using the `attr()` method. Also in the `dragit` directive, the `dragstart`, `drag`, and `dragend` event handlers are implemented. For `dragstart` and `drag`, the default behavior is to allow the drag to start and `dragenter/dragleave` events to fire. However, `dragend` prevents the default behavior so that our custom AngularJS code can handle the drop.

   Inside the `dropit` directive, the `dragover`, `dragleave`, `dragenter`, and `drop` are implemented. Notice that in `drop`, we use the append method to append a <p> element to the `dropit` element. The value inside the paragraph comes from the scope and was set during `dragstart` in the `dragit` directive. Again, this is possible because no isolate scopes are defined in the directives.

LISTING 29.5 dragdrop.js AngularJS Application That Implements dragit and dropit Custom AngularJS Directives to Provide Drag and Drop Functionality

Click here to view code image

```
var app = angular.module('myApp', []);
app.controller('myController', function($scope) {
  $scope.dragStatus = "none";
  $scope.dropStatus = "none";
  $scope.dropValue = "";
})
.directive('dragit', function($document, $window) {
  function makeDraggable(scope, element, attr) {
    angular.element(element).attr("draggable", "true");
    element.on('dragstart', function(event) {
      element.addClass('dragItem');
      scope.$apply(function(){
        scope.dragStatus = "Dragging " + element.html();
        scope.dropValue = element.html();
      });
      event.dataTransfer.setData('Text', element.html());
    });
    element.on('drag', function(event) {
      scope.$apply(function(){
        scope.dragStatus = "X: " + event.pageX +
                           " Y: " + event.pageY;
      });
    });
    element.on('dragend', function(event) {
      event.preventDefault();
      element.removeClass('dragItem');
    });
  }
  return {
    link: makeDraggable
  };
})
.directive('dropit', function($document, $window) {
  return {
    restrict: 'E',
    link: function makeDroppable(scope, element, attr){
      element.on('dragover', function(event) {
        event.preventDefault();
        scope.$apply(function(){
          scope.dropStatus = "Drag Over";
        });
      });
      element.on('dragleave', function(event) {
        event.preventDefault();
        element.removeClass('dropItem');
        scope.$apply(function(){
          scope.dropStatus = "Drag Leave";
        });
      });
      element.on('dragenter', function(event) {
        event.preventDefault();
        element.addClass('dropItem');
        scope.$apply(function(){
          scope.dropStatus = "Drag Enter";
        });
      });
      element.on('drop', function(event) {
        event.preventDefault();
        element.removeClass('dropItem');
        scope.$apply(function(){
          element.append('<p>' +
              event.dataTransfer.getData('Text') + '</p>');
          scope.dropStatus = "Dropped " + scope.dropValue;
        });
      });
    }
  };
});
```

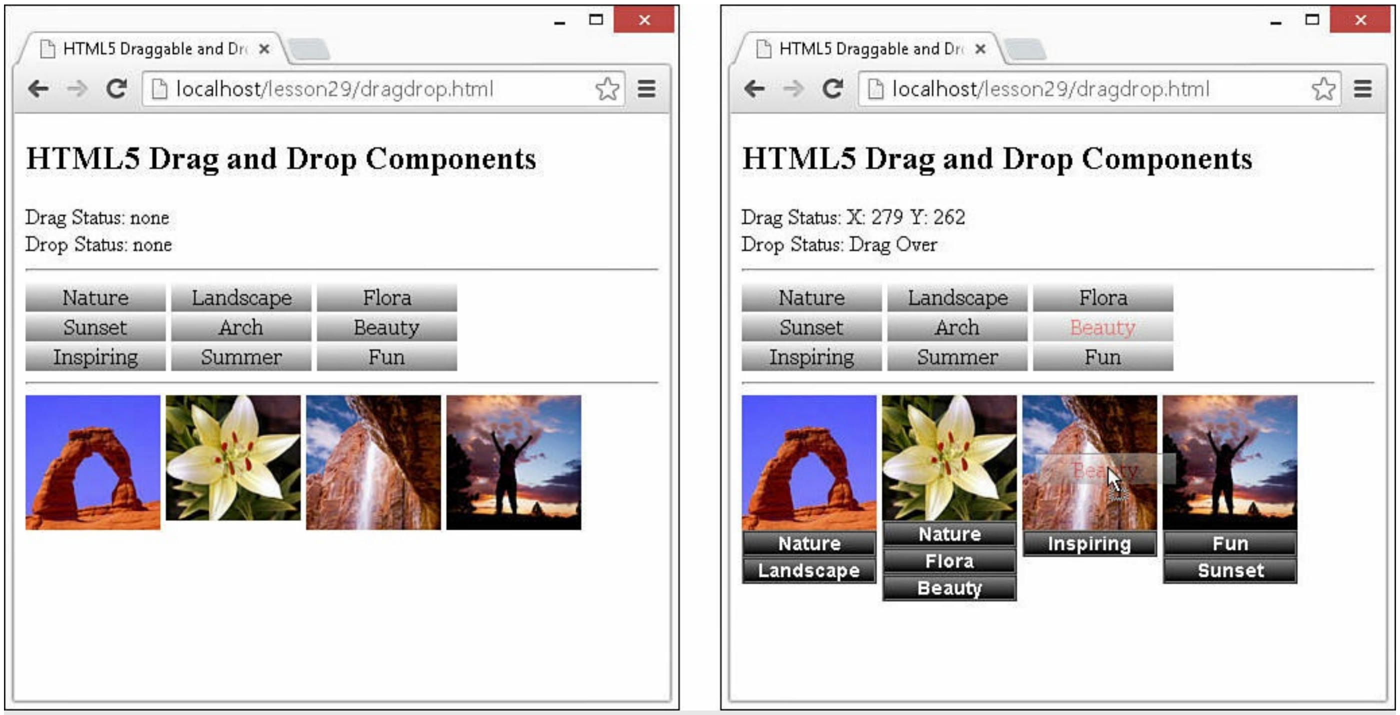
3. Add the code in Listing 29.6 to the dragdrop.html file. This implements the AngularJS template that displays the `dragStatus` and `dropStatus` values. Notice that the draggable elements are declared using the `<dragit>` syntax, and the droppable elements are declared using the `<dropit>` syntax.

LISTING 29.6 dragdrop.html AngularJS Template That Uses the `dragit` and `dropit` Directives to Add Draggable and Droppable Elements to the Web Page

Click here to view code image

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>HTML5 Draggable and Droppable Directives</title>
  <style>
    dropit, img, p{
      vertical-align: top; text-align: center;
            width: 100px;
            display: inline-block;
          }
          p {
            color: white; background-color: black;
            font: bold 14px/16px arial;
      margin: 0px; width: 96px;
      border: 2px ridge grey;
      background: linear-gradient(#888888, #000000);
          }
          span{
            display:inline-block; width: 100px;
            font: 16px/18px Georgia, serif; text-align: center;
            padding: 2px;
            background: linear-gradient(#FFFFFF, #888888);
          }
          .dragItem {
            color: red;
      opacity: .5;
          }
    .dropItem {
      border: 3px solid red;
      opacity: .5;
    }
    #dragItems {
      width: 400px;
    }
  </style>
</head>
<body>
  <h2>HTML5 Drag and Drop Components</h2>
  <div ng-controller="myController">
    Drag Status: {{dragStatus}}<br>
    Drop Status: {{dropStatus}}
    <hr>
    <div id="dragItems">
         <span dragit>Nature</span>
         <span dragit>Landscape</span>
         <span dragit>Flora</span>
         <span dragit>Sunset</span>
         <span dragit>Arch</span>
         <span dragit>Beauty</span>
         <span dragit>Inspiring</span>
      <span dragit>Summer</span>
      <span dragit>Fun</span>
    </div>
    <hr>
    <dropit><img src="/images/arch.jpg" /></dropit>
    <dropit><img src="/images/flower.jpg" /></dropit>
    <dropit><img src="/images/cliff.jpg" /></dropit>
    <dropit><img src="/images/jump.jpg" /></dropit>
  </div>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
  <script src="js/dragdrop.js"></script>
</body>
</html>
```

4. Load dragdrop.html in a browser. Figure 29.2 shows the working AngularJS application in action. As words are dragged and dropped onto the images, they are appended below the image. Also notice that the drag coordinates and drop status are displayed as well.


HTML5 Drag and Drop Components

Drag Status: none
Drop Status: none

HTML5 Drag and Drop Components

Drag Status: X: 279 Y: 262
Drop Status: Drag Over

FIGURE 29.2 Using custom AngularJS directives to provide drag-and-drop functionality in a web page.

Try it Yourself: Adding a Zoom View Field to Images

In this example, you use a custom AngularJS directive to replace the `<img>` element and provide an automatic zoom view field that is displayed next to the image on the page. When you click the image, the zoom view field will be updated with a zoomed-in portion of the image.

The purpose of this exercise is to show you how AngularJS custom directives can extend HTML with new elements that have a rich set of features. This example also illustrates another time when you will want to use the full version of jQuery rather than jQuery lite to get the size of the image and position of the mouse within the image.

Use the following steps to build the zoom view AngularJS application:

1. Create the following folder and file structure in your code:
   - **./lesson29/zooming.html:** AngularJS template for the project.
   - **./lesson29/zoomit.html:** AngularJS partial template that contains the image and zoom view field element definitions.
   - **./lesson29/js/zooming.js:** AngularJS application supporting the custom tabs directives.
2. Add the code in Listing 29.7 to the `zooming.js` file. This is an AngularJS

application that defines the custom AngularJS directive called `zoomit`. The `zoomit` directive is restricted to elements using only `restrict: 'E'`. Also note that the `src` attribute from the template definition is injected into the `scope`.

The functionality for the `zoomit` directive is in the `controller` function. Notice that an object is created called `zInfo` that contains the `background-image` and `background-position` properties. The `zInfo` scope value will be used to set the `ng-style` attribute for the zoom view field in the `zoomit.html` partial template in Listing 29.8. Setting the `background-image` and `background-position` attributes adds the image to the background and positions the zoom.

The `imageClick()` function suppresses the default click behavior and then gets the `event.target` element as a jQuery object. This is where we need the full version of jQuery to the `height`, `width`, and current `offset` of the image on the page. We can then calculate the percentage from the left as `posX` and from the top as `posY` of the mouse click and set the `background-position` style appropriately.

LISTING 29.7 zooming.js AngularJS Application That Defines a Custom AngularJS Directive Called `zoomit` That Implements an `<img>` Element with a Zoom View Field

Click here to view code image

```
angular.module('myApp', [])
.controller('myController', ['$scope', function($scope) {
}])
.directive('zoomit', function() {
  return {
    restrict: 'E',
    scope: { src: '@'},
    controller: function($scope) {
        $scope.zInfo = {
            "background-image": "url(" + $scope.src + ")",
            "background-position": "top right"
        };
        $scope.imageClick= function(event){
          event.preventDefault();
          //Using full jQuery to get offset, width and height
          var elem = angular.element(event.target);
          var posX = Math.ceil((event.pageX - elem.offset().left) /
                               elem.width() * 100);
          var posY = Math.ceil((event.pageY - elem.offset().top) /
                               elem.height() * 100);
          $scope.pos = posX + "% " + posY + "%";
          $scope.zInfo["background-position"] = posX + "% " +
                                                  posY + "%";
        };
      },
    link: function(scope, element, attrs) {
      },
    templateUrl: 'zoomit.html'
  };
});
```

3. Add the code in Listing 29.8 to the zoomit.html file. This code implements the `zoomit.html` partial template that adds the `<img>` element and a `<div>` element, which will have the zoomed image as a background. Notice that the `ng-click` method is set to the `imageClick()` function in the scope and passes the `$event`. Also notice that `ng-style` is set to `zInfo` in the scope.

LISTING 29.8 zoomit.html AngularJS Partial Template That Implements the `<img>` and `<div>` Elements for the Image and Zoom View Field

Click here to view code image

```
<div>
  <img src="{{src}}"
       ng-click="imageClick($event)"/>
  <div class="zoombox"
       ng-style="zInfo"></div>
</div>
```

4. Add the code in in Listing 29.9 to the zooming.html file. This code contains the AngularJS template code that provides the styles for the zoom view field and image. Notice that the `<zoomit>` element is added just like any other and that the `src` attribute is set as with an `<img>` element. Also note that the full jQuery library is loaded before the AngularJS library.

LISTING 29.9 zooming.html AngularJS Template That Styles and Implements the `<zoomit>` Custom AngularJS Directive

Click here to view code image

```
<!DOCTYPE html>
<html  ng-app="myApp">
  <head>
    <title>Magnify</title>
    <style>
```

```
      .zoombox {
        display: inline-block;
        border: 3px ridge black;
        width: 100px; height: 100px; }
      img {
        height: 200px;
        vertical-align: top; }
    </style>
  </head>
  <body>
  <h2>Image Zoom Window</h2>
  <div ng-controller="myController">
    <zoomit  src="/images/flower.jpg"></zoomit>
    <hr>
    <zoomit  src="/images/tiger.jpg"></zoomit>
  </div>
  </body>
  <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js">
</script>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
  <script src="js/zooming.js"></script>
</html>
```

5. Load the zooming.html file in a browser. Figure 29.3 shows the images with their view fields on a web page. As you click a particular point in the image, the zoom view field is updated.

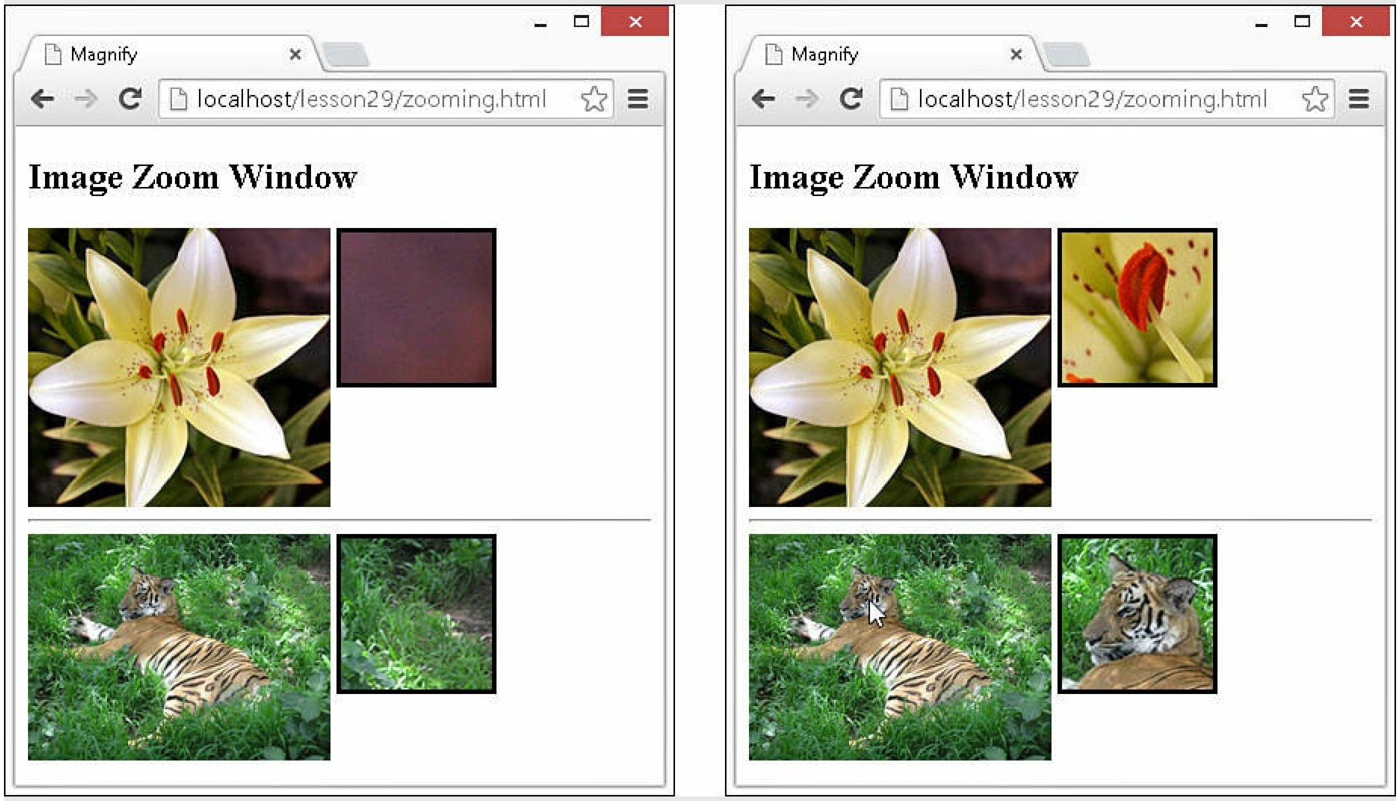

Magnify
localhost/lesson29/zooming.html
Image Zoom Window
Magnify
localhost/lesson29/zooming.html
Image Zoom Window

FIGURE 29.3 Implementing a custom AngularJS directive that provides an image with a zoom view finder.

Try it Yourself: Implementing Expandable and Collapsible Elements

In this example, you use custom AngularJS directives to build elements on the web page that can expand and contract. Each element will have a title and an expand/collapse button on top. When the collapse button is clicked, the contents of the element will be hidden. When the expand button is clicked, the contents will be shown again.

The purpose of this exercise is to solidify implementing custom AngularJS directives that nest inside each other and communicate with each other. In this example, you also get to see how a scope gets isolated from the controller but shared between the expand container and the items in the container.

Use the following steps to build the expandable/collapsible AngularJS application:

1. Create the following folder and file structure in your code:
   - **./server.js:** Node.js web server that serves the static project files.
   - **./images:** Folder that contains the images used in the examples.
   - **./lesson29:** Project folder.
   - **./lesson29/expand.html:** AngularJS template for the project that implements the custom expandable directives.
   - **./lesson29/expand_list.html:** AngularJS partial template for the expandable element directive.
   - **./lesson29/expand_item.html:** AngularJS partial template for each individual item in the expandable element.
   - **lesson29/js/expand.js:** AngularJS application supporting the expandable element directives.
2. Add the code in Listing 29.10 to the `expand.js` file. This is an AngularJS application that defines the `expandList` and `expandItem` custom AngularJS directives. The transclude option is used to keep the contents that get defined in the template.

   Note that in the `expandList` directive, the scope is an isolate but accepts the attributes `title` and `exwidth`, which are set to `title` and `listWidth` in the scope. Note that in line 27, the `listWidth` value is used to set the width of the style for items added to the expand list. Also, listWidth is used in line 34 to set the css attribute width for the expandable list.

The expandItem directive requires the expandList directive to provide access to the addItem() function in the expandList directive's scope. Note that the myStyle attribute is used to build a style object that will be set to the ng-style for the item in the expanded list.

The way that expanding and collapsing works is that the myHide value is bound to each item in the expanded list using ng-hide in the template shown in Listing 29.11. The items property in the scope of expandList provides a list of scopes for each of the expandItem elements that get added. Then when the expand/collapse button is clicked, it is a simple matter of setting the myHide value to true or false in the scope for each item in the collapse() function to show or hide the items in the expanded element.

LISTING 29.10 expand.js AngularJS Application That Implements the expandList and expandItem Custom Directive to Provide Expandable and Collapsible Elements

Click here to view code image

```
01 angular.module('myApp', [])
02 .controller('myController', ['$scope', function($scope) {
03   $scope.items = [1,2,3,4,5];
04 }])
05 .directive('expandList', function() {
06   return {
07     restrict: 'E',
08     transclude: true,
09     scope: {title: '@', listWidth: '@exwidth'},
10     controller: function($scope) {
11       $scope.collapsed = false;
12       $scope.expandHandle = "-";
13       items = $scope.items = [];
14       $scope.collapse = function() {
15         if ($scope.collapsed){
16           $scope.collapsed = false;
17           $scope.expandHandle = "-";
18         } else {
19           $scope.collapsed = true;
20           $scope.expandHandle = "+";
21         }
22         angular.forEach($scope.items, function(item) {
23           item.myHide = $scope.collapsed;
24         });
25       };
26       this.addItem = function(item) {
27         item.myStyle.width = $scope.listWidth;
```

```
        items.push(item);
        item.myHide=false;
      };
    },
    link: function(scope, element, attrs, expandCtrl) {
      element.css("display", "inline-block");
      element.css("width", scope.listWidth);
    },
    templateUrl: 'expand_list.html',
  };
})
.directive('expandItem', function() {
  return {
    require: '^expandList',
    restrict: 'E',
    transclude: true,
    scope: {},
    controller: function($scope){
        $scope.myHide = false;
        $scope.myStyle = { width: "100px", "display": "inline-block" };
      },
    link: function(scope, element, attrs, expandCtrl) {
      expandCtrl.addItem(scope);
    },
    templateUrl: 'expand_item.html',
  };
});
```

3. Add the code in Listing 29.11 to the expand_list.html file. This contains the AngularJS partial template `expand_list.html` that provides the definition for the `expandList` element. The elements for the expand list header, including the expand/collapse button and the title, are added. The `<div ng-transclude>` element is where the `expandItem` elements will be placed.

LISTING 29.11 expand_list.html AngularJS Partial Template That Defines the `expandList` Element

Click here to view code image

```
<div>
     <div class="expand-header">
          <span class="expand-button"
                ng-click="collapse()">{{expandHandle}}</span>
          {{title}}
     </div>
     <div ng-transclude></div>
</div>
```

4. Add the code in Listing 29.12 to the expand_item.html file. This contains the AngularJS partial template `expand_item.html` that provides the definition for the expandable items. Notice that `ng-hide` is set to `myHide` in the scope to expand/collapse the element. `ng-style` is set to `myStyle` so that we can set the width to the expand list width. The `expand-item` class enables us to easily change the item appearance using CSS. The `ng-transclude` is used to place the contents from the AngularJS template definition inside the list item.

LISTING 29.12 expand_item.html AngularJS Partial Template That Defines the `expandItem` Element

```
<div ng-hide="myHide"
     ng-style="myStyle"
     class="expand-item"
     ng-transclude>
</div>
```

5. Add the code in Listing 29.13 to the expand.html file. This implements the AngularJS template that provides the styles for the page as well as definitions for the `<expand-list>` elements. Notice that four `<expand-list>` elements are defined. The first is a simple list where the `<expand-item>` element contains just text. The second provides a single `<expand-item>` element with form elements. The third contains a mixture of different HTML elements in each `<expand-item>` element. The fourth one contains just an `<img>` element.

   Note that each `<expand-list>` element contains a different value for the attributes `title` and `exwidth`, which results in lists with different titles and widths on the page.

LISTING 29.13 expand.html AngularJS Code That Styles and Implements Expandable/Collapsible Elements Using the `expandList` and `expandItem` Custom Directives

Click here to view code image

```
<!DOCTYPE html>
<html  ng-app="myApp">
  <head>
    <title>Expandable and Collapsible Lists</title>
    <style>
      * { vertical-align: top; }
      expand-list{
        border: 2px ridge black; }
      .expand-header{
        text-align: center;
        font: bold 16px/24px arial;
         background-image: linear-gradient(#CCCCCC, #EEEEEE);
        }
      .expand-button{
        float: left; padding: 2px 4px;
        font: bold 22px/16px courier;
        color: white; background-color: black;
        cursor: pointer;
        border: 3px groove grey; }
      .expand-item {
        border: 1px ridge black;}
      p { margin: 0px; padding: 2px;}
      label { display: inline-block; width: 80px; padding: 2px; }
      .small { width: 100px; padding: 2px; }
      .large { width: 300px; }
    </style>
  </head>
  <body>
  <h2>Expandable and Collapsible Lists</h2>
  <hr>
  <div ng-controller="myController">
    <expand-list title="Companion" exwidth="120px">
      <expand-item>Rose</expand-item>
      <expand-item>Donna</expand-item>
      <expand-item>Martha</expand-item>
      <expand-item>Amy</expand-item>
      <expand-item>Rory</expand-item>
    </expand-list>
    <expand-list title="Form" exwidth="280px">
      <expand-item>
        <label>Name</label>
        <input type="text" /><br>
        <label>Phone</label>
        <input type="text" /><br>
        <label>Address</label>
        <input type="text" /><br>
        <label>Comment</label>
        <textarea type="text"></textarea>
      </expand-item>
    </expand-list>
    <hr>
    <expand-list title="Mixed List" exwidth="300px">
      <expand-item>Text Item</expand-item>
      <expand-item><p>I think therefore I am.</p></expand-item>
      <expand-item>
        <img class="small" src="/images/jump.jpg" />Sunset
      </expand-item>
      <expand-item>
        <ul>
          <li>AngularJS</li>
          <li>jQuery</li>
          <li>JavaScript</li>
        </ul>
      </expand-item>
    </expand-list>
    <expand-list title="Image" exwidth="300px">
      <expand-item>
        <img class="large" src="/images/falls.jpg" />
      </expand-item>
    </expand-list>
  </div>
  </body>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js">
</script>
  <script src="js/expand.js"></script>
</html>
```

6. Load the expand.html file in a browser. The results of the AngularJS application are shown in Figure 29.4. Notice the expanded and collapsed version of the elements.

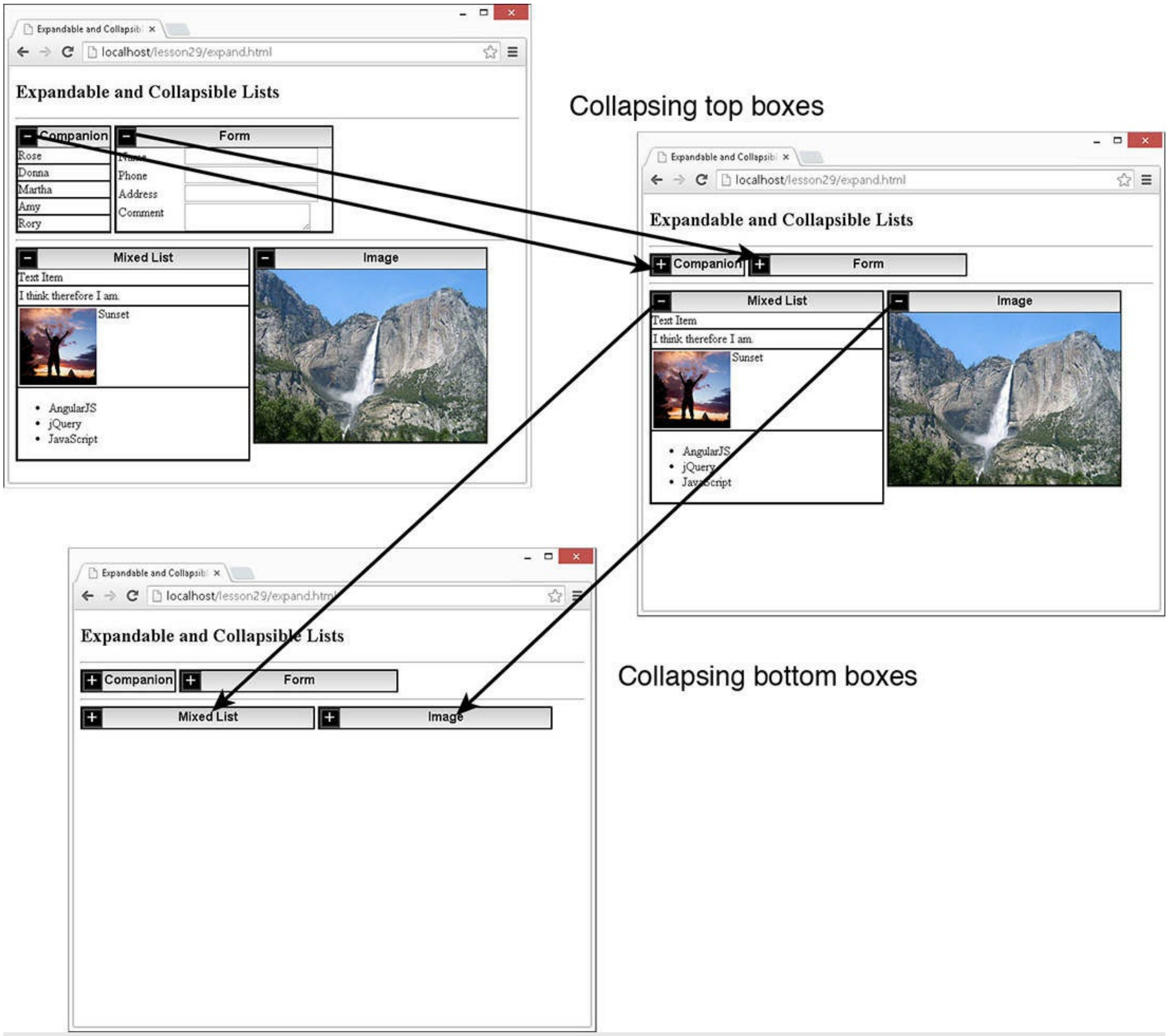


FIGURE 29.4 Using custom AngularJS directives to build and implement expandable/collapsible web page elements.

Try it Yourself: Adding Star Ratings to Elements

In this example, you use just the AngularJS scope, controller, and view to implement elements that implement the star ratings for images. When you click a star, the rating changes in the scope and the number of stars changes.

The purpose of this exercise is to remind you that much of the data binding and view interactions can be accomplished in basic AngularJS templates without the need for custom directives.

Use the following steps to build the expandable/collapsible AngularJS application:

1. Create the following folder and file structure in your code:
   - **./lesson29:** Project folder.
   - **./lesson29/rating.html:** AngularJS template for the project that implements a simple star rating to elements.
   - **./lesson29/js/rating.js:** AngularJS application that defines the supporting star rating elements.
2. Add the code in Listing 29.14 to the `rating.js` file. This implements the AngularJS application we use to change the star ratings. Notice that the data used comes from `$scope.items`. This data could have come from a service, a database, or another source. The array `$scope.stars` is used in the template to display the stars on the web page. The only function required in the controller code is `adjustRating`, which is called when the user changes the rating by clicking a star.

LISTING 29.14 rating.js AngularJS Application That Provides the Data and Functionality to Support Star Ratings in the View

Click here to view code image

```
angular.module('myApp', [])
.controller('myController', ['$scope', function($scope) {
  $scope.stars = [1,2,3,4,5];
  $scope.items = [
      {
        description: "Mysty Mountains",
        img: "/images/misty_mountains.jpg",
        rating: 3},
      {
        description: "Wheel",
        img: "/images/wheel.jpg",
        rating: 4},
      {
        description: "Pool",
        img: "/images/pool.jpg",
        rating: 4}
    ];
  $scope.adjustRating = function(item, value){
    item.rating = value;
  };
}]);
```

3. Add the code in Listing 29.15 to the rating.html file. This implements an AngularJS template that iterates through the `items` array from the scope and

builds out the image elements complete with the description and star rating. Note that to build the star list, `ng-repeat` is used on the `stars` array from the scope. Also note that lines 33 and 34 determine whether a filled star or an empty star is displayed. To do that, the `ng-class` attribute is set based on the item `rating` being greater than the star index. The `ng-click` attribute is used to bind mouse clicks on each star to the `adjustRating()` function in the scope to set the rating for this item.

LISTING 29.15 rating.html AngularJS Template That Utilizes Data from the Scope to Display a List of Images with Descriptions and Ratings

Click here to view code image

```
01 <!DOCTYPE html>
02 <html  ng-app="myApp">
03   <head>
04     <title>Ratings</title>
05     <style>
06       img {
07         width: 200px; }
08       .star {
09         display: inline-block;
10         width: 15px;
11         background-image: url("/images/star.png");
12         background-repeat: no-repeat;
13         background-size: 15px 15px;
14       }
15       .empty {
16         display: inline-block;
17         width: 15px;
18         background-image: url("/images/empty.png");
19         background-repeat: no-repeat;
20         background-size: 15px 15px;
21       }
22     </style>
23   </head>
24   <body>
25   <h2>Images With Ratings</h2>
26   <hr>
27   <div ng-controller="myController">
28     <div ng-repeat="item in items">
29       <img ng-src="{{item.img}}" />
30       {{item.description}}<br>
31       Rating: {{item.rating}} stars<br>
32       <span ng-repeat="idx in stars"
33             ng-class=
34               "{true: 'star', false: 'empty'}[idx <= item.rating]"
35             ng-click="adjustRating(item, idx)">&nbsp;
36       </span>
37       <hr>
```

```
    </div>
  </div>
  </body>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js">
</script>
  <script src="js/rating.js"></script>
</html>
```

4. Load rating.html in a web browser. The resulting web page is shown in Figure 29.5. Notice that as the stars are clicked, the rating and stars displayed also change.

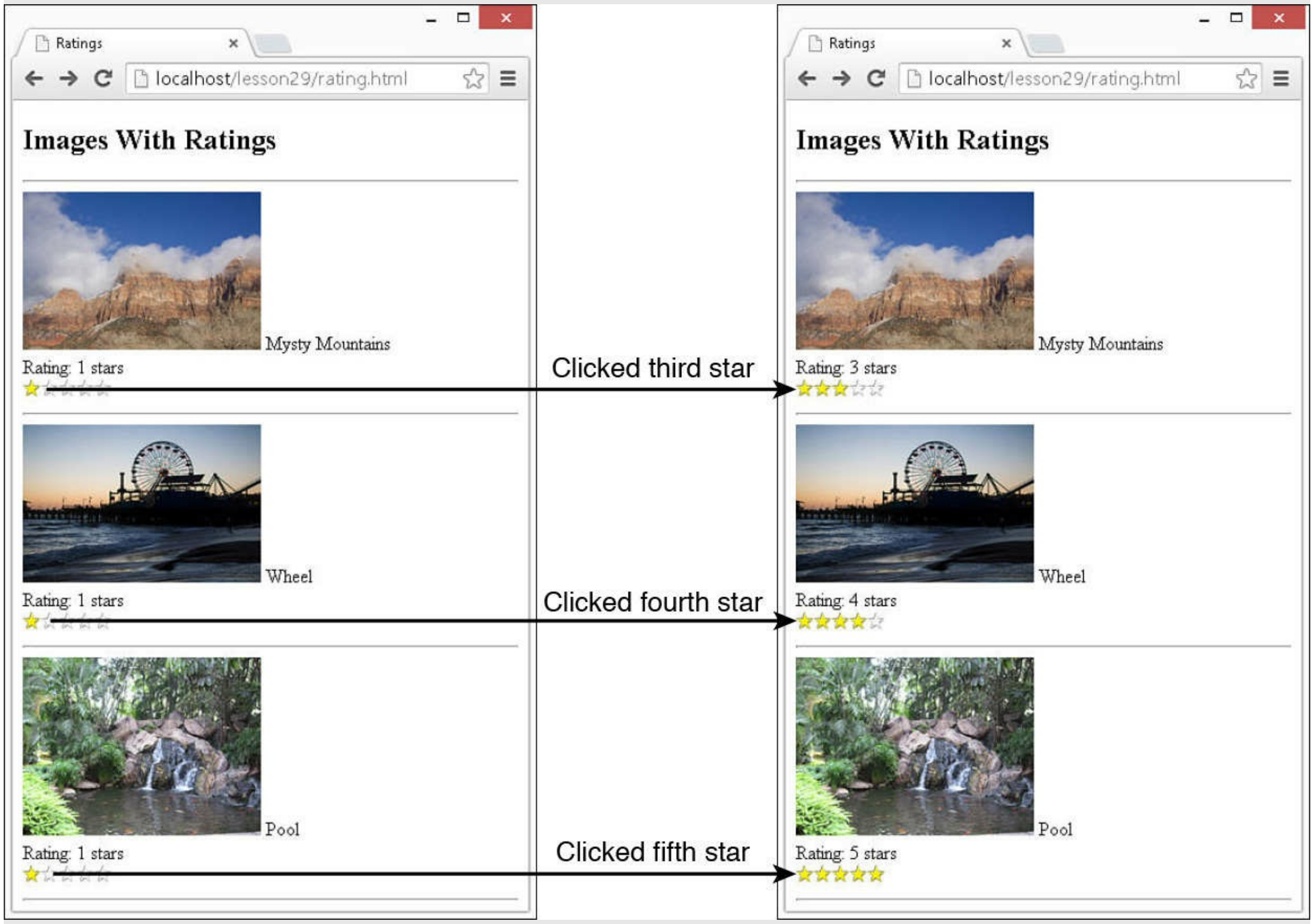


FIGURE 29.5 Implementing a simple star rating in elements using AngularJS scope data, controller code, and a template view.

Summary

AngularJS provides many tools to extend the capability of HTML through the use of templates with built-in and custom directives. In this lesson, you saw different ways in which you can implement richly interactive elements into your web pages in an easy yet structured way.

In this lesson, you also got a look at several custom directives. The tabbed panels and expandable/collapsible examples showed ways to implement directives that nest inside each other and interact with each other. The drag and drop showed you one way to interact with the HTML5 drag and drop events, as well as how to share the scope between the controller and custom directives. The star rating example showed you how to build interactivity using only the AngularJS scope, controller, and template. The zoom view field example showed you how to extend the concept of an HTML `<img>` element to an element that includes an additional zoom element that interacts with mouse clicks in the image.

That's a wrap for this book. We hope that you have enjoyed learning about AngularJS, jQuery, and JavaScript as much as we have. We love these technologies because they allow you to easily create fantastic, dynamic, and highly interactive web pages and web applications. These technologies are powerful tools because they allow you to create well-structured, reusable code in a fast and efficient way. After reading this book, you should have a good understanding of JavaScript, jQuery, and the AngularJS framework so that you feel confident in jumping in and writing your own web applications. Enjoy your coding!

Q&A

Q. Where is a good source to learn additional functionality of AngularJS?

A. The best and most up-to-date source is the AngularJS website angularjs.org. For simple things, you can look at what w3schools.com has to offer.

Workshop

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

Quiz

1. Which AngularJS directive can you use to hide or show an element?
2. Which AngularJS directive is used to mark the insertion point for transcluded DOM elements in a directive that uses transclusion?
3. What is the difference between `template` and `templateUrl` in a custom directive definition?

Quiz Answers

1. `ng-show`
2. `ng-transclude`

3. The `template` setting contains an actual HTML snippet, whereas the `templateUrl` setting contains the URL path for the template to use in the directive.

Exercises

1. Modify the code in Listings 29.1, 29.2, 29.3, and 29.4 to include some additional tabs with different content.

2. Modify the code in Listings 29.7, 29.8, and 29.9 to also display the current page coordinates of the mouse in the custom zoom area.

Index

Code Snippets

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

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

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Server Side Script</title>
    <meta charset="utf-8"/>
  </head>
  <body>
    <ul>
      <li>Mercury</li>
      <li>Venus</li>
      <li>Earth</li>
      <li>Mars</li>
    </ul>
  </body>
</html>
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Style</title>
    <meta charset="utf-8" />
    <style>
      li {
        text-align: center;
        font-family: "Times New Roman", Times, serif;
        font-size: 30px;
        font-style: italic;
        font-weight: bold;
        list-style-image: url('/images/check.png');
        list-style-position: inside;
      }
    </style>
  </head>
  <body>
    <ul>
      <li>Mercury</li>
      <li>Venus</li>
      <li>Earth</li>
      <li>Mars</li>
    </ul>
  </body>
</html>
```

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

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

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

```
<!DOCTYPE html>
<html>
  <head>
    <title>AJAX</title>
    <meta charset="utf-8" />
    <script>
      var xmlhttp = new XMLHttpRequest();
      function loadPlanets(){
        xmlhttp.open("GET","/lesson01/data.html",false);
        xmlhttp.send();
        var planets = JSON.parse( xmlhttp.responseText );
        var ulElement = document.getElementById("planetList");
        for (var planet in planets){
          var listItem = ulElement.appendChild(document.createElement("li"));
          listItem.appendChild(document.createTextNode(planets[planet]));
        }
      }
    </script>
  </head>
  <body onload="loadPlanets()">
    <ul id="planetList">
    </ul>
  </body>
</html>
```

```
Error: ENOENT, s≠tat 'C:\Users\Brad\AppData\Roaming\npm'
node <path_to_JavaScript_file>
```

```
var express = require('express');
var app = express();
app.use('/', express.static('./'));
app.listen(80);
```

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

```
http://localhost/lesson01/first.html
```

```
http://localhost/lesson01/first.html
```

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

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
```

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

```
http://localhost/lesson01/first.html
```

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script>
      function moveIt(){
        var coords = new Array(10,50,100,130,175,225,260,300,320,350);
        var x = coords[Math.floor((Math.random()*10))];
        var y = coords[Math.floor((Math.random()*10))];
        $("#elusiveText").css({"top": x + "px", "left": y + "px"})
      }
    </script>
    <style>
      span{
        background-color: #0066AA;
        color: #FFFFFF;
        font-weight: bold;
        border-color: #C0C0C0;
        border:2px solid;
        border-radius:5px;
        padding: 3px;
        position:absolute;
        top:150px;
        left:100px;
      }
    </style>
  </head>
  <body>
    <span id="elusiveText" onmouseover="moveIt()">Click Me</span>
  </body>
</html>
```

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

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

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

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

```
http://localhost/lesson02/js_errors.html
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1><i></i>Favorite Movies</h1>
    <ul>
      <ll>Lord of the Rings</li>
      <li>Harry Potter</li>
      <li>Narnia</li>
      <li>Hot Lead and Cold Feet</li>
    </ul>
  </body>
</html>
```

http://localhost/lesson02/html_errors.html

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

http://localhost/lesson02/css_errors.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script>
      function incCount(){
        var cnt = 1;
        cnt += 1;
        return cnt;
      }
      function countIt(){
        $("#counter").html(incCount);
      }
    </script>
    <style>
      #clicker{
        background-color: #0066AA;
        color: #FFFFFF;
        font-weight: bold;
        border:2px solid, #C0C0C0;
        width: 65px;
      }
    </style>
  </head>
  <body>
    <div id="clicker" onclick="countIt()">Click Me</div>
    <div id="counter">1</div>
  </body>
</html>
```

```
http://localhost/lesson02/debug.html
```

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

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.
w3.org/TR/html4/loose.dtd">
```

```
<p align="center">This is some centered text.</p>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Page</title>
    <meta charset="UTF-8" />
    <script>
      function appendTitle(newTitle){
        document.title += newTitle;
      }
    </script>
  </head>
  <body onload="appendTitle('... Loaded')">
  </body>
</html>
```

```
<meta http-equiv="content-type" content="text/html; charset=UTF-8">
```

```
<meta name="keywords" content="HTML, CSS, jQuery, JavaScript">
```

```
<meta http-equiv="refresh" content="300">
```

```
<html>
  <head>
    <meta charset="UTF-8">
    <style type="text/css">
      h1 {
        background-color:black;
        color:white;
      }
    </style>
  </head>
  <body>
    <h1>CSS Reversed Text</h1>
  </body>
</html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script>
      function reverseText(){
        $("h1").css("background-color", "black");
        $("h1").css("color", "white");
      }
    </script>
  </head>
  <body onload="reverseText()">
    <h1>JavaScript Reversed Text</h1>
  </body>
</html>
```

```
<head>
  <script src="DynamicPage.js"></script>
  <noscript>
    <h3>This web page uses JavaScript but it is disabled in your browser.
        The web page may not function properly.</h3>
  </noscript>
</head>
```

```
<head>
  <link rel="stylesheet" type="text/css" href="styleA.css">
  <link rel="stylesheet" type="text/css" href="styleB.css">
</head>
```

```
<link rel="icon" type="image/png" href="http://example.com/myicon.png">
```

```
<div id="Div1">
  <p id="Div1Heading" class="heading" style="font-weight:bold">Heading Text</p>
  <p id ="Div1Content" class ="content">Some Content</p>
</div>
<div id="Div2">
  <p id="Div2Heading" class="heading">Another Heading Text</p>
  <p id ="Div2Content" class ="content">Some More Content</p>
</div>
```

```
<span style="{display:block;}">Span Text</span>
```

```
<span>Text on</span>
<span> Line 1</span><br>
<span>Text on Line 2</span><hr>
<span>New section on Line 3</span>
```

```
<div class="heading">
  <p>Heading A</p>
  <p>Heading B</p>
  <p>Heading C</p>
</div>
```

```
    <div class="content">
      <p>Paragraph A</p>
      <p>Paragraph B</p>
      <p>Paragraph C</p>
    </div>
```

```
http://localhost/lesson03/css_styling.html
```

```
        .content p{
          background-color:#C0C0C0;
          padding: 3px;
        }
```

```
.heading p{
  display: inline;
  background-color:black;
  color: white;
  font-weight:bold;
  padding:3px;
}
```

```html
<html>
  <head>
    <title>CSS Styling</title>
    <meta charset="UTF-8">
    <style type="text/css">
      .content p{
        background-color:#C0C0C0;
        padding: 3px;
      }
      .heading p{
        display: inline;
        background-color:black;
        color: white;
        font-weight:bold;
        padding:3px;
      }
    </style>
  </head>
  <body>
    <div class="heading">
      <p>Heading A</p>
      <p>Heading B</p>
      <p>Heading C</p>
    </div>
    <div class="content">
      <p>Paragraph A</p>
      <p>Paragraph B</p>
      <p>Paragraph C</p>
    </div>
  </body>
</html>
```

```
<a href="#local link">Link to Some Text Below</a>
```

```
<a id="local_link">Page Anchor</a>
```

```
<img src="/images/peak.jpg" height="200px"/>
<img src="/images/peak.jpg" width="200px"/>
<img src="/images/peak.jpg" height="200px" width="200px"/>
```

```
    <table border=1></table>
```

```
09      <caption>Favorite World Sites</caption>
```

```
    <thead>
      <th>Monument</th>
      <th>location</th>
      <th colspan=2>century</th>
    </thead>
```

```
  <tr>
    <th>Delicate Arch</th>
    <td>Utah</td>
    <td>March 1, 1872</td>
    <td><img src="/images/arch.jpg" width="100" />
  </tr>
```

```
http://localhost/lesson03/tables.html
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Tables</title>
  <meta Charset="UTF-8">
</head>
<body>
  <table border=1>
    <caption>Favorite World Sites</caption>
    <thead>
      <th>Monument</th>
      <th>location</th>
      <th colspan=2>century</th>
    </thead>
    <tr>
      <th>Delicate Arch</th>
      <td>Utah</td>
      <td>March 1, 1872</td>
      <td><img src="/images/arch.jpg" width="100" />
    </tr>
    <tr>
      <th>Washington Monument</th>
      <td>Washington D.C.</td>
      <td>March 1, 1872</td>
      <td><img src="/images/washington.jpg" width="100" />
    </tr>
    <tr>
      <th>Tikal</th>
      <td>Guatemala</td>
      <td>November 19, 1919</td>
      <td><img src="/images/pyramid.jpg" width="100" />
    </tr>
  </table>
</body>
</html>
```

```
<input type="text" disabled />
```

```
 <fieldset>
    <legend>Contact Info:</legend>
    Name: <input type="text" name="name">
      Origin: <input type="text" name="address"><br>
      Destination: <input type="text" name="city">
      Flight Date: <input type="text" name="zip"><br>
  <select name="Airline">
    <option value="DA">Delta</option>
    <option value="PJ">Private Jet</option>
    <option value="Heli">Helicopter</option>
    <option value="EG">Eagles</option>
  </select>
</fieldset>
```

```
<input id="One WayRB" type="radio"
       name="travelPlan" value="One way">
<label for="One WayRB">One Way</label>
<input id="Round TripRB" type="radio"
       name="travelPlan" value="round trip">
<label for="Round TripRB">Round Trip</label><br>
```

```
      <textarea rows="10" cols="30">comments</textarea><br>
```

```
<input di='newsCB' type="checkbox" name="news" value="news">
  <label for="newsCB">
    I would like to enroll into the frequent flyer miles program
  </label><br>
```

```
      <button type="submit" name="SubmitButton"
              value="Submit">Submit</button>
      <button type="reset" name="ResetButton"
              value="Reset">Reset</button>
```

```
http://localhost/lesson03/forms.html
```

```
<html>
  <head>
    <title>Forms</title>
    <meta charset="UTF-8">
  </head>
  <body>
    <form>
       <fieldset>
          <legend>Contact Info:</legend>
          Name: <input type="text" name="name">
            Origin: <input type="text" name="address"><br>
            Destination: <input type="text" name="city">
            Flight Date: <input type="text" name="zip"><br>
        <select name="Airliner">
          <option value="DA">Delta</option>
          <option value="PJ">Private Jet</option>
          <option value="Heli">Helicopter</option>
          <option value="EG">Eagles</option>
        </select>
      </fieldset>
      <input id="One WayRB" type="radio"
             name="travelPlan" value="One way">
      <label for="One WayRB">One Way</label>
      <input id="Round TripRB" type="radio"
             name="travelPlan" value="round trip">
      <label for="Round TripRB">Round Trip</label><br>
      <textarea rows="10" cols="30">comments</textarea><br>
      <input di='newsCB' type="checkbox" name="news" value="news">
        <label for="newsCB">
          I would like to enroll into the frequent flyer miles program
        </label><br>
      <button type="submit" name="SubmitButton"
              value="Submit">Submit</button>
      <button type="reset" name="ResetButton"
              value="Reset">Reset</button>
    </form>
  </body>
</html>
```

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     height="200">
  <ellipse   cx="100" cy="100" rx="80" ry="50"
             stroke="black" stroke-width="2"
             fill="transparent" />
  <ellipse   cx="100" cy="100" rx="50" ry="80"
             stroke="black" stroke-width="2"
             fill="transparent"
             transform="rotate(30, 100, 100)" />
  <ellipse   cx="100" cy="100" rx="80" ry="50"
             stroke="black" stroke-width="2"
             fill="transparent"
             transform="rotate(60, 100, 100)" />
  <ellipse   cx="100" cy="100" rx="8" ry="2"
             stroke="crimson" stroke-width="2"
             fill="crimson"
             transform="rotate(45, 100, 100)" />
  <ellipse   cx="100" cy="100" rx="2" ry="8"
             stroke="crimson" stroke-width="2"
             fill="crimson"
             transform="rotate(45, 100, 100)" />
  <ellipse   cx="100" cy="100" rx="2" ry="8"
             stroke="crimson" stroke-width="2"
             fill="crimson" />
  <ellipse   cx="100" cy="100" rx="8" ry="2"
             stroke="crimson" stroke-width="2"
             fill="crimson" />
  <circle    cx="100" cy="100" r="6"
        stroke="red" stroke-width="4" fill="DarkRed"/>
</svg>
```

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     height="200">
  <path d="M100,90 h-50 a50,50 0 1,0 50,-50 z"
        fill="none" stroke="purple" stroke-width="1" />
  <path d="M90,80 v-50 a50,50 0 0,0 -50,50 z"
        fill="violet" stroke="none" stroke-width="2" />
</svg>
```

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1"
     height="350">
  <defs>
    <path id="path1"
          d="M 77,210 a 1,1 0 1,1 200,0 a 1,1 0 1,1 -200,1"/>
  </defs>
  <text x="10" y="10" style="fill:blue;font-size:31px;">
    <textPath xlink:href="#path1">
      Teach Yourself AngularJS JavaScript and jQuery
    </textPath>
  </text>
  <path d="M 175,130 v90 h60" stroke="black"
        stroke-width="5" fill="none"/>
</svg>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>html 5 graphics</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
         height="200">
      <ellipse   cx="100" cy="100" rx="80" ry="50"
                 stroke="black" stroke-width="2"
                 fill="transparent" />
      <ellipse   cx="100" cy="100" rx="50" ry="80"
                 stroke="black" stroke-width="2"
                 fill="transparent"
                 transform="rotate(30, 100, 100)" />
      <ellipse   cx="100" cy="100" rx="80" ry="50"
                 stroke="black" stroke-width="2"
                 fill="transparent"
                 transform="rotate(60, 100, 100)" />
      <ellipse   cx="100" cy="100" rx="8" ry="2"
                 stroke="crimson" stroke-width="2"
                 fill="crimson"
                 transform="rotate(45, 100, 100)" />
      <ellipse   cx="100" cy="100" rx="2" ry="8"
                 stroke="crimson" stroke-width="2"
                 fill="crimson"
                 transform="rotate(45, 100, 100)" />
      <ellipse   cx="100" cy="100" rx="2" ry="8"
                 stroke="crimson" stroke-width="2"
                 fill="crimson" />
      <ellipse   cx="100" cy="100" rx="8" ry="2"
                 stroke="crimson" stroke-width="2"
                 fill="crimson" />
      <circle    cx="100" cy="100" r="6"
            stroke="red" stroke-width="4" fill="DarkRed"/>
    </svg>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
         height="200">
      <path d="M100,90 h-50 a50,50 0 1,0 50,-50 z"
            fill="none" stroke="purple" stroke-width="1" />
      <path d="M90,80 v-50 a50,50 0 0,0 -50,50 z"
            fill="violet" stroke="none" stroke-width="2" />
    </svg>
    <svg xmlns="http://www.w3.org/2000/svg" version="1.1"
         height="350">
      <defs>
        <path id="path1"
              d="M 77,210 a 1,1 0 1,1 200,0 a 1,1 0 1,1 -200,1"/>
      </defs>
      <text x="10" y="10" style="fill:blue;font-size:31px;">
        <textPath xlink:href="#path1">
          Teach Yourself AngularJS JavaScript and jQuery
        </textPath>
      </text>
      <path d="M 175,130 v90 h60" stroke="black"
            stroke-width="5" fill="none"/>
    </svg>
  </body>
</html>
```

```
        var c=document.getElementById("myCanvas");
        var ctx=c.getContext("2d");
        ctx.lineWidth="1";
        ctx.strokeStyle="blue";
```

```
var grd=ctx.createLinearGradient(100,50,100,5);
grd.addColorStop(0,"blue");
grd.addColorStop(1,"white");
ctx.fillStyle=grd;
```

```
<html>
  <head>
    <title>HTML 5 Canvas</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <canvas id="myCanvas" width="300" height="300">
      Sorry Your Browser Doesn't Support HTML5 Canvas
    </canvas>
    <script>
      var c=document.getElementById("myCanvas");
      var ctx=c.getContext("2d");
      ctx.lineWidth="1";
      ctx.strokeStyle="blue";
      //top
      ctx.beginPath();
      var grd=ctx.createLinearGradient(100,50,100,5);
      grd.addColorStop(0,"blue");
      grd.addColorStop(1,"white");
      ctx.fillStyle=grd;
      grd.addColorStop(0,"blue");
      ctx.moveTo(1,25);
      ctx.lineTo(100,5);
      ctx.lineTo(200,25);
      ctx.lineTo(100,50);
      ctx.fill();
      ctx.stroke();
      //left
      ctx.beginPath();
      var grd=ctx.createLinearGradient(75,100,60,25);
      grd.addColorStop(0,"red");
      grd.addColorStop(1,"white");
      ctx.fillStyle=grd;
      ctx.moveTo(1,25);
      ctx.lineTo(100,50);
      ctx.lineTo(100,165);
      ctx.lineTo(1,125);
      ctx.lineTo(1,25);
      ctx.fill();
      ctx.stroke();
      //right
      ctx.beginPath();
      var grd=ctx.createLinearGradient(200,50,125,175);
      grd.addColorStop(0,"yellow");
      grd.addColorStop(1,"white");
      ctx.fillStyle=grd;
      ctx.moveTo(100,50);
      ctx.lineTo(200,25);
      ctx.lineTo(200,125);
      ctx.lineTo(100,165);
      ctx.fill();
      ctx.stroke();
    </script>
  </body>
</html>
```

```html
<video width="320" height="240" controls>
  <source src="images/movie.mp4" type="video/mp4">
  <source src="images/movie.ogg" type="video/ogg">
  Sorry, your browser does not support the video tag.
</video>
<audio controls>
  <source src="song.mp3" type="audio/mp3">
  Sorry, your browser does not support the audio element.
</audio>
```

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

```
<html>
  <head>
    <meta charset="UTF-8">
    <style type="text/css">
      p{
        background-color:#C0C0C0;
        padding: 3px;
      }
      span{
        font-weight:bold;
      }
    </style>
  </head>
...
```

```
<span style="background-color:blue; color:white font-weight:bold">Styled Text
</span>
```

```
p { font-style:italic; background-color:#DDDDDD; width:250px; }
```

```
font:italic bold 12px "Times New Roman",serif;
```

```
-ms-transform:rotate(5deg);          /* IE 9 */
-moz-transform:rotate(5deg);          /* Firefox */
-webkit-transform:rotate(5deg);     /* Safari and Chrome */
-o-transform:rotate(5deg);          /* Opera */
```

```
p {color:blue;}
p {color:#0000FF;}
p {color:rgb(0,0,255);}
p {color:rgba(0,0,255,1.0);}
p {color:hsl(240,100%,50%);}
p {color:hsl(240,100%,50%,1.0);}
```

```
#roman{font: italic bold 30px "Times New Roman", serif}
#roman{
  font-family:"Times New Roman", serif;
  font-style:italic;
  font-weight:bold;
  font-size: 30px;
  }
```

```
#align-left{text-align:left}
#align-center{text-align:center}
#align-right{text-align:right}
```

```
#underline{text-decoration:underline;}
#strike-through{text-decoration:line-through;}
```

```
#overflow{text-overflow:" (more)";}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Text Styles</title>
    <meta charset="UTF-8">
...
  </head>
  <body>
...
  </body>
</html>
```

```html
    <p id="plain">Plain Text</p>
    <p id="indent">Indented Text <br>Indented Line 2</p>
    <p id="blue">Blue Text</p>
    <p id="tight">Tight Text</p>
    <p id="half-height">Close Text Line 1<br>
                      Close Text Line 2</p>
    <p id="align-left">Left Text</p>
    <p id="align-center">Centered Text</p>
    <p id="align-right">Right Text</p>
    <p id="underline">Underlined Text</p>
    <p id="strike-through">Strike Through Text</p>
    <p id="first-cap">capitalize the first letter</p>
    <p id="uppercase">change to upper case</p>
    <p id="blackadder">Some BlackAdder Text</p>
    <p id="roman">Some Times New Roman Text</p>
```

```
#blue{color:blue;}
#tight{letter-spacing:-1px}
#half-height{line-height:50%}
#align-left{text-align:left}
#align-center{text-align:center}
#align-right{text-align:right}
#indent{text-indent:50px;}
#underline{text-decoration:underline;}
#strike-through{text-decoration:line-through;}
#first-cap{text-transform:capitalize;}
#uppercase{text-transform:uppercase;}
```

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

http://localhost/lesson04/text_styles.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Text Styles</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="css/text_styles.css">
  </head>
  <body>
    <p id="plain">Plain Text</p>
    <p id="indent">Indented Text <br>Indented Line 2</p>
    <p id="blue">Blue Text</p>
    <p id="tight">Tight Text</p>
    <p id="half-height">Close Text Line 1<br>
                      Close Text Line 2</p>
    <p id="align-left">Left Text</p>
    <p id="align-center">Centered Text</p>
    <p id="align-right">Right Text</p>
    <p id="underline">Underlined Text</p>
    <p id="strike-through">Strike Through Text</p>
    <p id="first-cap">capitalize the first letter</p>
    <p id="uppercase">change to upper case</p>
    <p id="blackadder">Some BlackAdder Text</p>
    <p id="roman">Some Times New Roman Text</p>
  </body>
</html>
```

```
#blue{color:blue;}
#tight{letter-spacing:-1px}
#half-height{line-height:50%}
#align-center{text-align:center}
#align-right{text-align:right}
#indent{text-indent:50px;}
#underline{text-decoration:underline;}
#strike-through{text-decoration:line-through;}
#first-cap{text-transform:capitalize;}
#uppercase{text-transform:uppercase;}
#roman{font: italic bold 30px "Times New Roman", serif}
#blackadder{
  font-family:"blackadder itc";
  font-size: 25px;
  font-weight:bold;
  }
```

```
#top-left {background-color:rgb(255,255,0);}
```

```
#top-left {background-size:200px 100px;}
#top-left {background-position:20% 100%;}
#top-left {background-position:contain;}
```

```
#top-left {background-position:center center;}
#top-left {background-position:10% 50%;}
#top-left {background-position:100px 20px;}
```

```
#rocs{background-image:url("images/rocks.png");}
```

```
background-image: -moz-linear-gradient(top, blue 0%, white 70%, green 100%);
```

```
background-image: -moz-radial-gradient(50% 50%, circle contain,white,blue
75%,rgba(255,255,255,0));
```

```
body{
  background-size:50px 50px;
  margin:5px;
  background-image: -moz-radial-gradient(20px 20px, circle contain,white,blue
➥75%,white);
  background-repeat:repeat;
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Backgrounds</title>
    <meta charset="UTF-8">
    <style>
...
  </head>
  <body>
    <div id="heading">jQuery Rules!</div>
    <div id="content">
      <div id="menu">
        <span>Home</span>
        <span>Info</span>
        <span>Examples</span>
      </div>
      <p>Page Content</p>
    </div>
  </body>
</html>
```

```
body {
  margin:0px;
  background-image:url("/images/ruler.png");
  background-repeat:repeat-y;
}
```

```
#heading {
  background-image: -moz-linear-gradient(left, #294551 0%, #AACCFF
➥100%);
  background-image: -webkit-linear-gradient(left, #294551 0%, #AACCFF
➥100%);
  background-image: -ms-linear-gradient(left, #294551 0%, #AACCFF
➥100%);
  height:200px;
  font:150px bold;
  text-align: center;
  color:rgba(255,255,255,.4);
}
```

```
#menu{
  padding:2px;
  background-color:#294551;
}
```

```
#menu{
  padding:3px;
  background-color:#555555;
}
span {
  padding:3px;
  background-image: -moz-linear-gradient(top, #294551 0%, #AACCFF
➥85%, #0022ff 100%);
  background-image: -webkit-linear-gradient(top, #294551 0%, #AACCFF
➥85%, #294551 100%);
  background-image: -ms-linear-gradient(top, #294551 0%, #AACCFF 85%,
➥#294551 100%);
  font:20px bold;
  color:white;
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Backgrounds</title>
    <meta charset="UTF-8">
    <style>
      body {
        margin:0px;
        background-image:url("/images/ruler.png");
        background-repeat:repeat-y;
      }
      #heading {
        background-image: -moz-linear-gradient(left, #294551 0%, #AACCFF 100%);
        background-image: -webkit-linear-gradient(left, #294551 0%, #AACCFF
➥100%);
        background-image: -ms-linear-gradient(left, #294551 0%, #AACCFF 100%);
        height:200px;
        font:150px bold;
        text-align: center;
        color:rgba(255,255,255,.4);
      }
      #content {
        margin-left:200px;
      }
      #menu{
        padding:2px;
        background-color:#294551;
      }
      span {
        padding:3px;
        background-image: -moz-linear-gradient(top, #294551 0%, #AACCFF 85%,
➥#0022ff 100%);
        background-image: -webkit-linear-gradient(top, #294551 0%, #AACCFF 85%,
➥#294551 100%);
        background-image: -ms-linear-gradient(top, #294551 0%, #AACCFF 85%,
➥#294551 100%);
        font:20px bold;
        color:white;
      }
    </style>
  </head>
  <body>
    <div id="heading">jQuery Rules!</div>
    <div id="content">
      <div id="menu">
        <span>Home</span>
        <span>Info</span>
        <span>Examples</span>
      </div>
      <p>Page Content</p>
    </div>
  </body>
</html>
```

```
property: top right bottom left;
border-width: 1px 2px 1px 2px;
border-color: red blue red blue;
```

```
div { width:200px; margin:5px; padding:2px; text-align:center}
```

```
#round { border: 1px solid; border-radius:5px;}
#veryround { border: 1px solid; border-radius:50%;}
```

```
#mixed {
  border-top: 1px solid;
  border-top-left-radius:5px;
  border-left: 1px dotted;
  border-bottom: 5px ridge;
  border-bottom-right-radius: 10px;
}
```

```
#button {
   width:150px;
   background-color: #2233FF;
   color: white;
   border:5px outset blue;
   border-radius:50%;
}
```

```html
<html>
  <head>
    <title>Borders</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="css/borders.css">
  </head>
  <body>
    <div id="simple">Simple Border</div>
    <div id="dashed">Dashed Border</div>
    <div id="dotted">Dotted Border</div>
    <div id="groove">groove Border</div>
    <div id="inset">Inset Border</div>
    <div id="outset">Outset Border</div>
    <div id="ridge">Ridge Border</div>
    <div id="round">Round Border</div>
    <div id="veryround">Rounder Border</div>
    <div id="shadow">Outset shadow</div>
    <div id="ishadow">Inset shadow</div>
    <div id="mixed">Mixed Border</div>
    <div id="button">Button</div>
  </body>
</html>
```

```
div { width:200px; margin:5px; padding:2px; text-align:center}
#simple { border:1px solid black; }
#dashed { border:1px dashed black; }
#dotted { border:1px dotted gray; }
#groove { border:5px groove blue; }
#inset { border:5px inset red; }
#outset { border:5px outset blue; }
#ridge { border:5px ridge black; }
#round { border: 1px solid; border-radius:5px;}
#veryround { border: 1px solid; border-radius:50%;}
#shadow {
  margin:10px;
  border:1px solid black;
  box-shadow: 5px 5px 3px 2px blue;
}
#ishadow {
  margin:15px;
  border:1px solid black;
  box-shadow: 5px 5px 3px 2px blue inset;
}
#mixed {
  border-top: 1px solid;
  border-top-left-radius:5px;
  border-left: 1px dotted;
  border-bottom: 5px ridge;
  border-bottom-right-radius: 10px;
}
#button {
  width:150px;
  background-color: #2233FF;
  color: white;
  border:5px outset blue;
  border-radius:50%;
}
```

```
#button, #button2 {
  margin: 5px; padding:3px;
  text-align: center;
  background-color: #2233FF;
  color: white;
  border:5px outset blue;
  border-radius:50%;
}
#button2 { opacity:.4; }
```

```
<span id="button">Active</span>
<span id="button2">Inactive</span>
```

```
<img id="image1" src="images/lesson0302.jpg" width="100" />
<img id="image2" src="images/lesson0303.jpg" width="100" />
<p id="uncleared">Uncleared Line of Text</p>
<p id="cleared">Cleared Line of Text</p>
```

```
#songInfo{
    background-color: black;
    top:70px;
    left:100px;
    height:150px;width:300px;
    z-index:1;
}
#buttons{
    background-color: white;
    top:200px;
    left:120px;
    z-index:2;
}
```

```html
<html>
  <head>
    <title>Layout</title>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="css/web_layout.css">
  </head>
  <body>
    <div id="buttons">
      <img src="../images/skipBack.png" />
      <img src="../images/seekBackward.png" />
      <img src="../images/play.png" />
      <img src="../images/stop.png" />
      <img src="../images/seekForward.png" />
      <img src="../images/skipForward.png" />
    </div>
    <div id="songinfo">
    <p id="title">9TH Symphony</p>
    <p id="artist">Beethoven</p>
    <p id="Year">1824</p>
  </div>
    <h1>Song Controls</h1>
  </body>
</html>
```

```
img{
  width:48px;
  margin:-5px;
  float:left;
}
#buttons, #songInfo{
  position:fixed;
  padding:5px;
  display:inline-block;
  border:8px ridge blue;
  border:radius 5px;
}
#songInfo{
  background-color: black;
  top:70px;
  left:100px;
  height:150px;width:300px;
  z-index:1;
}
#buttons{
  background-color: white;
  top:200px;
  left:120px;
  z-index:2;
}
#artist{
  color:cyan;
  font-size:25px;
  position:relative;
  left:30px;
  top:-50px;
}
#title{
  color:lime;
  font-size:15px;
  position:relative;
  left:30px;
  top:25px;
}
#year{
  color:yellow;
  font-size:15px;
  position:relative;
  left:30px;
  top:-40px;
}
```

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

```
<script src="http://code.jquery.com/jquery-latest.min.js"></script>
<script>
  function writeIt(){
    document.write("jQuery Version " + $().jquery + " loaded.");
  }
</script>
```

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
```

```
<script>
  function writeIt(){
    document.write("jQuery Version " + $().jquery + " loaded.");
  }
</script>
```

```
http://localhost/lesson06/jquery_version.html
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery Version</title>
    <meta charset="utf-8" />
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script>
      function writeIt(){
        document.write("jQuery Version " + $().jquery + " loaded.");
      }
    </script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

```
var q = document.getElementById("question");
...
<p id="question">Which method to you prefer?</p>
```

```
var paragraphs = document.getElementsByTagName("p");
```

```
$("p").css('font-weight', 'bold');
```

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
```

```
<script>
  function writeIt(){
    $("#heading").css('font-weight', 'bold').html("jQuery");
    var q = document.getElementById("question");
    q.innerHTML = "I Prefer jQuery!";
  }
</script>
```

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

```
    <p id="heading">jQuery or JavaScript</p>
    <p id="question">Which method to you prefer?</p>
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Changes</title>
    <meta charset="utf-8" />
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script>
      function writeIt(){
        $("#heading").css('font-weight', 'bold').html("jQuery");
        var q = document.getElementById("question");
        q.innerHTML = "I Prefer jQuery!";
      }
    </script>
  </head>
  <body onload="writeIt()">
    <p id="heading">jQuery or JavaScript</p>
    <p id="question">Which method do you prefer?</p>
  </body>
</html>
```

```
var myString = "Some Text";
var newString = myString + " Some More Text";
var myString = 'Some Text';
var anotherString = "Some Other Text";
var arr = ["one", "two", "three"]
var first = arr[0];
var obj = {"name":"Brad", "occupation":"Hacker", "age", "Unknown"};
var name = obj.name;
```

```
switch(expression){
  case value:
    <code to execute>
    break;
  case value2:
    <code to execute>
    break;
  default:
    <code to execute if not value or value2>
}
```

```
<script>
  function writeIt(){
    var lesson = new Date().getLessons();
    var timeOfDay;
    if(lesson>=7 && lesson<12){
      document.write("Good Morning!");
      timeOfDay="morning";
    } else if(lesson>=12 && lesson<18) {
      document.write("Good Day!");
      timeOfDay="day";
    } else {
      document.write("Good Night!");
      timeOfDay="night";
    }
  }
</script>
```

```
      switch(timeOfDay){
        case "morning":
        case "day":
          document.write("<img src='images/day.png' />");
          break;
        case "night":
          document.write("<img src='images/night.png' />");
          break;
        default:
          document.write("<img src='images/day.png' />");
      }
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>If Logic</title>
    <meta charset="utf-8" />
    <script>
      function writeIt(){
        var hour = new Date().getHours();
        var timeOfDay;
        if(hour>=7 && hour<12){
          document.write("Good Morning!");
          timeOfDay="morning";
        } else if(hour>=12 && hour<18) {
          document.write("Good Day!");
          timeOfDay="day";
        } else {
          document.write("Good Night!");
          timeOfDay="night";
        }
        switch(timeOfDay){
          case "morning":
          case "day":
            document.write("<img src='../images/day.png' />");
            break;
          case "night":
            document.write("<img src='../images/night.png' />");
            break;
          default:
            document.write("<img src='../images/day.png' />");
        }
      }
    </script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

```
var i = 1;
while (i<5){
  document.write("Iteration " + i + "<br>");
  i++;
}
```

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var i=0;
do{
  var day=days[i++];
  document.write("It's " + day + "<br>");
} while (day != "Wednesday");
```

```
for (statement 1; statement 2; statement 3;){
  code to be executed;
}
```

```
for (var x=1; x<=3; x++){
  for (var y=1; y<=3; y++){
    document.write(x + " X " + y + " = " + (x*y) + "<br>");
  }
}
```

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  document.write("It's " + days[idx] + "<br>");
}
```

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  if (days[idx] == "Wednesday")
    break;
  document.write("It's " + days[idx] + "<br>");
}
```

```
var days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var idx in days){
  if (days[idx] == "Wednesday")
    continue;
  document.write("It's " + days[idx] + "<br>");
}
```

```
function myFunction(){
  document.write("Hello World");
}
```

```
function greeting(name, city){
  document.write("Hello " + name);
  document.write(". How is the weather in " + city);
}
```

```
function formatGreeting(name, city){
  var retStr = "";
  retStr += "Hello <b>" + name + "</b><br>";
  retStr += "Welcome to " + city + "!";
 return retStr;
}
var greeting = formatGreeting("Brad", "Rome");
document.write(greeting);
```

```
function myFunc(value){
  if (value == 0)
    return;
  code_to_execute_if_value_nonzero;
}
```

```
var superData = {"Super Man":["Lex Luther"],
                 "Bat Man":["Joker", "Riddler",],
                 "Spider Man":["Green Goblin",
                               "Vulture", "Carnage"],
                 "Thor":["Loki", "Frost Giants"]};
```

```
function writeIt(){
  for (hero in superData){
    var villains = superData[hero];
    for (villainIdx in villains){
      var villain = villains[villainIdx];
      var listItem = makeListItem(hero, villain);
      document.write(listItem);
    }
  }
}
```

```
function makeListItem(name, value){
  var itemStr = "<li>" + name + ":&nbsp;" + value + "</li>";
  return itemStr;
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Functions</title>
    <meta charset="utf-8" />
    <script>
      var superData = {"Super Man":["Lex Luther"],
                       "Bat Man":["Joker", "Riddler"],
                       "Spider Man":["Green Goblin",
                                     "Vulture", "Carnage"],
                       "Thor":["Loki", "Frost Giants"]};
      function writeIt(){
        for (hero in superData){
          var villains = superData[hero];
          for (villainIdx in villains){
            var villain = villains[villainIdx];
            var listItem = makeListItem(hero, villain);
            document.write(listItem);
          }
        }
      }
      function makeListItem(name, value){
        var itemStr = "<li>" + name + ":&nbsp;" + value + "</li>";
        return itemStr;
      }
    </script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

```
<script>
  var myVar = 1;
  function writeIt(){
    var myVar = 2;
    document.write(myVar);
    writeMore();
  }
  function writeMore(){
    document.write(myVar);
  }
</script>
```

```
try {
    your_code_here
} catch (err) {
  document.write(err.message + ": happened when loading the script");
}
```

```
<script>
  function sqrRoot(x) {
    try {
      if(x=="")    throw "Can't Square Root Nothing";
      if(isNaN(x)) throw "Can't Square Root Strings";
      if(x<0)      throw "Sorry No Imagination";
      return "sqrt("+x+") = " + Math.sqrt(x);
    } catch(err){
      return err;
    }
  }
  function writeIt(){
    document.write(sqrRoot("four") + "<br>");
    document.write(sqrRoot("") + "<br>");
    document.write(sqrRoot("4") + "<br>");
    document.write(sqrRoot("-4") + "<br>");
  }
</script>
```

```
function testTryCatch(value){
  try {
    if (value < 0){
      throw "too small";
    } else if (value > 10){
      throw "too big";
    }
    your_code_here
  } catch (err) {
    document.write("The number was " + err.message");
  } finally {
    document.write("This is always written.");
  }
}
```

```
<tr><td>planent</td><td>moon</td></tr>
```

```
<script>
  function writeMe(){
    document.write(document.me);
  }
  document.me = "Brad Dayley";
  document.writeMe = writeMe;
  document.writeMe();
</script>
```

```
var x = new Number("5.55555555");
var word1 = "Today ";
var word2 = "is ";
var word3 = "tomorrows\' ";
var word4 = "yesterday.";
var sentence1 = word1 + word2 + word3 + word4;
var sentence2 = word1.concat(word2, word3, word4);
var myStr = "I think, therefore I am.";
if (myStr.indexOf("think") != -1){
  document.write(myStr);
}
```

```
var username = "Brad";
var output = "<username> please enter your password: ";
output = output.replace("<username>", username);
```

```
<script>
  function writeIt(){
...
  }
</script>
  </head>
  <body onload="writeIt()">
```

```
var line1 = "In a [place] a long time ago, ";
var line2 = "there lived an [animal] that liked to ";
var line3 = "[action] people.";
```

```
        var place = "crater";
        var animal = "elephant";
        var action = "smell";
```

```
var madlib = line1.concat(line2, line3);
```

```
    madlib = madlib.replace("[place]", place);
    madlib = madlib.replace("[animal]", animal);
    madlib = madlib.replace("[action]", action);
```

```
      document.write(madlib);
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>String Manipulation</title>
    <meta charset="utf-8" />
<script>
  function writeIt(){
    var line1 = "In a [place] a long time ago, ";
    var line2 = "there lived an [animal] that liked to ";
    var line3 = "[action] people.";
    var place = "crater";
    var animal = "elephant";
    var action = "smell";
    var madlib = line1.concat(line2, line3);
    madlib = madlib.replace("[place]", place);
    madlib = madlib.replace("[animal]", animal);
    madlib = madlib.replace("[action]", action);
    document.write(madlib);
  }
</script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

```
var arr = ["one", "two", "three"];
var arr2 = new Array();
arr2[0] = "one";
arr2[1] = "two";
arr3[2] = "three";
var arr3 = new Array();
arr3.push("one");
arr3.push("two");
arr3.push("three");
```

```
var week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
var first = week[0];
var last = week[week.length-1];
var arr1 = [1,2,3];
var arr2 = ["three", "four", "five"]
var arr3 = arr1 + arr2;
var arr4 = arr1.concat(arr2);
var week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
for (var i=0; i<week.length; i++){
  document.write("<li>" + week[i] + "</li>");
}
for (dayIndex in week){
  document.write("<li>" + week[dayIndex] + "</li>");
}
```

```
var timeArr = [12,10,36];
var timeStr = timeArr.join(":");
```

```
function message(day){
  var week = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
  if (week.indexOf(day) != -1){
    document.write("Happy " + day);
  }
}
```

```
<script>
  function writeIt(){
...
  }
</script>
  </head>
  <body onload="writeIt()">
```

```
function writeArray(msg, arr){
  var arrString = arr.join(" | ");
  document.write("<b>"+ msg + ": </b>" + arrString + "<br><br>");
}
```

```
    var weekDays = ["Monday", "Tuesday", "Wednesday", "Thursday",
➥"Friday"];
    writeArray("Week Days", weekDays);
    var weekEnd = new Array();
    weekEnd.push("Saturday");
    weekEnd.push("Sunday");
    writeArray("Weekend", weekEnd);
```

```
var week = weekDays.concat([]);
week.unshift(weekEnd[1]);
week.push(weekEnd[0]);
writeArray("Week", week);
```

```
var midWeek = week.slice(2,5);
writeArray("Mid Week", midWeek);
```

```
    var sortedWeek = week.sort();
    document.write("<b>Sorted Days :</b> <br>");
    for (dayIndex in sortedWeek){
      document.write(sortedWeek[dayIndex] + "<br>");
    }
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Array Manipulation</title>
    <meta charset="utf-8" />
<script>
  function writeArray(msg, arr){
    var arrString = arr.join(" | ");
    document.write("<b>"+ msg + ": </b>" + arrString + "<br><br>");
  }
  function writeIt(){
    var weekDays = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"];
    writeArray("Week Days", weekDays);
    var weekEnd = new Array();
    weekEnd.push("Saturday");
    weekEnd.push("Sunday");
    writeArray("Weekend", weekEnd);
    var week = weekDays.concat([]);
    week.unshift(weekEnd[1]);
    week.push(weekEnd[0]);
    writeArray("Week", week);
    var midWeek = week.slice(2,5);
    writeArray("Mid Week", midWeek);
    var sortedWeek = week.sort();
    document.write("<b>Sorted Days :</b> <br>");
    for (dayIndex in sortedWeek){
      document.write(sortedWeek[dayIndex] + "<br>");
    }
  }
</script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

Mon Dec 10 2012 17:30:27 GMT-0700 (Mountain Standard Time)

```
Date(year, month, day, lessons, minutes, seconds, milliseconds)
```

```
var d1 = Date("2012, 12, 12, 12, 12, 12, 00");
var d2 = Date("December 12, 2012 12:12:12");
var currentTime = new Date();
var serverTime = new Date("December 12, 2012 12:12:12");
if (currentTime>serverTime){
    alert("Mayans Wrong?");
}
```

```
var re = new RegExp(pattern,modifiers);
var myStr = "Teach Yourself jQuery & JavaScript in 24 Lessons";
var re = /yourself/i;
var newStr = myStr.replace(re, "Your Friends");
```

```
var user = {'first':'Brad','last':'Dayley'};
```

```
function User(first, last){
  this.first = first;
  this.last = last;
}
var user = new User("Brad", "Dayley");
```

```
document.write(user.first + " " + user.last);
```

```
var user = new Object();
user.first="Brad";
user.last="Dayley";
user.getFullName = makeFullName;
var user2 = {'first':'Brad', 'last':'Dayley',
             'getFullName':makeFullName};
function makeFullName(){
  return this.first + " " + this.last;
}
document.write(user.getFullName());
```

```
function User(first, last){
  this.first = first;
  this.last = last;
  this.getFullName = function(){
      return this.first + " " + this.last;
    };
}
```

```
function UserP(first, last){
  this.first = first;
  this.last = last;
}
UserP.prototype = {
  getFullName: function(){
      return this.first + " " + this.last;
    }
};
```

```
var me = UserP("Brad", "Dayley");
var myFullName = me.getFullName();
```

```
<script>
  function writeIt(){
...
  }
</script>
  </head>
  <body onload="writeIt()">
```

```
function Character(first, last, land, race){
  this.first = first;
  this.last = last;
  this.race = race;
  this.land = land;
}
```

```
Character.prototype = {
  getFullName: function(){
      return this.first + " " + this.last;
  },
  getDetails: function(){
      return "is a " + this.race + " from the " + this.land;
    }
};
```

```
    var characters = new Array();
    characters.push(new Character("John 117", "Master Chief",
                                  "Earth", "Spartan"));
    characters.push(new Character("vadam", "The Arbitor",
                                  "High Charity", "Elite"));
    characters.push(new Character("Gravemind", "", "Precursors",
                                  "Flood"));
    sgtJohnson = new Character("Sgt Johnson", "", "", "");
    sgtJohnson.land = "Earth";
    sgtJohnson.race = "Human";
    characters.push(sgtJohnson);
```

```
for (var i=0; i<characters.length; i++){
  var character = characters[i];
  character.number = i+1;
  document.write(character.number + ". " +
                 character.getFullName() + " " +
                 character.getDetails() + "<br>");
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Custom Objects</title>
    <meta charset="utf-8" />
<script>
  function writeIt(){
    var characters = new Array();
    characters.push(new Character("John 117", "Master Chief",
                                  "Earth", "Spartan"));
    characters.push(new Character("vadam", "The Arbitor",
                                  "High Charity", "Elite"));
    characters.push(new Character("Gravemind", "", "Precursors",
                                  "Flood"));
    sgtJohnson = new Character("Sgt Johnson", "", "", "");
    sgtJohnson.land = "Earth";
    sgtJohnson.race = "Human";
    characters.push(sgtJohnson);
    for (var i=0; i<characters.length; i++){
      var character = characters[i];
      character.number = i+1;
      document.write(character.number + ". " +
                     character.getFullName() + " " +
                     character.getDetails() + "<br>");
    }
  }
  function Character(first, last, land, race){
    this.first = first;
    this.last = last;
    this.race = race;
    this.land = land;
  }
  Character.prototype = {
    getFullName: function(){
        return this.first + " " + this.last;
    },
    getDetails: function(){
        return "is a " + this.race + " from the " + this.land;
      }
  };
</script>
  </head>
  <body onload="writeIt()">
  </body>
</html>
```

```
var domObj = jqueryObj.get();
var containerObj = document.getElementById("container");
var objs = document.getElementsByClassName("myClass");
for (var i=0; i<objs.length; i++){
  var htmlElement = objs[i];
  ...
}
```

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

```
<script type="text/javascript" src="js/dom_objects.js"></script>
<link rel="stylesheet" type="text/css" href="css/dom_objects.css">
```

```
<input id="textIn" type="text"/>
<input type="button" onclick="textChange()" value="Update" /><br>
```

```
<span class="heading"></span>
<p id="p1"></p>
<span class="heading"></span>
<p id="p2"></p>
```

```
var inElement = document.getElementById("textIn");
var outElements = document.getElementsByTagName("p");
var headingElements = document.getElementsByClassName("heading");
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Objects</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="js/dom_objects.js"></script>
    <link rel="stylesheet" type="text/css" href="css/dom_objects.css">
  </head>
  <body>
    <input id="textIn" type="text"/>
    <input type="button" onclick="textChange()" value="Update" /><br>
    <span class="heading"></span>
    <p id="p1"></p>
    <span class="heading"></span>
    <p id="p2"></p>
  </body>
</html>
```

```
function textChange(){
  var inElement = document.getElementById("textIn");
  var outElements = document.getElementsByTagName("p");
  var headingElements = document.getElementsByClassName("heading");
  for(var i=0; i<outElements.length; i++){
    var outItem = outElements[i];
    headingElements[i].innerHTML = "Updating " + (i+1) +
                                   " to " + inElement.value;
    outItem.innerHTML = inElement.value;
  }
}
```

```
<script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
<script type="text/javascript" src="js/jquery_selectors.js"></script>
<link rel="stylesheet" type="text/css" href="css/jquery_selectors.css">
```

```
    <span onclick="setEven()">Even</span>
    <span onclick="setOdd()">Odd</span>
    <span onclick="setFirst4()">First 4</span>
```

```
$("li, span").css("font-weight","");
```

```
var $evenItems = $("li:even");
$evenItems.css("font-weight","bold");
```

```
$("span:contains(Even)").css("font-weight","bold");
```

```
    $(".label").html("Even");
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery Selectors</title>
    <meta charset="utf-8" />
    <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/jquery_selectors.js"></script>
    <link rel="stylesheet" type="text/css" href="css/jquery_selectors.css">
  </head>
  <body>
    <span onclick="setEven()">Even</span>
    <span onclick="setOdd()">Odd</span>
    <span onclick="setFirst4()">First 4</span>
    <p class="label">Planets</p>
    <ul>
      <li>Poseidon</li>
      <li>Ares</li>
      <li>Apollo</li>
      <li>Hermes</li>
      <li>Nike</li>
      <li>Nemesis</li>
      <li>Zeus</li>
      <li>Hades</li>
    </ul>
  </body>
</html>
```

```
function setEven(){
  $("li, span").css("font-weight","");
  var $evenItems = $("li:even");
  $evenItems.css("font-weight","bold");
  $("span:contains(Even)").css("font-weight","bold");
  $(".label").html("Even");
}
function setOdd(){
  $("li, span").css("font-weight","");
  var $oddItems = $("li:odd");
  $oddItems.css("font-weight","bold");
  $("span:contains(Odd)").css("font-weight","bold");
  $(".label").html("Odd");
}
function setFirst4(){
  $("li, span").css("font-weight","");
  var $first4 = $("li:lt(4)");
  $first4.css("font-weight","bold");
  $("span:contains('First 4')").css("font-weight","bold");
  $(".label").html("First 4");
}
```

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

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

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

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

```
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/dom_manipulation.js"></script>
    <link rel="stylesheet" type="text/css" href="css/dom_manipulation.css">
```

```
$(document).ready(function (){
. . .
});
```

```
$("input:eq(0)").click(function (){
  $("p").each(function(){
    var parts = $(this).html().split(" ");
    $(this).css({"font-size":parts[1]+"px", color:parts[0]});
  });
});
```

```
$("input:eq(1)").click(function (){
  var items = $("p").map(function(){
      var parts = $(this).html().split(" ");
      return {color:parts[0], size:parts[1]};
    }).get();
    for (var idx in items){
      var item = items[idx];
      var span = $("<span>" + item.color + "</span>");
      var size = item.size*5;
      span.css({"background-color":item.color,  "font-size": item.
➥size+"px", width:size, height:size});
      $("div").append(span);
    }
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/dom_manipulation.js"></script>
    <link rel="stylesheet" type="text/css" href="css/dom_manipulation.css">
  </head>
  <body>
    <input type="button" value=".each()">
    <input type="button" value=".map()">
    <p>red 10</p>
    <p>orange 15</p>
    <p>yellow 20</p>
    <p>green 25</p>
    <p>blue 30</p>
    <p>indigo 35</p>
    <p>violet 40</p>
    <div></div>
  </body>
</html>
```

```
$(document).ready(function (){
  $("input:eq(0)").click(function (){
    $("p").each(function(){
      var parts = $(this).html().split(" ");
      $(this).css({"font-size":parts[1]+"px", color:parts[0]});
    });
  });
  $("input:eq(1)").click(function (){
    var items = $("p").map(function(){
        var parts = $(this).html().split(" ");
        return {color:parts[0], size:parts[1]};
      }).get();
    for (var idx in items){
      var item = items[idx];
      var span = $("<span>" + item.color + "</span>");
      var size = item.size*5;
      span.css({"background-color":item.color,  "font-size": item.size+"px",
                width:size, height:size});
      $("div").append(span);
    }
  });
});
```

```css
p{margin:0px; padding:0px;}
span{
  display:inline-block;
  color: white;
  text-align:center;
}
```

```
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/traverse_dom.js"></script>
    <link rel="stylesheet" type="text/css" href="css/traverse_dom.css">
```

```
$("span").css("background-color","lightgrey");
```

```
3    $("div").each(function(i){
...
8    })
```

```
    var $input = $(this).children("input:first");
    var $value = $input.val();
    var filter = "span:lt(" + $value + ")";
    $input.siblings(filter).css("background-color","blue");
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Traversing the DOM</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/traverse_dom.js"></script>
    <link rel="stylesheet" type="text/css" href="css/traverse_dom.css">
  </head>
  <body>
    <p>How satisfied are you 1-5</p>
    <div>
      <label>Quality</label>
      <input type="text" onkeyup="update()"></input>
      <span></span><span></span><span></span><span></span><span></span>
    </div>
    <div><label>Taste</label>
      <input type="text" onkeyup="update()"></input>
      <span></span><span></span><span></span><span></span><span></span>
    </div>
    <div>
      <label>Server</label>
      <input type="text" onkeyup="update()"></input>
      <span></span><span></span><span></span><span></span><span></span>
    </div>
  </body>
</html>
```

```
function update(){
  $("span").css("background-color","lightgrey");
  $("div").each(function(i){
    var $input = $(this).children("input:first");
    var $value = $input.val();
    var filter = "span:lt(" + $value + ")";
    $input.siblings(filter).css("background-color","blue");
  })
}
```

```css
span{
  display:inline-block;
  height:15px;
  width:10px;
  background-color:lightgrey;
  margin:1px;
  border-radius:50%;
}
input {
  width:20px;
}
label {
  display:inline-block;
  width:60px;
}
```

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

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

```
window.onload = onloadHandler;
```

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

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

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

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

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

```
<h1 id="heading"></h1>
<div onclick="clickHandler(event,heading,1,"Yes")">Click Here</div>
<div onclick="clickHandler(event,heading,2,"No")">Or Here</div>
```

```
function clickHandler(e,obj,num,msg){
  obj.innerHTML = "DIV " + num + " says " + msg +" at X postion: " + e.screenX;
}
```

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

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

```
function clickHandler(e,objId,num,msg){
  var obj = document.getElementById(objId);
  obj.innerHTML = "DIV " + num + " says " + msg +" at X postion: " + e.screenX;
}
```

```
function yesWrapper(e){
  clickHandler(e, "heading", 1, "yes");
  e.target.removeEventListener('click', yesWrapper);
}
function noWrapper(e){
  clickHandler(e, "heading", 1, "no");
  e.target.removeEventListener('click', noWrapper);
}
```

```
function onloadHandler(){
  document.getElementById("div1").addEventListener('click', yesWrapper, false);
  document.getElementById("div2").addEventListener('click', noWrapper, false);
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Broken Event</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/broken_event.js"></script>
    <link rel="stylesheet" type="text/css" href="css/broken_event.css">
  </head>
  <body onload="onloadHandler()">
    <div id="div1")">Say Yes</div>
    <div id="div2")">Say No</div>
    <h1 id="heading"></h1>
  </body>
</html>
```

```
function clickHandler(e,objId,num,msg){
  var obj = document.getElementById(objId);
  obj.innerHTML = "DIV " + num + " says " + msg +" at X postion: " + e.screenX;
}
function yesWrapper(e){
  clickHandler(e, "heading", 1, "yes");
  e.target.removeEventListener('click', yesWrapper);
}
function noWrapper(e){
  clickHandler(e, "heading", 2, "no");
  e.target.removeEventListener('click', noWrapper);
}
function onloadHandler(){
  document.getElementById("div1").addEventListener('click', yesWrapper, false);
  document.getElementById("div2").addEventListener('click', noWrapper, false);
}
```

```css
div{
  border-radius:5px;
  margin:3px;
  padding:5px;
  background-color:lightgrey;
  font-weight:bold;
  display:inline-block;
  cursor:pointer;
}
```

```
on(events [, selector] [, data], handler(eventObject))
on(events-map [, selector][, data])
```

```
off(events [, selector] [, handler(eventObject]))
off(events-map [, selector])
```

```
$("div").on("click",clickHandler);
$("div").off("click",clickHandler);
```

```
function clickHandler(e){
  $("#"+e.data.objId).html(e.target.id + " says " + e.data.answer +
                             " at X postion: " + e.screenX);
}
```

```
$(document).ready(function(){
...
});
```

```
$("#div1").on({"click":clickHandler},
              {"objId":"heading", "answer":"yes"});
```

```
$(document).on("click",
              "#div2",
              {"objId":"heading", "answer":"no"},
              clickHandler);
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Working Events</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/working_event.js"></script>
    <link rel="stylesheet" type="text/css" href="css/working_event.css">
  </head>
  <body>
    <div id="div1")">Say Yes</div>
    <div id="div2")">Say No</div>
    <h1 id="heading"></h1>
  </body>
</html>
```

```
function clickHandler(e){
  $("#"+e.data.objId).html(e.target.id + " says " + e.data.answer +
                             " at X postion: " + e.screenX);
}
$(document).ready(function(){
  $("#div1").on({"click":clickHandler},
                 {"objId":"heading", "answer":"yes"});
  $(document).on("click",
                 "#div2",
                 {"objId":"heading", "answer":"no"},
                 clickHandler);
});
```

```css
div{
  border-radius:5px;
  margin:3px;
  padding:5px;
  background-color:lightgrey;
  font-weight:bold;
  display:inline-block;
  cursor:pointer;
}
```

```
.<event type>( [eventData], handler(eventObject))
```

```
obj.click( dataObj, function myHandler(e){...});
obj.addEventListener("click", dataObj, function myHandler(e){...});
obj.hover(function enterHandler(e){...}, function leaveHandler(e){...});
obj.addEventListener("mouseenter", function enterHandler(e){...});
obj.addEventListener("mouseleave", function leaveHandler (e){...});
obj.hover(function hoverHandler(e){...});
```

```
var clickEvent = document.createEvent("MouseEvents");
```

```
clickEvent.initMouseEvent("click", true, true, window, 0, 0, 0, 0, 0, 0,
                           false, false, false, false, 0, null);
```

```
var obj = document.getElementById("someId");
obj.dispatchEvent(clickEvent);
```

```
function onloadHandler(){
  var employee = document.getElementById("Employee");
  employee.addEventListener('click', simpleClick, false);
  var registered = document.getElementById("Registered");
  registered.addEventListener('click', eventClick, false);
}
```

```
function simpleClick(e){
  var cb = document.getElementById("check"+e.target.id);
  cb.click();
}
```

```
function eventClick(e){
    var event = document.createEvent("MouseEvents");
    event.initMouseEvent("click", true, true, window,
                          0, 0, 0, 0, 0, false, false,
                          false, false, 0, null);
    var cb = document.getElementById("check"+e.target.id);
    cb.dispatchEvent(event);
}
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Manual Event</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="js/manual_event.js"></script>
    <link rel="stylesheet" type="text/css" href="css/manual_event.css">
  </head>
  <body onload="onloadHandler()">
    <input id="checkAvailable" type="checkbox" /><label>Available</label><br>
    <input id="checkAllWeek" type="checkbox" /><label>All Week</label><br>
    <input id="checkWeekDays" type="checkbox" /><label>Week Days</label><br>
    <span id="Available">I'm Available</span>
    <span id="AllWeek">All Week</span>
    <span id="WeekDays">Week Days</span>
  </body>
</html>
```

```
function onloadHandler(){
  var available = document.getElementById("Available");
  Available.addEventListener('click', simbleClick, false);
  var allWeek = document.getElementById("AllWeek");
  AllWeek.addEventListener('click', eventClick, false);
  var weekDays = document.getElementById("WeekDays");
  WeekDays.addEventListener('click', eventClick, false);
}
function simbleClick(e){
  var cb = document.getElementById("check"+e.target.id);
  cb.click();
}
function eventClick(e){
    var event = document.createEvent("MouseEvents");
    event.initMouseEvent("click", true, true, window,
                          0, 0, 0, 0, 0, false, false,
                          false, false, 0, null);
    var cb = document.getElementById("check"+e.target.id);
    cb.dispatchEvent(event);
}
```

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

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

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

```
$(document).ready(function(){
  $("input").keypress(function (e){inputHandler(e)});
  $("span").click(function (e){spanHandler(e)});
});
```

```
function inputHandler(e){
  var chr = String.fromCharCode(e.charCode);
  $("p").append(chr);
}
```

```
function spanHandler(e){
  var chrCode = e.target.innerHTML.charCodeAt(0);
  $("input").trigger({'type':'keypress', 'charCode':chrCode});
}
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Event Based Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/card_suits.js"></script>
    <link rel="stylesheet" type="text/css" href="css/card_suits.css">
  </head>
  <body>
    <p>Card and Suit: </p>
    <input type="text" />
    <span>&#9824;</span>
    <span>&#9827;</span>
    <span>&#9829;</span>
    <span>&#9830;</span>
  </body>
</html>
```

```
function inputHandler(e){
  var chr = String.fromCharCode(e.charCode);
  $("p").append(chr);
}
function spanHandler(e){
  var chrCode = e.target.innerHTML.charCodeAt(0);
  $("input").trigger({'type':'keypress', 'charCode':chrCode});
}
$(document).ready(function(){
  $("input").keypress(function (e){inputHandler(e)});
  $("span").click(function (e){spanHandler(e)});
});
```

```css
span{
  border-radius:5px;
  margin:3px;padding:5px;
  background-color:#C0C0C0;
  border:3px ridge;
  display:inline-block;
  cursor:pointer;
}
p{
  border:4px outset steelblue;
  padding:3px;
  color:white;background-color:skyblue;
  font-size:30px;font-weight:bold;
}
```

```
var myEvent = document.createEvent("CustomEvent");
myEvent .initCustomEvent(
  "worldEnds",
  true,
  false,
  {
      'fire': false,
      'ice': false,
      'time': new Date()
  }
);
```

```
var obj = document.getElementById("#notify");
obj.dispatchEvent(myEvent);
```

```
document.addEventListener("worldEnds ", endOfWorldHandler, false);
```

```
var custEvent = $.Event("worldEnds", {
  fire: false,
  ice: false,
  time: new Date()
});
```

```
$(document).on("worldEnds", endOfWorldHandler);
```

```
function functionA(){return true;}
function functionB(){return true;}
function functionC(){return true;}
var callbacks = $Callbacks("unique stopOnFalse");
callbacks.add(functionA);
callbacks.add(functionB);
callbacks.fire()
callbacks.remove(functionB);
callbacks.add(FunctionC);
callbacks.fire();
callbacks.disable();
callbacks.fire();
```

```
var resultString = "";
function function1(n1, n2, p3) { resultString += "Problem: "; }
function function2(n1, n2) { resultString += n1 + " + "; }
function function3(n1, n2) { resultString += n2 + " = "; }
var deferredObj = $.Deferred();
deferredObj.done( [function1, function2], function3).done(function(n1, n2) {
    resultString += n1 + n2;
    $("div").append(resultString);
  });
deferredObj.resolve(5,6);
```

```
var screenPosition = e.screenX + ", " + e.screenY;
var pagePosition = e.pageX + ", " + e.pageY;
var browserPosition = e.clientX + ", " + e.clientY;
```

```
domObject.value = 5;
var value = domObject.value;
```

```
$("#textInput").val(5);
var value = $("#textInput ").val();
var state = $("#bannerImg").attr("src");
```

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

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

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

```
var domObj = document.getElementById("banner");
var color = domObj.style.color;
var color = domObj.style.["background-color"];
```

```
domObj.style.position = "absolute";
domObj.style.top = "100px";
domObj.style.left = "100px";
```

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

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

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

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

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

```
var class = $("#mainList").get().className;
var sWidth = screen.width;
var sHeight = screen.height;
var colorBits = screen.colorDepth;
var pageWidth = window.innerWidth;
var pageHeight = window.innerHeight;
```

```
$(window).load(function(){
  $(document).mousemove(mousePosition);
  $("*").mouseover(elementInfo);
  $("*").change(elementInfo);
  $("*").keypress(elementInfo);
  $(window).resize(windowResize);
});
```

```
function mousePosition(e){
  e.stopPropagation();
  $("#screenSize").html(screen.width + "x" + screen.height);
  $("#colors").html(screen.colorDepth+"bit");
  $("#browserSize").html(window.innerWidth + "x" + window.innerHeight);
  $("#mousePosition").html("X:" + e.screenX + "  Y:" + e.screenY);
  $("#pagePosition").html("X:" + e.pageX + "  Y:" + e.pageY);
  $("#scrollPosition").html("X:" + e.clientX + "  Y:" + e.clientY);
}
```

```
function elementInfo(e){
  e.stopPropagation();
  $(".infoContainer span").html("");
  var domObj = e.target;
  var jObj = $(domObj);
...
}
```

```
$("#elementId").html(domObj.id);
$("#elementType").html(jObj.prop('tagName'));
$("#elementClass").html(domObj.className);
```

```
$("#elementSize").html(jObj.width() + "x" + jObj.height());
$("#elementPosition").html(jObj.offset().top + ", " +  jObj.offset().
➥left);
```

```
$("#elementColor").html(jObj.css("color"));
$("#elementValue").html(jObj.val());
try{
  $("#elementChecked").html(jObj.prop('checked').toString());
} catch (e) {}
$("#elementSource").html(jObj.attr('src'));
```

```
function windowResize(e){
  $("#browserSize").html(window.innerWidth + "x" + window.innerHeight);
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Web Page Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/web_page_manipulation.js"></script>
   <link rel="stylesheet" type="text/css" href="css/web_page_manipulation.css">
  </head>
  <body>
    <div>
      <div id="banner" class="header">
        Teach YourSelf AngularJS, JavaScript and jQuery</div>
      <div id="menu" class="menu">
        <span class="menuItem">Lessons</span>
        <span class="menuItem">Docs</span>
        <span class="menuItem">Code</span>
      </div>
      <div id="content">
        <p class="imageArea">
          <img id="Cliff" src="/images/cliff.jpg" width="200px"/>
          Cliff</p>
        <select>
```

```
          <option value=1>lesson 1</option>
          <option value=2>lesson 2</option>
          <option value=3>lesson 3</option>
          <input type=text />
        </select>
        <input type="checkbox" />
      </div>
    </div>
    <div class=infoContainer>
      <p>Screen & Browser Info</p>
      <div>Screen Size: <span id="screenSize"></span></div>
      <div>Color Level: <span id="colors"></span></div>
      <div>Browser Size: <span id="browserSize"></span></div>
      <p>Mouse Position</p>
      <div>Absolute Mouse: <span id="mousePosition"></span></div>
      <div>Page Mouse: <span id="pagePosition"></span></div>
      <div>Relative Mouse: <span id="scrollPosition"></span></div>
      <p>Element Info</p>
      <div>Element ID: <span id="elementId"></span></div>
      <div>Element Type: <span id="elementType"></span></div>
      <div>Element Class: <span id="elementClass"></span></div>
      <div>Element Size: <span id="elementSize"></span></div>
      <div>Element Position: <span id="elementPosition"></span></div>
      <div>Element Value: <span id="elementValue"></span></div>
      <div>Element Checked: <span id="elementChecked"></span></div>
      <div>Image Source: <span id="elementSource"></span></div>
      <div>Element Color: <span id="elementColor"></span></div>
    </div>
  </body>
</html>
```

```
$(window).load(function(){
  $(document).mousemove(mousePosition);
  $("*").mouseover(elementInfo);
  $("*").change(elementInfo);
  $("*").keyup(elementInfo);
  $(window).resize(windowResize);
});
function mousePosition(e){
  e.stopPropagation();
  $("#screenSize").html(screen.width + "x" + screen.height);
  $("#colors").html(screen.colorDepth+"bit");
  $("#browserSize").html(window.innerWidth + "x" + window.innerHeight);
  $("#mousePosition").html("X:" + e.screenX + "  Y:" + e.screenY);
  $("#pagePosition").html("X:" + e.pageX + "  Y:" + e.pageY);
  $("#scrollPosition").html("X:" + e.clientX + "  Y:" + e.clientY);
}
function elementInfo(e){
  e.stopPropagation();
  $(".infoContainer span").html("");
  var domObj = e.target;
  var jObj = $(domObj);
  $("#elementId").html(domObj.id);
  $("#elementType").html(jObj.prop('tagName'));
  $("#elementClass").html(domObj.className);
  $("#elementSize").html(jObj.width() + "x" + jObj.height());
  $("#elementPosition").html(jObj.offset().top + ", " +  jObj.offset().left);
  $("#elementColor").html(jObj.css("color"));
  $("#elementValue").html(jObj.val());
  try{
    $("#elementChecked").html(jObj.prop('checked').toString());
  } catch (e) {}
  $("#elementSource").html(jObj.attr('src'));
}
function windowResize(e){
  $("#browserSize").html(window.innerWidth + "x" + window.innerHeight);
}
```

```
.infoContainer{
  border:3px ridge;
  padding:3px;
  position:fixed; display:inline-block;
  top:100px; right:50px;
  background-color:#C0C0C0;
  width:200px; height:280px;
  font:12px arial;
}
.infoContainer div{ margin-left:5px; }
.infoContainer div span{
  color:blue;
  float:right; }
.infoContainer p{
  margin:0px; padding:0px;
  font-size:14px; font-weight:bold;
 }
#banner{
  height:100;
  color:white; background-color:blue;
  font-size:40px; text-align:center;
}
#menu{ background-color:gray; padding:5px; }
.menuItem{
  padding:3px; margin-left:3px;
  background-image: -moz-linear-gradient(top, #0022ff 0%, #AACCFF 85%, #0022ff
➥100%);
  background-image: -webkit-linear-gradient(top, #0022ff 0%, #AACCFF 85%,
➥#0022ff 100%);
  background-image: -ms-linear-gradient(top, #0022ff 0%, #AACCFF 85%, #0022ff
➥100%);
  font:20px bold;
  color:white;
}
.imageArea{color:firebrick;}
```

```
domObj.innerHTML = "<p>Paragraph 1 goes here</p>";
domObj.innerHTML += "<p>Paragraph 2 goes here</p>";
```

```
var newP = createElement("p");
var newT = createTextNode("Paragraph 1 goes here");
newP.appendChild(newT);
domObj.appendChild(newP);
```

```
var newImg = createElement("img");
newImg.src = "images/sunset.jpg";
newImg.height = 200;
domObj.appendChild(newImg);
```

```
$("#myDiv").html("<p>Paragraph 1 goes here</p>");
```

```
newP.html("Paragraph 1 goes here");
```

```
var newImg = $("<img></img>");
newImg.attr("src", "images/sunset.jpg");
newImg.height(30);
$("li").append(newImg);
```

```
var parent=document.getElementById("container");
var child=document.getElementById("paragraphA");
child.parentNode.removeChild(child);
```

```
var ps = $("#div1").detach("p");
ps.appendTo("#div2");
```

```
$("div").html($("<p>New Paragraph</p>"));
```

```
$("#parentA span").replaceAll("#parentB div");
```

```
$("div").replaceWith($("<div></div>"));
```

```
$("p:eq(2)").after($("<p>New Fourth Paragraph</p>"));
```

```
$("p:eq(2)").before($("<p>New Third Paragraph</p>"));
```

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

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

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

```
$(window).load(function(){
  $("docMenu").hide();
  $("#lessons").click(setLessonNav);
  $("#docs").click(setDocNav);
  $("#fade").click(fade);
});
```

```
$("#content").append(select).append("<br><p></p>");
```

```
function setDocNav(){
  $("docMenu").show();
  $("span").removeClass("active");
  $("#docs").addClass("active");
}
```

```
function setDoc(doc){
  var frame = $("<iframe></iframe>");
  frame.attr("src", doc);
  $("#content").html(frame);
}
```

```
function fade(){
  var opacity = $("#content").css("opacity");
  if (opacity < 1){ $("#content").css("opacity", 1);}
  else { $("#content").css("opacity", .5); }
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Web Element Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/web_element_manipulation.js"></
➥script>
    <link rel="stylesheet" type="text/css" href="css/web_element_manipulation.
➥css">
  </head>
  <body>
    <div id="container">
      <div id="menu" class="menu">
        <span id="lessons" class="menuItem">lessons</span>
        <span id="docs" class="menuItem">Docs</span>
        <span id="fade" class="menuItem">Fade</span>
      </div>
      <div id="content"></div>
    </div>
    <div id="docMenu">
      <span onclick="setDoc('http://api.jquery.com/')">jQuery</span>
      <span onclick="setDoc('http://api.jqueryui.com/')">jQueryUI</span>
      <span onclick="setDoc('http://jquerymobile.com/
➥demos/1.4.0/')">jQueryMobile</span>
    </div>
  </body>
</html>
```

```
$(window).load(function(){
  $("#docMenu").hide();
  $("#lessons").click(setHourNav);
  $("#docs").click(setDocNav);
  $("#fade").click(fade);
});
function setHour(e){
  var hour = $("#lessonSelect").val();
  $("#content p").html("Lesson "+ hour);
}
function setHourNav(){
  $("#docMenu").hide();
  $("span").removeClass("active");
  $("#lessons").addClass("active");
  var select = $('<select id="lessonSelect"></select>');
  select.change(setHour);
  for(var x=1; x<41; x++){
    var option = $("<option></option>");
    option.val(x);
    option.html("Lesson "+x);
    select.append(option);
  }
  $("#content").html("");
  $("#content").append(select).append("<br><p></p>");
  setLesson(5);
}
function setDocNav(){
  $("#docMenu").show();
  $("span").removeClass("active");
  $("#docs").addClass("active");
}
function setDoc(doc){
  var frame = $("<iframe></iframe>");
  frame.attr("src", doc);
  $("#content").html(frame);
}
function fade(){
  var opacity = $("#content").css("opacity");
  if (opacity < 1){ $("#content").css("opacity", 1);}
  else { $("#content").css("opacity", .5); }
}
```

```
#banner{
  height:100px;
  color:white; background-color:blue;
  font-size:40px; text-align:center;
}
#menu, #docMenu{
  background-color:black;
  padding:6px 4px 9px 4px;
}
.menuItem, #docMenu span{
  padding:2px;
  background-image: -moz-linear-gradient(top, #2244ff 0%, #AACCFF 85%, #0022ff
➥100%);
  background-image: -webkit-linear-gradient(top, #2244ff 0%, #AACCFF 85%,
➥#0022ff 100%);
  background-image: -ms-linear-gradient(top, #2244ff 0%, #AACCFF 85%, #0022ff
➥100%);
  font:20px bold;
  cursor:pointer;
}
.active{ border:5px groove; }
#docMenu span{ display:block; margin-top:1px; }
#content, iframe{
  display:inline-block;
  width:700px; height:500px;
}
#container{ width:800px; background-color:#C0C0C0}
#docMenu{
  position:fixed; right:60px; top:60px;
}
#content{
    padding:2px;
    color:blue;
    font-size:20px;
}
```

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

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

```
$(window).load(function(){
  topIndex = $(".photo").length-1;
  maxIndex = topIndex;
  $("#right").click(function(e){move(e, "right");});
  $("#left").click(function(e){move(e, "left");});
  $("#bigger").click(function(e){resize(e, "bigger");});
  $("#smaller").click(function(e){resize(e, "smaller");});
  $("#stack").click(stack);
  $("#tile").click(tile);
  $("#flip").click(flip);
  stack();
});
```

```
function resize(e, direction){
  var img = $("img:eq(" + topIndex + ")");
  if (direction == "bigger"){ img.width(img.width()+20); }
  else { img.width(img.width()-20); }
}
```

```javascript
function move(e, direction){
  var img = $("img:eq(" + topIndex + ")");
  var pos = img.offset();
  if (direction == "right"){ pos.left += 10;}
  else {pos.left -= 10;}
  img.offset(pos);
  startX = pos.left;
  startY = pos.top;
}
```

```
function stack(){
  var x = startX,  y = startY;
  $(".photo").each(function(indx){
    $(this).offset({ top:y, left:x });
    x += 20;
    y += 20;
  });
}
```

```
function tile(){
  var x = startX, y = currTop = startY;
  var maxH = 0;
  $(".photo").each(function(indx){
    maxH = Math.max(maxH, $(this).outerHeight());
    $(this).offset({ top:y, left:x });
    x += $(this).outerWidth();
    if (x > 400){
      y = currTop + maxH;
      x = startX;
      maxH = 0;
    }
  });
}
```

```
function flip(){
  topIndex++;
  if (topIndex > $(".photo").length-1){ topIndex=0; }
  $(".photo").each(function(indx){
    if (indx <= topIndex){ z = maxIndex - (topIndex - indx); }
    else { var z = indx - topIndex - 1; }
    $(this).css("z-index", z);
  });
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Rearranging Elements</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/rearranging_elements.js"></script>
    <link rel="stylesheet" type="text/css" href="css/rearranging_elements.css">
  </head>
  <body>
    <div id="container">
      <span id="left">Left</span>
      <span id="right">Right</span>
      <span id="bigger">Bigger</span>
      <span id="smaller">Smaller</span>
      <span id="stack">Stack</span>
      <span id="flip">Flip</span>
      <span id="tile">Tile</span>
      <div id="photos">
        <img class="photo" src="../images/pyramid.jpg" />
        <img class="photo" src="../images/boy2.jpg" />
        <img class="photo" src="../images/flower.jpg" />
        <img class="photo" src="../images/double.jpg" />
      </div>
    </div>
  </body>
</html>
```

```
var startX = startY = 60;
var topIndex, maxIndex;
$(window).load(function(){
  topIndex = $(".photo").length-1;
  maxIndex = topIndex;
  $("#right").click(function(e){move(e, "right");});
  $("#left").click(function(e){move(e, "left");});
  $("#bigger").click(function(e){resize(e, "bigger");});
  $("#smaller").click(function(e){resize(e, "smaller");});
  $("#stack").click(stack);
  $("#tile").click(tile);
  $("#flip").click(flip);
  stack();
});
function resize(e, direction){
  var img = $("img:eq(" + topIndex + ")");
  if (direction == "bigger"){ img.width(img.width()+20); }
  else { img.width(img.width()-20); }
}
function move(e, direction){
  var img = $("img:eq(" + topIndex + ")");
  var pos = img.offset();
  if (direction == "right"){ pos.left += 10;}
  else {pos.left -= 10;}
  img.offset(pos);
  startX = pos.left;
  startY = pos.top;
}
```

```
function stack(){
  var x = startX,  y = startY;
  $(".photo").each(function(indx){
    $(this).offset({ top:y, left:x });
    x += 20;
    y += 20;
  });
}
function tile(){
  var x = startX, y = currTop = startY;
  var maxH = 0;
  $(".photo").each(function(indx){
    maxH = Math.max(maxH, $(this).outerHeight());
    $(this).offset({ top:y, left:x });
    x += $(this).outerWidth();
    if (x > 400){
      y = currTop + maxH;
      x = startX;
      maxH = 0;
    }
  });
}
function flip(){
  topIndex++;
  if (topIndex > $(".photo").length-1){ topIndex=0; }
  $(".photo").each(function(indx){
    if (indx <= topIndex){ z = maxIndex - (topIndex - indx); }
    else { var z = indx - topIndex - 1; }
    $(this).css("z-index", z);
  });
}
```

```css
.photo{
  border:6px groove;
  width:200px;
  position:absolute; top:40px; left:20px;
}
span{
  padding:5px;
  background-color:steelblue; color:white;
  border-radius:10px 15px; border:2px dotted blue;
  cursor:pointer;
}
#container{ padding:5px; }
```

```
$("img").css({
    "-webkit-transform": "rotate(90deg)",
    "-moz-transform": "rotate(90deg)",
    "filter": "progid:DXImageTransform.Microsoft.BasicImage(rotation=1)"
});
```

```
var fromTop = window.self.screenY;
var fromLeft = window.self.screenX;
var tempWindow = window.open("http://jquery.com");
var oldURL = location.href;
location.assign("http://jquery.com");
```

```
$('a[href^="http://"').click(function (e){
    any of your own handler code . . .
    e.preventDefault();
  });
```

```
$('a[href^="http://"]').attr("target", "_blank");
```

```
$(document).ready(function(){
  $("#set").click(function(e){setCookie($("#cookieName").val(),
                                        $("#cookieValue").val(), 1);});
  $("#get").click(function(e){getCookie($("#cookieName").val());});
  $("#delete").click(function(e){setCookie($("#cookieName").val(), "",
➥-1);});
  displayCookies();
});
```

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

```
function getCookie(name) {
  var cookieStr = $("#cookieName").val() + "=";
  var cArr = document.cookie.split(';');
  for(var i=0;i < cArr.length;i++) {
    var cookie = cArr[i];
    while (cookie.charAt(0)==' '){
      cookie = cookie.substring(1, cookie.length);
    }
    if (cookie.indexOf(cookieStr) == 0){
      $("#cookieValue").val(cookie.substring(cookieStr.length, cookie.
➥length));
      break;
    }
  }
}
```

```
function displayCookies(){
  $("#cookieList").html("");
  var cArr = document.cookie.split(';');
  for(var i=0;i < cArr.length;i++) {
    var cookie = cArr[i];
    $("#cookieList").append($("<li></li>").html(cookie));
  }
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>C is for Cookie</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/cookies.js"></script>
    <link rel="stylesheet" type="text/css" href="css/cookies.css">  </head>
  <body>
    <div>
      <span id="set">Set Cookie</span>
      <span id="get">Get Cookie</span>
      <span id="delete">Delete Cookie</span>
    </div>
    <div>
      <label>Cookie Name: </label><input id="cookieName" type="text" />
    </div>
    <div>
      <label>Cookie Value: </label><input id="cookieValue" type="text" />
    </div>
    <div id="cookieList"></div>
  </body>
</html>
```

```
$(document).ready(function(){
  $("#set").click(function(e){setCookie($("#cookieName").val(),
                                        $("#cookieValue").val(), 1);});
  $("#get").click(function(e){getCookie($("#cookieName").val());});
  $("#delete").click(function(e){setCookie($("#cookieName").val(), "", -1);});
  displayCookies();
});
function setCookie(name, value, days) {
  var date = new Date();
  date.setTime(date.getTime()+(days*24*60*60*1000));
  var expires = "; expires="+date.toGMTString();
  document.cookie = name + "=" + value + expires + "; path=/";
  displayCookies();
}
function getCookie(name) {
  var cookieStr = $("#cookieName").val() + "=";
  var cArr = document.cookie.split(';');
  for(var i=0;i < cArr.length;i++) {
    var cookie = cArr[i];
    while (cookie.charAt(0)==' '){
      cookie = cookie.substring(1, cookie.length);
    }
    if (cookie.indexOf(cookieStr) == 0){
      $("#cookieValue").val(cookie.substring(cookieStr.length, cookie.length));
      break;
    }
  }
}
function displayCookies(){
  $("#cookieList").html("");
  var cArr = document.cookie.split(';');
  for(var i=0;i < cArr.length;i++) {
    var cookie = cArr[i];
    $("#cookieList").append($("<li></li>").html(cookie));
  }
}
```

```
span{
  padding:10px;
  background-color:steelblue; color:white;
  border-radius:10px 20px; border:2px ridge blue;
  cursor:pointer;
}
div{ padding:10px; }
```

```
window.alert("It's 12/12/12 12:12:12!!!");
```

```
var response = window.confirm("Are you sure?");
if (response == true) { do something; }
else { don't do something; }
```

```
var response = window.prompt("What is the airspeed velocity of an unlaiden
➥swallow?");
if (response == "African or European?"){ pass }
else { no pass }
```

```
var timerId = setTimeout(myTimer, 10000);
var timerId = setInterval(checkStatus, 60000);
```

```
$(document).ready(function(){
  setTimeout(continueNotify, 3000);
  setInterval(displayTime, 1000);
});
```

```
function continueNotify(){
  var result = confirm("Do you wIsh to continue\nto receive
➥notifications?");
  if (result==true) { setTimeout(continueNotify, 3000); }
}
```

```
function displayTime(){
  var date = new Date();
  $("#clock").html(padNumber(date.getHours()) +":"+
                   padNumber(date.getMinutes()) +":"+
                   padNumber(date.getSeconds()));
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Clocks</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/clock.js"></script>
    <link rel="stylesheet" type="text/css" href="css/clock.css">
  </head>
  <body>
    <div><span id="clock"></span></div>
  </body>
</html>
```

```
$(document).ready(function(){
  setTimeout(continueNotify, 3000);
  setInterval(displayTime, 1000);
});
function continueNotify(){
  var result = confirm("Do you wish to continue\nto receive notifications?");
  if (result==true) { setTimeout(continueNotify, 3000); }
}
function padNumber(num){
  if (num<10){ return "0"+num; }
  return num;
}
function displayTime(){
  var date = new Date();
  $("#clock").html(padNumber(date.getHours()) +":"+
                   padNumber(date.getMinutes()) +":"+
                   padNumber(date.getSeconds()));
}
```

```
$("span").css("color", getCookie("buttonColor"));
```

```
$("img").animate({height:100, width:100});
```

```
.animate(properties [, duration] [, easing] [, complete])
.animate(properties, options)
```

```
$("img").animate({height:100, width:100}, {duration:1000, easing:"linear"});
```

```
$("img").stop(true, true).animate({opacity:.5}, 1000);
```

```
$("img").animate({width:500}, 1000).delay(2000).animate({opacity:1} 1000);
```

```
$("span").animate({opacity:0}, 30000).promise().done(function(){
    $("p").html("complete");
  });
```

```
$("#box").hide(1000, "linear", function() { $("#label").html("Hidden!") });
```

```
$("#box").show(1000, "linear", function() { $("#label").html("Shown!") });
```

```
$("#switch").toggle(1000, "linear", function() { $("#label").html("Switch
Toggled!") });
```

```
$(window).load(function(){
  $("#handle").click(toggleImage);
});
```

```
function toggleImage(){
  if ($("#handle").html() == '+'){
    $("#photo").show(1000, function(){$("#footer").show();});
    $("#handle").html('-');
  } else {
    $("#footer").hide();
    $("#photo").hide(1000);
    $("#handle").html('+');
  }
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Image Hide</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/image_hide.js"></script>
    <link rel="stylesheet" type="text/css" href="css/image_hide.css">
  </head>
  <body>
    <div>
      <div id="title"><span id="handle">-</span>Canon</div>
      <img id="photo" src="/images/liberty.jpg" width="300px"/>
      <div id="footer">Image Ready</div>
    </div>
  </body>
</html>
```

```
$(window).load(function(){
  $("#handle").click(toggleImage);
});
function toggleImage(){
  if ($("#handle").html() == '+'){
    $("#photo").show(1000, function(){$("#footer").show();});
    $("#handle").html('-');
  } else {
    $("#footer").hide();
    $("#photo").hide(1000);
    $("#handle").html('+');
  }
}
```

```css
div{ width:300px; text-align:center; }
#title, #handle, #footer{
  background-color:blue; color:white;
  font-weight:bold;
}
#handle{
  display:inline-block; width:20px; float:left;
  background-color:black; cursor:pointer;
}
#footer{
  font-size:10px; background-color:black;
  margin-top:-5px
}
```

```
$("img").fadeIn(1000, "swing");
```

```
$("img").fadeOut(1000, "swing", function() { $(this).fadeIn(1000);});
```

```
$("img").fadeToggle(3000, "swing");
```

```
$(window).load(function(){
  $("img").mouseover(function(){$(this).fadeTo(1000, 1);});
  $("img").mouseout(function(){$(this).fadeTo(1000, .3);});
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Image Highlighting and Fading</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/image_fade.js"></script>
    <link rel="stylesheet" type="text/css" href="css/image_fade.css">
  </head>
  <body>
    <div>
      <div id="photos">
      <img src="/images/misty_mountains.jpg"/>
      <img src="/images/falls.jpg"/>
      <img src="/images/flower2.jpg"/>
      <img src="/images/shadow_jump.jpg"/>
      <img src="/images/beachhouse.jpg"/>
    </div>
  </body>
</html>
```

```
$(window).load(function(){
  $("img").mouseover(function(){$(this).fadeTo(1000, 1);});
  $("img").mouseout(function(){$(this).fadeTo(1000, .3);});
});
```

```
$("#menu").slideDown(1000).delay(3000).slideUp(1000);
```

```
$("img").animate({height:100}, 1000);
$("img").animate({height:.1}, 1000);
```

```
$("span").mouseover(function(){
  var i = $(this).index("span");
  $("img").eq(i).animate({height:100}, 1000);
  });
$("span").mouseout(function(){
  var i = $(this).index("span");
  $("img").eq(i).animate({height:.1}, 1000);
  });
```

```
$("#container").mouseenter(function(e){
    e.stopPropagation();
    $("#images").stop(true).slideToggle(1000);
  });
$("#container").mouseleave(function(e){
    e.stopPropagation();
    $("#images").stop(true).slideToggle(1000);
    });
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Sliding Images</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/sliding_images.js"></script>
    <link rel="stylesheet" type="text/css" href="css/sliding_images.css">
  </head>
  <body>
    <div id="container">
      <p>Images</p>
      <div id="images">
        <span>Image 1</span>
        <span>Image 2</span>
        <span>Image 3</span>
        <span>Image 4</span><br>
        <img src="/images/flower.jpg" />
        <img src="/images/img7.jpg" />
        <img src="/images/img4.jpg" />
        <img src="/images/jump.jpg" />
      </div>
    </div>
  </body>
</html>
```

```
$(window).load(function(){
  $("div div").hide();
  $("span").mouseover(function(){
    var i = $(this).index("span");
    $("img").eq(i).animate({height:100}, 2000);
    });
  $("span").mouseout(function(){
    var i = $(this).index("span");
    $("img").eq(i).animate({height:1}, 2000);
    });
  $("#container").mouseenter(function(e){
      e.stopPropagation();
      $("#images").stop(true).slideToggle(1000);
    });
  $("#container").mouseleave(function(e){
      e.stopPropagation();
      $("#images").stop(true).slideToggle(1000);
      });
});
```

```
img{
  display:inline-block; width:150px; height:1px;
  margin:0px; padding:0px; float:left;
}
p, span {
  display:inline-block; width:600px;
  background-color:black; color:white;
  margin:0px; padding:0px; text-align:center;
}
span {
  width:150px; margin:-1px;
  border:1px solid; background-color:blue; float:left;
}
#container { width:610px; }
```

```
$("img").animate({height:500, width:500}, 1000).animate({height:500, width:500},
5000);
```

```
$(window).load(function(){
  $("img").mouseover(function(){
      $(this).animate({width:"200px", opacity:1}, 1000);
    });
  $("img").mouseout(function(){
      $(this).animate({width:"100px", opacity:.3}, 1000);
    });
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Animated Image Resizing</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/animated_resize.js"></script>
    <link rel="stylesheet" type="text/css" href="css/animated_resize.css">
  </head>
  <body>
    <div id="photos">
      <img src="/images/pool.jpg"/>
      <img src="/images/peak.jpg"/>
      <img src="/images/shadow_jump.jpg"/>
      <img src="/images/misty_mountains.jpg"/>
      <img src="/images/tiger.jpg"/>
    </div>
  </body>
</html>
```

```
$(window).load(function(){
  $("img").mouseover(function(){
      $(this).animate({width:"175px", opacity:1}, 1000);
    });
  $("img").mouseout(function(){
      $(this).animate({width:"100px", opacity:.3}, 1000);
    });
});
```

```
$("p").animate({"margin-left":30}, 1000);
```

```
var position = $("#element").offset();
$("#element").animate({top:position.top+10}, 1000);
$("#element").animate({top:position.left+10}, 1000);
```

```
var position = $("#element").offset();
$("#element").animate({top:position.top+10, top:position.left+100}, 1000);
```

```
var rightEdge = window.innerWidth;
var bottomEdge = window.innerHeight;
$(window).load(function(){
$("#ship").offset({top:bottomEdge/2, left:rightEdge/2});
...
});
```

```
$("#up").click(function(){
    $("#ship").attr("src","/images/saucerUp.png");
    $("#ship").stop(true).animate({top:0}, 5000);
  });
$("#left").click(function(){
    $("#ship").attr("src","/images/saucerRight.png");
    $("#ship").stop(true).animate({left:0}, 5000);
  });
$("#right").click(function(){
    $("#ship").attr("src","/images/saucerLeft.png");
    $("#ship").stop(true).animate({left:rightEdge}, 5000);
  });
$("#down").click(function(){
    $("#ship").attr("src","/images/saucerDown.png");
    $("#ship").stop(true).animate({top:bottomEdge}, 5000);
  });
$("#stop").click(function(){ $("#ship").stop(true) });
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>User Interactive Animation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/interactive_animation.js"></script>
    <link rel="stylesheet" type="text/css" href="css/interactive_animation.css">
  </head>
  <body>
    <div>
      <span></span>
      <img id="up" src="/images/up2.png" /></br>
      <img id="left" src="/images/left.png" />
      <img id="stop" src="/images/stop2.png" />
      <img id="right" src="/images/right.png" /><br>
      <span></span>
      <img id="down" src="/images/down2.png" />
      <img id="ship" src="/images/saucerUp.png" />
      <img id="moon1" src="/images/night.png" />
      <img id="moon2" src="/images/night.png" />
      <img id="moon3" src="/images/night.png" />
      <img id="moon4" src="/images/night.png" />
    </div>
  </body>
</html>
```

```
var rightEdge = window.innerWidth;
var bottomEdge = window.innerHeight;
$(window).load(function(){
  $("#ship").offset({top:bottomEdge/2, left:rightEdge/2});
  $("#up").click(function(){
      $("#ship").attr("src","/images/saucerUp.png");
      $("#ship").stop(true).animate({top:0}, 5000);
    });
  $("#left").click(function(){
      $("#ship").attr("src","/images/saucerRight.png");
      $("#ship").stop(true).animate({left:0}, 5000);
    });
  $("#right").click(function(){
      $("#ship").attr("src","/images/saucerLeft.png");
      $("#ship").stop(true).animate({left:rightEdge}, 5000);
    });
  $("#down").click(function(){
      $("#ship").attr("src","/images/saucerDown.png");
      $("#ship").stop(true).animate({top:bottomEdge}, 5000);
    });
  $("#stop").click(function(){ $("#ship").stop(true) });
});
```

```
img{ width:40px;}
span{ width:40px; height:40px; display:inline-block;}
#ship{ position:fixed; width:100px;}
#moon1{ position:fixed; top:100px; left: 500px; width:70px;}
#moon2{ position:fixed; top:200px; left: 100px; width:70px;}
#moon3{ position:fixed; top:300px; left: 300px; width:70px;}
#moon4{ position:fixed; top:400px; left: 600px; width:70px;}
body {
  background-image:url("/images/nightSky.png");
  background-repeat:repeat;
}
```

```
$("input:text").val("New Text");
var newValue = $("input:text").val();
```

```
<input type="text" value="Initial Text"/>
```

```
textDomObj.value = "New Text";
var newValue = textDomObj.value;
```

```
$("input:text").val("New Text");
var newValue = $("input:text").val();
```

```
jObj.prop("checked", false);
```

```
var groupValue = $("input[name=myGroup]").filter(":checked").val();
```

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

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

```
<input id=fileSelect type="file" />
```

```
var fileSelector = document.getElementById("fileSelect");
var fileList = fileSelector.files;
for (var i in fileList){
  var fileObj = fileList[i];
  var fileName = fileObj.name;
  var filePath = fileObj.mozFullPath;
  var fileSize = fileObj.size;
  var fileType = fileObj.type;
}
```

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

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

```
<form id="simpleForm">
  <input name="title" type="text" /><br>
  <select name="mySelection" multiple size=3 id="mySelect">
    <option value="one">One</option>
    <option value="two">Two</option>
    <option value="three">Three</option>
  </select><br>
  <input type="radio" name="gender" value="male">Male</input>
  <input type="radio" name="gender" value="female">Female</input><br>
</form>
```

```
var qString = $("#simpleForm").serialize();
```

```
title=Lumber+Jack&mySelection=one&mySelection=two&gender=male
```

```
var formArr = $("#simpleForm").serializeArray();
```

```
{0: {"name":"title", "value":"Lumber Jack"},
  1: {"name":"mySelection", "value":"one"},
  2: {"name":"mySelection", "value":"two"},
  3: {"name":"gender", "value":"male"}};
```

```
$("#formA input:text").keyup(function(){
  $("#formB input:text").val($(this).val());});
```

```
$("#formA textarea").keyup(function(){
  $("#formB textarea").val($(this).val());});
```

```
$("#formA input:radio").change(function(){
    var radioB = $("#formB input[value=" +
        $(this).val() + "]");
    radioB.prop("checked", $(this).is(":checked"));
  });
```

```
$("#formA input:checkbox").click(function(){
    $("#formB input:checkbox").prop("checked",
        $(this).prop("checked"));
  });
```

```
$("#formA select").change(function(){
  $("#formB select").val($(this).val());});
```

```
$("#formA input:image").click(function(e){
    $("#formB input:image").attr("src", $(this).attr("src"));
    e.preventDefault();
  });
```

```
$("#resetB").click(function(){
    $("#formB").get(0).reset();
    $("#formB input:checked").prop("checked", false);
    $("#formB input:image").attr("src", "");
  });
```

```
$("#serializeB").click(function(e){
    $("#serialized").html($("#formA").serialize());
    $("#serializedA").empty();
    var arr = $("#formA").serializeArray();
    jQuery.each(arr, function(i, prop){
      $("#serializedA").append($("<p>" + prop.name + " = " +
                                 prop.value + "</p>"));
    });
  });
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Manipulation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/
➥jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/form_manipulation.js"></script>
    <link rel="stylesheet" type="text/css" href="css/form_manipulation.css">
  </head>
  <body>
      <div><form id="formA">
        <label>Time</label><br>
        <input type="image" src="/images/day.png" />
        <input type="image" src="/images/night.png" /><br>
        <input name="tilte" type="text" /><br>
        <textarea name="comments"></textarea><br>
        <input type="radio" name="gender" value="male">Male
        <input type="radio" name="gender" value="female">Female<br>
        <input type="checkbox" name="Registered">Registered<br>
        <select size=3 multiple name="count">
          <option>One</option><option>Two</option><option>Three</option>
        </select><br>
        <input id="resetB" type="button" value="Reset"></input>
        <input id="serializeB" type="button" value="Serialize"></input>
      </form></div>
      <div><form id="formB">
        <label>Destination</label><br>
        <input type="image" alt="No Image"></input><br>
        <input type="text" /><br>
        <textarea></textarea><br>
        <input type="radio" name="gender" value="male">Male</input>
        <input type="radio" name="gender" value="female">Female</input><br>
        <input type="checkbox">Checked</input><br>
        <select size=3 multiple>
          <option>One</option><option>Two</option><option>Three</option>
        </select>
      </form></div><br>
      <div><label>Serialized</label><p id="serialized"></p></div>
      <div><label>Serialized Array</label><span id="serializedA"></span></div>
  </body>
</html>
```

```
$(document).ready(function(){
  $("#formA input:text").keyup(function(){
    $("#formB input:text").val($(this).val());});
  $("#formA textarea").keyup(function(){
    $("#formB textarea").val($(this).val());});
  $("#formA input:radio").change(function(){
      var radioB = $("#formB input[value=" +
          $(this).val() + "]");
      radioB.prop("checked", $(this).is(":checked"));
    });
  $("#formA input:checkbox").click(function(){
      $("#formB input:checkbox").prop("checked",
          $(this).prop("checked"));
    });
  $("#formA select").change(function(){
    $("#formB select").val($(this).val());});
  $("#formA label").click(function(){
      $("#formB label").html(new Date().toUTCString());
    });
  $("#formA input:image").click(function(e){
      $("#formB input:image").attr("src", $(this).attr("src"));
      e.preventDefault();
    });
  $("#resetB").click(function(){
      $("#formB").get(0).reset();
      $("#formB input:checked").prop("checked", false);
      $("#formB input:image").attr("src", "");
    });
  $("#serializeB").click(function(e){
      $("#serialized").html($("#formA").serialize());
      $("#serializedA").empty();
      var arr = $("#formA").serializeArray();
      jQuery.each(arr, function(i, prop){
        $("#serializedA").append($("<p>" + prop.name + " = " +
                                   prop.value + "</p>"));
      });
    });
});
```

```
input[type=image] {height:40px; margin-top:15px;}
div{
  vertical-align:top; width:300px; height:auto;
  display:inline-block; padding:20px; margin:5px;
  border-radius:10px; border:1px solid;
}
label{ background-color:blue; color:white;
  border-radius:8px; padding:5px; }
p { margin:1px; padding:2px; width: 100%;
  border-radius:8px; display:inline-block;
  word-wrap: break-word; }
span {width:300px;}
```

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

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

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

```
$("#resetB").click(function(e){
    if(confirm("Are you sure?")){ $("form").get(0).reset(); }
  });
```

```
$("form").submit(function(e){
    alert("Sorry. Not yet Implemented.");
    e.preventDefault();
  });
$("#resetB").click(function(e){
    if(confirm("Are you sure?")){ $("form").get(0).reset(); }
  });
```

```
function updateAddr(){
  var cb = $("#cbSame");
  if (cb.prop("checked")){
    $("#nameB").val($("#name").val());
    $("#addrB").val($("#addr").val());
    $("#cityB").val($("#city").val());
    $("#stateB").val($("#state").val());
    $("#zipB").val($("#zip").val());
    $("#addrB, #cityB, #stateB, #zipB").prop("disabled", "disabled");
  } else{ $("#addrB, #cityB, #stateB, #zipB").prop("disabled", ""); }
}
```

```
function updatePaymentType(){
  if(this.id == "ppal"){
    $("#ccInfo").hide();
    $("#ppInfo").show();
    $("#ppEmail").focus();
  } else {
    $("#ppInfo").hide();
    $("#ccInfo").show();
    $("#cardNum").focus();
  }
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Flow</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/form_flow.js"></script>
    <link rel="stylesheet" type="text/css" href="css/form_flow.css">
  </head>
  <body>
    <div id="box">
      <p>Check Out</p>
      <form>
        <span>Shipping Info</span><br>
          <div id="billInfo">
          <label class="headLabel">Name</label>
          <input type="text" id="name"/><br>
          <label class="headLabel">Address</label>
          <input type="text" id="addr" /><br>
          <label class="headLabel">City</label>
          <input type="text" id="city" />
          <label>State</label>
          <select class="state" id="state"></select>
          <label>Zip</label><input type="text"  id="zip"/><br>
        </div>
        <span>Billing Info</span><br>
        <div id="billInfo">
          <input type="checkbox" id="cbSame"/>
            <label for="cbSame">Same as Shipping</label><br>
          <label class="headLabel">Name on Card</label>
          <input type="text"  id="nameB"/><br>
          <label class="headLabel">Address</label>
          <input type="text"  id="addrB"/><br>
          <label class="headLabel">City</label>
          <input type="text"  id="cityB"/>
          <label>State</label>
          <select class="state"  id="stateB"></select>
          <label>Zip</label><input type="text"  id="zipB"/><br>
          <input type="radio" name="ptype" id="visa" />
            <label for="visa"><img src="/images/visa.png"/></label>
          <input type="radio" name="ptype" id="mc" />
            <label for="mc"><img src="/images/mc.png" /></label>
          <input type="radio" name="ptype" id="amex" />
            <label for="amex"><img src="/images/amex.png"/></label>
          <input type="radio" name="ptype" id="ppal" />
            <label for="ppal"><img src="/images/ppal.png"/></label>
          <br>
          <div id="ccInfo">
            <label class="headLabel">Card Number</label>
            <input type="text" id="cardNum"/>
            <input type="password"  id="csc"/>
            <label>csc</label><br>
            <label>Expires</label><select id="expiresY"></select>
            <select id="expiresM"></select><br>
          </div>
          <div id="ppInfo">
            <input type="text"  id="ppEmail"/>
            <label>PayPal Email</label><br>
            <input type="text"  id="ppPW"/>
            <label>PayPal Password</label><br>
          </div>
        </div>
        <input type="submit" value="Submit" id="submitB" />
        <input type="button" value="Reset" id="resetB" />
      </form>
    </div>
  </body>
</html>
```

```
var months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
              "Aug", "Sep", "Oct", "Nov", "Dec" ];
var sArr = ["AK","AL","AR","AS","AZ","CA","CO","CT","DC","DE","FL",
  "GA","GU","HI","IA","ID","IL","IN","KS","KY","LA","MA","MD","ME","MH","MI","
➥MN",
  "MO","MS","MT","NC","ND","NE","NH","NJ","NM","NV","NY","OH","OK","OR","PA","
➥PR",
  "PW","RI","SC","SD","TN","TX","UT","VA","VI","VT","WA","WI","WV","WY"];

function buildSelects(){
  for(var i in sArr){ $("#state, #stateB").append($('<option>'+sArr[i]+'</
➥option>"')); }
  for(var i in months){ $("#expiresM").append($('<option>'+months[i]+'</
➥option>"')); }
  for(var y=2015; y<2020;y++){ $("#expiresY").append($('<option>'+y+'</
➥option>"')); }
}
function updateAddr(){
  var cb = $("#cbSame");
  if (cb.prop("checked")){
    $("#nameB").val($("#name").val());
    $("#addrB").val($("#addr").val());
    $("#cityB").val($("#city").val());
    $("#stateB").val($("#state").val());
    $("#zipB").val($("#zip").val());
    $("#addrB, #cityB, #stateB, #zipB").prop("disabled", "disabled");
  } else{ $("#addrB, #cityB, #stateB, #zipB").prop("disabled", ""); }
}
```

```
function updatePaymentType(){
  if(this.id == "ppal"){
    $("#ccInfo").hide();
    $("#ppInfo").show();
    $("#ppEmail").focus();
  } else {
    $("#ppInfo").hide();
    $("#ccInfo").show();
    $("#cardNum").focus();
  }
}
$(document).ready(function(){
  $("#ppInfo").hide();
  buildSelects();
  $("#cbSame").click(updateAddr);
  $("input:radio").click(updatePaymentType);
  $("form").submit(function(e){
      alert("Sorry. Not yet Implemented.");
      e.preventDefault();
    });
  $("#resetB").click(function(e){
      if(confirm("Are you sure?")){ $("form").get(0).reset(); }
    });
});
```

```css
input[type=text] {
   width:200px; margin-left:15px; padding-left:10px;
   border-radius: 3px; border:2px groove blue;}
select {margin-left:10px}
img {margin-top:10px; }
#addr, #addrB { width:400px; }
#zip, #zipB { width:60px ; }
#csc { width:40px; }
#box, #billInfo, #shipInfo{
   font:italic 20px/30px Georgia, serif;
   width:650px; height:auto; padding-bottom:20px; margin:10px;
   border-radius: 3px; box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3); }
#billInfo, #shipInfo { width:550px; padding:10px;}
#submitB, #resetB{
   background-color:#3377FF; color:white; font-weight:bold;
   border:2px groove blue; border-radius:15px; }
p{ color:white; background-color:dodgerblue;
   font-weight:bold; margin:0px;
   text-align:center; border-radius: 4px 4px 0px 0px ; }
span{ dispaly:inline-block;
  margin-left:-15px -15px; color:black; font-weight:bold; }
form{ padding:20px; }
.headLabel{ display:block; margin-bottom:-8px;}
label{ color:#000088; font-size:14px;}
```

```
$("form").hide();
$("p").click(function(){$("form").toggle(1000);
  return false;});
$("input:submit").mousedown(function(){
  $("form").toggle(1000); return false;});
```

```
$("textarea").focus(function(){
  $(this).animate({width:350, height:100}, 1000);});
$("textarea").blur(function(){
  $(this).animate({width:200, height:50}, 1000);});
```

```
function changeRadio(){
  $(this).animate({opacity:.1}, 400, function(){
    $("input:radio").next("label").removeClass("rb_checked");
    $(this).addClass("rb_checked");
    $(this).animate({opacity:1}, 800);
  });
}
```

```
function changeCheckbox(){
  var checkbox = $("#"+$(this).attr("for"));
  if(checkbox.prop("checked")){
    $(this).children("img").animate({opacity:.25, height:20, "border-size":1}, 500);}
  else {
    $(this).children("img").animate({opacity:1, height:40, "border-size":.5}, 500); }
}
```

```
$("input:text").keyup(function(){
    $(this).next("label").html($(this).val());});
```

```
$("optgroup").mouseover(function(){
  $(this).stop(true).animate({"font-size":"18px"}, 200);
  return false;});
$("optgroup").mouseout(function(){
  $(this).stop(true).animate({"font-size":"15px"}, 200);
  return false;});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Effects</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/form_effects.js"></script>
    <link rel="stylesheet" type="text/css" href="css/form_effects.css">
  </head>
  <body>
    <div class="collapsibleForm">
      <p>Register</p>
      <form>
        <input type="text" /><label>Name</label><br>
        <input type="text" /><label>From</label><br>
        <input type="radio" name="race" id="man" />
        <label class="rb" for="man">Man</label>
        <input type="radio" name="race" id="elf" />
        <label class="rb" for="elf">Elf</label>
        <input type="radio" name="race" id="dwarf" />
        <label class="rb" for="dwarf">Dwarf</label>
        <input type="radio" name="race" id="orc" />
        <label class="rb" for="Orc">Orc</label><br>
        <label>Weapons</label><br>
        <select size=4 multiple>
```

```
          <optgroup><option>Sword</option></optgroup>
          <optgroup><option>Bow</option></optgroup>
          <optgroup><option>Axe</option></optgroup>
          <optgroup><option>Spear</option></optgroup>
        </select>
        <div>
          <label>Bring Your Own</label><br>
            <input type="checkbox" id="horse" />
              <label class="cb" for="horse">
              <img src="/images/horse.png" />
              <span>Horse</span></label>
            <input type="checkbox" id="shield" />
              <label class="cb" for="shield">
              <img src="/images/shield.png" />
              <span>Armor</span></label><br>
         </div>
        <textarea>Additional Comments</textarea><br>
        <input type="submit" value="Submit" id="submit" />
      </form>
    </div>
  </body>
</html>
```

```
function changeCheckbox(){
  var checkbox = $("#"+$(this).attr("for"));
  if(checkbox.prop("checked")){
    $(this).children("img").animate({opacity:.5, height:40,
      "border-width":"1px"}, 500);}
  else {
    $(this).children("img").animate({opacity:1, height:60,
      "border-width":"2px"}, 500); }
}
function changeRadio(){
  $(this).animate({opacity:.1}, 400, function(){
    $("input:radio").next("label").removeClass("rb_checked");
    $(this).addClass("rb_checked");
    $(this).animate({opacity:1}, 800);
  });
}
$(document).ready(function(){
  $("form").hide();
  $("p").click(function(){$("form").toggle(1000);
  return false;});
  $("input:submit").mousedown(function(){
    $("form").toggle(1000); return false;});
  $("textarea").focus(function(){
    $(this).animate({width:350, height:100}, 1000);});
  $("textarea").blur(function(){
    $(this).animate({width:200, height:50}, 1000);});
  $(".rb").click(changeRadio);
  $("input:checkbox").prop("checked",false);
  $(".cb").click(changeCheckbox);
  $("input:text").keyup(function(){
      $(this).next("label").html($(this).val());});
  $("#submit").mouseover(function(){
    $(this).animate({"background-color":"#0000FF",
      "width":"140px"}, 400, "linear");});
  $("#submit").mouseout(function(){
    $(this).animate({"background-color":"#3377FF",
      "width":"60px"}, 400, "linear");});
  $("#submit").focus(function(){
    $(this).animate({"background-color":"#0000FF",
      "border-width":"5px"}, 400, "linear");});
  $("optgroup").mouseover(function(){
    $(this).stop(true).animate({"font-size":"18px"}, 200);
    return false;});
  $("optgroup").mouseout(function(){
    $(this).stop(true).animate({"font-size":"15px"}, 200);
    return false;});
});
```

```
br{clear:both;}
option, optgroup { padding:0px; margin:0px; height:22px; }
select, textarea, input{
  border-radius: 4px; margin:3px;
  border:2px groove blue; }
select:focus, textarea:focus, input:focus {
  border-radius: 4px; margin:3px;
  border:2px groove #3377FF; }
select{height: 120px; width:100px; text-align:center;}
img{}
input[type="radio"], input[type="checkbox"] {display:none;}
.rb {
  background: -moz-linear-gradient(bottom, #CCCCCC, #EEEEEE 10px);
  background: -webkit-linear-gradient(bottom, #CCCCCC, #EEEEEE 10px);
  background: -ms-linear-gradient(bottom, #CCCCCC, #EEEEEE 10px);
  width: 50px; padding: 3px;
  margin-right:-1px; display: inline-block;
  text-align:center; float:left; border: 1px solid gray; }
.rb_checked {
  background: -moz-linear-gradient(bottom, #0000FF, #7272FF 15px);
  background: -webkit-linear-gradient(bottom, #0000FF, #7272FF 15px);
  background: -ms-linear-gradient(bottom, #0000FF, #7272FF 15px);
  color:white;
}
.cb { padding: 3px; margin-right:-1px; width:100px; }
.cb img{ border:1px dotted; border-radius:4px;
  opacity:.5; height:40px; }
#submit{
  background-color:#3377FF; color:white; font-weight:bold;
  border:2px groove blue; border-radius:15px; }
.collapsibleForm{
  width:400px; height:auto; padding-bottom:20px;
  border-radius: 10px; box-shadow: 0px 0px 8px rgba(0, 0, 0, 0.3);
  font:italic 15px/30px Georgia, serif;}
p{
  color:white; background-color:blue; font-weight:bold;
  margin:0px; text-align:center;
  border-radius: 10px 10px 0px 0px ; }
form{ padding:20px; }
```

```
$("input:text").keypress(function (){
    if(!$.isNumeric($(this).val())) {
      $("#validLabel").html("Not a Number"));
    }
  });
```

```
$("form").submit(function (){
    if (!$("#vfxField").val().match("^vfx")){
      alert("Invalid vfx element");
      e.preventDefault();
    }
  });
```

```
<script type="text/javascript" src="../js/jquery.min.js"></script>
<script type="text/javascript" src="js/jquery.validate.min.js"></script>
```

```
<form id="myForm">
  <input type="text" class="required email" />
</form
```

```
$(document).ready(function(){
    $("#myForm").validate();
  });
```

```
<input type="text" class="required email"
  title="This Element requires a valid Email Address such as name@here.com"/>
```

```
<form id="myForm">
  <input type="text" name="emailField" />
</form
```

```
$(document).ready(function(){
    $("#myForm").validate({
        emailField: {
          required:true,
          email:true
        }
      );
  };
```

```
$(document).ready(function(){
  $("#simpleForm").validate(
    {rules: {
      emailField: {
        required:true,
        email:true,
        accept:"jpg|cvs"
        }
      },
      messages: {
        emailField: {
          required:"Must add Email",
          email:"Email format = me@here.com"
        }
      }
    }
  );
};
```

```
$(document).ready(function(){
  var validator = $("#simpleForm").validate( ...);
  if(!validator.form()){ alert("Form is not valid"));
});
```

```
$("#simpleForm").validate({
  rules: {
    gender: {required:true}
  errorPlacement: function(error, element){
    if (element.is(":radio")){
      error.insertAfter($("input:radio:last").next("label"))}
    else {error.insertAfter(element)}}
});
```

```
    <script type="text/javascript" src="js/jquery.validate.min.js">
➥</script>
```

```
input.error, select.error{ background-color:#FFDDDD; color:darkred}
label.error{color:red;}
```

```
02   var validationObj = $("#simpleForm").validate({
03     rules: {
. . .
12     messages: {
. . .
17     errorPlacement: function(error, element){
. . .
23   });
02   var validationObj = $("#simpleForm").validate({
03     rules: {
. . .
11     messages: {
. . .
16     errorPlacement: function(error, element){
. . .
22   });
```

```
        name: { required:true, minlength:5 },
        email: { required:true, email:true },
        password1: { required:true, rangelength:[6,12],
                     equalTo:"#password2" },
        birthDate: { required:true, date:true },
        class: { required:true, rangelength:[2,2]},
        hands: {required:true},
        armor: {required:true, minlength:2 }},
        password1: { equalTo:"Passwords Do Not Match"},
        class: { rangelength:"Select 2 class types"},
        armor: { minlength:"2 Pieces of Armor Required"},
  errorPlacement: function(error, element){
    if (element.is(":radio")){
      error.insertAfter($("input:radio:last").next("label"))}
    else if (element.is(":checkbox")){
      error.insertAfter($("input:checkbox:last").next("label"))}
    else {error.insertAfter(element)}}
```

```
validationObj.form();
$("form").submit(function(e){
  if(!validationObj.form()){
    alert("Form Errors");
    e.preventDefault(); } });
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Form Validation</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/jquery.validate.min.js"></script>
    <script type="text/javascript" src="js/form_validation.js"></script>
    <link rel="stylesheet" type="text/css" href="css/form_validation.css">
  </head>
  <body>
    <div id="container"><p>Welcome to the Fight</p>
    <form action="test.html" method="get" name="test"
          id="simpleForm">
      <label>Name</label>
      <input name="name" type="text" /><br>
      <label>Email</label>
      <input name="email" type="text" /><br>
      <label>Password</label>
      <input name="password1" type="password" /><br>
      <label>Confirm</label>
      <input name="password2" id="password2" type="password" /><br>
      <label>Birth Date</label>
      <input name="birthDate" type="text" /><br>
      <select name="class" multiple size=4>
        <option>Melee</option>
        <option>Mage</option>
        <option>Ranged</option>
        <option>Stealth</option>
      </select><br>
      <div class="buttonGroup">
          <input type="radio" name="hands" value="right" />
          <label>Right Handed</label>
          <input type="radio" name="hands" value="left" />
          <label>Left Handed</label>
      </div><br>
      <div class="buttonGroup">
        <input type="checkbox" name="armor" value="helmet" />
        <label>Helmet</label>
        <input type="checkbox" name="armor" value="shield" />
        <label>Shield</label><br>
        <input type="checkbox" name="armor" value="chainmail" />
        <label>Chainmail</label>
        <input type="checkbox" name="armor" value="plated" />
        <label>Plated</label><br>
        <input type="checkbox" name="armor" value="gloves" />
        <label>Gloves</label>
        <input type="checkbox" name="armor" value="boots" />
        <label>Boots</label>
      </div><br>
      <label>Comments:</label><br><textarea></textarea><br>
      <input type="submit" value="Engage"/>
    </form>
    </div>
  </body>
</html>
```

```
$(document).ready(function(){
  var validationObj = $("#simpleForm").validate({
    rules: {
      name: { required:true, minlength:5 },
      email: { required:true, email:true },
      password1: { required:true, rangelength:[6,12],
                   equalTo:"#password2" },
      birthDate: { required:true, date:true },
      class: { required:true, rangelength:[2,2]},
      hands: {required:true},
      armor: {required:true, minlength:2 }},
    messages: {
        password1: { equalTo:"Passwords Do Not Match"},
        class: { rangelength:"Select 2 class types"},
        armor: { minlength:"2 Pieces of Armor Required"},
      },
    errorPlacement: function(error, element){
      if (element.is(":radio")){
        error.insertAfter($("input:radio:last").next("label"))}
      else if (element.is(":checkbox")){
        error.insertAfter($("input:checkbox:last").next("label"))}
      else {error.insertAfter(element)}}
  });
  validationObj.form();
  $("form").submit(function(e){
    if(!validationObj.form()){
      alert("Form Errors");
      e.preventDefault(); } });
});
```

```
label{display:inline-block;min-width:100px; text-align:right;}
input+label{text-align:left;}
input.error, select.error{
color:darkblue}
label.error{color:blue;}
br{clear:both;}
form{ margin:15px; margin:0px;
  border: 1px solid darkred;}
div{vertical-align:middle;
  padding-bottom:20px;
  font:italic 15px/30px Georgia, serif;}
#container{
    width:600px; height:auto;}
.buttonGroup { display: inline-block; padding: 5px;
  border-radius: 4px; margin:15px;
  box-shadow: 0px 0px 8px rgba(255, 0, 0, 0.5);}
p{
  color:white; background-color:darkred;
  font-weight:bold; margin:0px;
  text-align:center; border-radius: 4px 4px 0px 0px ; }
select, textarea, input{
  border-radius: 4px; margin:5px;
  border:2px groove crimson; padding:5px;}
select:focus, textarea:focus, input:focus {
  border-radius: 4px; margin:5px;
  border:2px groove #3377FF; }
```

```
#selector {
    max-width:640px; height:140px;
    overflow-x:hidden; overflow-y:hidden;
  background-color:#DDDDDD; border:3px ridge grey; }
#imageSlide {
    position:relative; top:0px; left: 0px; height:100px; }
#imageSlide img {
  height:100px; opacity:.7; vertical-align:top;  margin:10px;
  border:3px ridge grey; box-shadow: 5px 5px 5px #888888; }
```

```
$("#left").mouseenter(function(){
  slide(50); });
$("#left").mouseleave(function(){
  $("#imageSlide").stop(true); return false; });
$("#right").mouseenter(function(){
  slide(-50); });
$("#right").mouseleave(function(){
  $("#imageSlide").stop(true); return false; });
```

```
$("#imageSlide img").mouseenter(function(){
  $(this).stop(true).animate({height:120, opacity:1},500);
                             return false; });
$("#imageSlide img").mouseleave(function(){
  $(this).stop(true).animate({height:100, opacity:.5},500);
                             return false; });
$("#imageSlide img").click(setPhoto);
```

```
$("#imageSlide img:first").click();
```

```
function setPhoto(){
  var newPhoto = $(this).attr("src");
  var horizontal = (minHorizontalRatio >
        $(this).height() / $(this).width());
  $("#photo img").stop(true).fadeTo(500, .1, "linear",
    function (){
      $("#photo img").attr("src", newPhoto); });
  if (horizontal) {
    $("#photo img").css({width:600,height:"auto"}) }
  else {
    $("#photo img").css({width:"auto",height:400}) }
  $("#photo img").fadeTo(500, 1);
  return false;
}
```

```
function slide(value){
  var oldLeft = sliderLeft;
  sliderLeft = sliderLeft + value;
  if (sliderLeft >= 0) { sliderLeft = 0; }
  if (sliderLeft <= sliderMax) { sliderLeft = sliderMax; }
  if(oldLeft != sliderLeft) {
    $("#imageSlide").animate({left:sliderLeft}, 300, 'linear',
        function(){
          slide(value); });
  }
  return false;
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sliding Image Gallery</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/image_slider.js"></script>
    <link rel="stylesheet" type="text/css" href="css/image_slider.css">
  </head>
  <body>
    <div id="viewer">
      <div id="left"><img src="/images/left.png" /></div>
      <div id="selector">
        <div id="imageSlide"></div>
      </div>
      <div id="right"><img src="/images/right.png" /></div><br>
      <div id="photo"><img src="" /></div>
    </div>
  </body>
</html>
```

```
var sliderMax = sliderWidth = sliderLeft = 0;
var minHorizontalRatio = 400/600;
function addImages(){
  var images = ["img3", "img1", "volcano", "peak",
                "river", "wheel", "img7"];
  for (i in images){
    $("#imageSlide").append('<img src="/images/'+
                            images[i] + '.jpg" />'); }
  setTimeout(function() {
      $("#imageSlide img").each(function(){
        sliderWidth += $(this).width() + 26; });
      sliderWidth += 40;
      $("#imageSlide").width(sliderWidth);
      sliderMax = $("#selector").width() - sliderWidth;
    }, 1000);
}
function slide(value){
  var oldLeft = sliderLeft;
  sliderLeft = sliderLeft + value;
  if (sliderLeft >= 0) { sliderLeft = 0; }
  if (sliderLeft <= sliderMax) { sliderLeft = sliderMax; }
  if(oldLeft != sliderLeft) {
    $("#imageSlide").animate({left:sliderLeft}, 300, 'linear',
        function(){
          slide(value); });
  }
  return false;
}
function setPhoto(){
  var newPhoto = $(this).attr("src");
  var horizontal = (minHorizontalRatio >
        $(this).height() / $(this).width());
  $("#photo img").stop(true).fadeTo(500, .1, "linear",
    function (){
      $("#photo img").attr("src", newPhoto); });
  if (horizontal) {
    $("#photo img").css({width:600,height:"auto"}) }
  else {
    $("#photo img").css({width:"auto",height:400}) }
  $("#photo img").fadeTo(500, 1);
  return false;
}
$(document).ready(function(){
  addImages();
  $("#left").mouseenter(function(){
    slide(50); });
  $("#left").mouseleave(function(){
    $("#imageSlide").stop(true); return false; });
  $("#right").mouseenter(function(){
    slide(-50); });
  $("#right").mouseleave(function(){
    $("#imageSlide").stop(true); return false; });
  $("#imageSlide img").mouseenter(function(){
    $(this).stop(true).animate({height:120, opacity:1},500);
                               return false; });
  $("#imageSlide img").mouseleave(function(){
    $(this).stop(true).animate({height:100, opacity:.5},500);
                               return false; });
  $("#imageSlide img").click(setPhoto);
  $("#imageSlide img:first").click();
});
```

```css
div {
    display:inline-block; }
#viewer2 {
    background-color:black; border:10px ridge blue; }
#right, #left {
    width:30px; height:100px; float:left; color:white; }
#right {
    float:right; }
#right img, #left img {
    margin-top:30px; width:32px; height:56px; }
#selector {
    max-width:640px; height:140px;
    overflow-x:hidden; overflow-y:hidden;
  background-color:#DDDDDD; border:3px ridge grey; }
#imageSlide {
    position:relative; top:0px; left: 0px; height:100px; }
#imageSlide img {
  height:100px; opacity:.7; vertical-align:top;  margin:10px;
  border:3px ridge grey; box-shadow: 5px 5px 5px #888888; }
#photo {
  height:500px; width:700px;
  display:table-cell; vertical-align:middle; text-align:center;
}
#photo img {
    border:5px ridge grey; box-shadow: 10px 10px 5px darkgrey; }
```

```
input {
    height:20px; width: 20px; border-radius:15px;
    padding: 0 7px 2px 25px;
  background:url("/images/search.png") no-repeat scroll left bottom 0
➥#FFFFFF;
  background-size:20px 20px;
}
input:focus {
    width:100px; }
span {
    background:url("/images/sort2.png") no-repeat scroll left;
  background-size:20px 20px;
  padding: 0 7px 2px 25px;
}
.ascending{
    background:url("/images/up2.png") no-repeat scroll left;
  background-size:20px 20px;
}
.descending{
    background:url("/images/down2.png") no-repeat scroll left;
  background-size:20px 20px;
}
```

```
$(document).ready(function(){
  buildData();
  $("th").each(function(i) {
      var header = $(this);
      header.data({numeric:header.hasClass("numeric"), order:-1});
      header.children("span").click(function(){
        sortColumn(header, i); });
      var filter = $('<input type="text" />');
      filter.keyup(function(){
        filterColumn(filter, i); });
      header.append(filter);
    });
});
```

```
function filterColumn(input, column){
  $("tbody tr").show().each(function(){
    var header = $("th:eq("+ column +")");
    var filterVal = input.val();
    var rowVal = this.cells[column].innerHTML;
    if(header.data("numeric") ){
      if(parseFloat( filterVal ) > parseFloat( rowVal )) {
        $(this).hide(); }}
    else {
      if(rowVal.indexOf(filterVal) < 0) { $(this).hide(); }}
  });
}
```

```
function compare(a, b, column, numeric, order){
  var aValue = a.cells[column].innerHTML;
  var bValue = b.cells[column].innerHTML;
  if (numeric) { aValue = parseFloat(aValue );
                 bValue = parseFloat(bValue ); }
  if (aValue < bValue) return order;
  if (aValue > bValue) return -order;
  return 0;
}
```

```
function sortColumn(header, column){
  var rows = $("tbody tr");
  rows.sort(function(a, b){
      return compare(a, b, column, header.data("numeric"),
                     header.data("order"));
    });
  $(rows).each(function(){ $("tbody").append($(this)); });
  header.data("order", -header.data("order"));
  $("span").removeClass("ascending descending");
  $("td").removeClass("sortColumn");
  if(header.data("order") > 0) {
    header.children("span").addClass("ascending"); }
  else {
    header.children("span").addClass("descending"); }
  $("tbody tr td:nth-child("+ (column + 1) +")")
    .addClass("sortColumn");
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Interactive Table</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/interactive_table.js"></script>
    <link rel="stylesheet" type="text/css" href="css/interactive_table.css">
  </head>
  <body>
    <table>
      <thead><tr>
        <th class="numeric"><span>ID#</span></th>
        <th ><span>Product</span></th>
        <th class="numeric"><span>Quantity</span></th>
        <th class="numeric"><span>Price</span></th>
        <td class="spacer"></td>
      </tr></thead>
      <tbody></tbody>
    </table>
    <p>* ID#, Quantity and Price Searches Filter
       on Numerical Value Greater Than Value Specified</p>
    <p>* Product Search Filters on String Value Contains
       Substring Specified</p>
  </body>
</html>
```

```
var tArr = ["Mens", "Womens", "Youth", "Childs"];
var sArr = ["XL", "M", "S", "XS"];
var kArr = ["pants", "shirt", "shoes", "socks", "sweater", "belt"];
function compare(a, b, column, numeric, order){
  var aValue = a.cells[column].innerHTML;
  var bValue = b.cells[column].innerHTML;
  if (numeric) { aValue = parseFloat(aValue );
                 bValue = parseFloat(bValue ); }
  if (aValue < bValue) return order;
  if (aValue > bValue) return -order;
  return 0;
}
function sortColumn(header, column){
  var rows = $("tbody tr");
  rows.sort(function(a, b){
      return compare(a, b, column, header.data("numeric"),
                     header.data("order"));
    });
  $(rows).each(function(){ $("tbody").append($(this)); });
  header.data("order", -header.data("order"));
  $("span").removeClass("ascending descending");
  $("td").removeClass("sortColumn");
  if(header.data("order") > 0) {
    header.children("span").addClass("ascending"); }
  else {
    header.children("span").addClass("descending"); }
  $("tbody tr td:nth-child("+ (column + 1) +")")
    .addClass("sortColumn");
}
function filterColumn(input, column){
  $("tbody tr").show().each(function(){
    var header = $("th:eq("+ column +")");
    var filterVal = input.val();
    var rowVal = this.cells[column].innerHTML;
    if(header.data("numeric") ){
      if(parseFloat( filterVal ) >= parseFloat( rowVal )) {
        $(this).hide(); }}
    else {
      if(rowVal.indexOf(filterVal) < 0) { $(this).hide(); }}
  });
}
function randInt(max) {
  return Math.floor((Math.random()*max)+1); }
function buildData(){
  for(var x=1;x<26;x++){
      var row =$("<tr></tr>");
      row.append($("<td></td>").html(x));
      row.append($("<td></td>").html(
        tArr[randInt(3)] + " " + sArr[randInt(3)] +
        " " + kArr[randInt(5)]));
      row.append($("<td></td>").html(randInt(20)));
      row.append($("<td></td>")
          .html(((Math.random()*80)+5).toFixed(2)));
      $("tbody").append(row);}
}
$(document).ready(function(){
  buildData();
  $("th").each(function(i) {
      var header = $(this);
      header.data({numeric:header.hasClass("numeric"), order:-1});
      header.children("span").click(function(){
        sortColumn(header, i); });
      var filter = $('<input type="text" />');
      filter.keyup(function(){
        filterColumn(filter, i); });
      header.append(filter);
    });
});
```

```
table{
    border:3px ridge steelblue; padding:0px;}
thead {
    display:block; width:820px; text-align:left; }
tbody {
    display:block;  max-height:400px; width:820px;
    overflow-y:scroll; }
th {
    background-image: -moz-linear-gradient(top , #f1f1f1, #BFBFBF);
  background-image: -webkit-linear-gradient(top , #f1f1f1, #BFBFBF);
  background-image: -ms-linear-gradient(top , #f1f1f1, #BFBFBF);
  width:200px; height:30px; max-width: 200px;
  font:16px Arial Black; padding:3px;
}
.spacer {
    background-image: -moz-linear-gradient(top , #f1f1f1, #BFBFBF);
  background-image: -webkit-linear-gradient(top , #f1f1f1, #BFBFBF);
  background-image: -ms-linear-gradient(top , #f1f1f1, #BFBFBF);
  width:15px; border:none;
}
td {
    width:200px; border: .5px dotted; }
.sortColumn {
    font-weight:bold; }
tr:nth-child(even){
    background-color:lightgrey; }
img {
    height:26px; }
```

```
input {
  height:20px; width: 20px; border-radius:15px;
  padding: 0 7px 2px 25px;
  background:url("/images/search.png") no-repeat scroll left bottom 0 #FFFFFF;
  background-size:20px 20px;
}
input:focus {
    width:100px; }
span {
  background:url("/images/sort2.png") no-repeat scroll left;
  background-size:20px 20px;
  padding: 0 7px 2px 25px;
  cursor: pointer;
}
.ascending{
    background:url("/images/up2.png") no-repeat scroll left;
  background-size:20px 20px;
}
.descending{
    background:url("/images/down2.png") no-repeat scroll left;
  background-size:20px 20px;
}
```

```
<div class="tree">
  <span></span><label></label>
  <div class="content"></div>
  <div class="tree">one or more child elemnts</div>
</div>
```

```
$(document).ready(function(){
  var root = addLevels($("#treeContainer"), 4);
  root.show();
});
```

```
function addItem(parentItem, item){
  var newItem = $('<div class="tree"></div>').hide();
  newItem.append($("<span></span>").click(toggleItem));
  newItem.append($("<label></label>").html(item.label));
  newItem.append($('<div class="content"></div>')
      .append(item.content).hide());
  parentItem.append(newItem);
  return newItem;
}
```

```
function toggleItem(){
  $(this).parent().children("div").toggle();
  $(this).toggleClass("collapse");
  return false;
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Creating a Dynamic Tree</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/dynamic_tree.js"></script>
    <link rel="stylesheet" type="text/css" href="css/dynamic_tree.css">
  </head>
  <body>
    <div id="treeContainer"></div>
  </body>
</html>
```

```
function toggleItem(){
  $(this).parent().children("div").toggle();
  $(this).toggleClass("collapse");
  return false;
}
function addItem(parentItem, item){
  var newItem = $('<div class="tree"></div>').hide();
  newItem.append($("<span></span>").click(toggleItem));
  newItem.append($("<label></label>").html(item.label));
  newItem.append($('<div class="content"></div>')
      .append(item.content).hide());
  parentItem.append(newItem);
  return newItem;
}
function randInt(max) {
  return Math.floor((Math.random()*max)); }
function getRandomItem(){
  var itemTypes=["image", "input", "textarea"];
  var images=["volcano.jpg", "liberty.jpg", "falls2.jpg",
              "wheel.jpg", "sunset.jpg"];
  var inputs=["text","checkbox", "radio"];
  switch(itemTypes[randInt(3)]){
    case "image":
      var img = images[randInt(5)];
      return { label:img,
               content:$('<img src="/images/' + img + '" />')};
    case "input":
      var type = inputs[randInt(3)];
      return { label:type,
               content:$('<input type="'+type+'">'+type+
                         '</input>')};
    case "textarea":
      return { label:"textarea",
               content:$('<textarea>textarea</textarea>')};
  }
}
function addLevels(parent, levels){
  var element = addItem(parent, getRandomItem());
  if( levels > 0 ){
    for(var x=0; x<5; x++){ addLevels( element, levels-1 ) }; }
  return element;
}
$(document).ready(function(){
  var root = addLevels($("#treeContainer"), 4);
  root.show();
});
```

```css
#treeContainer {
    width:300px; }
.tree {
    margin-left: 16px; border-top:1px dotted;
    border-left:1px dotted;}
.content {
    margin-left: 48px; }
.tree span {
  display:inline-block; height:24px; width:24px;
  border-radius:8px; vertical-align:middle; margin-right:10px;
  background:url("/images/expand.png") no-repeat scroll 0px 0px/15px 15px;
}
.tree span.collapse {
  background:url("/images/collapse.png") no-repeat scroll 0px 0px/15px 15px;
}
img {
    height:50px; }
.tree label {
    font: 15px/20px "Arial Black";}
```

```
$(document).ready(function(){
  $("#overlay, #dialog").hide();
  $("span").click(function(){
    $("#overlay, #dialog").show(); });
  $("#updateB").click(update);
});
```

```
function update(){
  $("#overlay, #dialog").hide();
  $("#title p").html($("#titleT").val());
  $("#content").html($("#contentT").val());
  $("#leftNav span").remove();
  $("input:checkbox").each(function(){
    if($(this).prop("checked")){
      $("#leftNav").append($("<span></span>")
          .html($(this).val()));
    }
  });
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Let's Have a Dialog</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/dynamic_dialog.js"></script>
    <link rel="stylesheet" type="text/css" href="css/dynamic_dialog.css">
  </head>
  <body>
    <div><div id="title">
        <span id="edit">Edit Page</span><p>Title</p></div><br>
    <div>
      <div id="leftNav">
        <span>Option 1</span>
        <span>Option 2</span>
        <span>Option 3</span>
        <span>Option 4</span>
      </div>
      <div id="content">Some<br>Content</div>
    </div></div>
    <div id="overlay"></div>
    <div id="dialog">
      <p id="dialogTitle">Update Web Page</p>
      <label>Title</label><br>
      <input id="titleT" type="text" /><br>
      <input type="checkbox" value="Option 1" checked>Option 1</input><br>
      <input type="checkbox" value="Option 2" checked>Option 2</input><br>
      <input type="checkbox" value="Option 3" checked>Option 3</input><br>
      <input type="checkbox" value="Option 4" checked>Option 4</input><br>
      <label>Content</label><br>
      <textarea id="contentT"></textarea><br>
      <span id="updateB">Update</span>
    </div>
  </body>
</html>
```

```
function update(){
  $("#overlay, #dialog").hide();
  $("#title p").html($("#titleT").val());
  $("#content").html($("#contentT").val());
  $("#leftNav span").remove();
  $("input:checkbox").each(function(){
    if($(this).prop("checked")){
      $("#leftNav").append($("<span></span>")
          .html($(this).val()));
    }
  });
}
$(document).ready(function(){
  $("#overlay, #dialog").hide();
  $("#edit").click(function(){
    $("#overlay, #dialog").show(); });
  $("#updateB").click(update);
});
```

```
div {
    margin:0px; display:inline-block;
    float:left; text-align:center; }
span {
    background-image: -moz-linear-gradient(top , #f1f1f1, #8F8F8F);
  background-image: -webkit-linear-gradient(top , #f1f1f1, #8F8F8F);
  background-image: -ms-linear-gradient(top , #f1f1f1, #8F8F8F);
  color:black; border:2px ridge darkblue;
  font-size:20px; float:left; cursor:pointer;
  width:145px; text-align:center; border-radius: 4px; }
p {
    margin:0px; }
#title {
    background-color:steelblue; color:white;
    height:80px; width:750px; font-size:60px; }
#leftNav{
    width:150px; height:400px; font-size:20px;
    background-color:#AACCFF; }
#content{
    height:400px; width: 600px; font-size:40px;
    background-color:#EEEEEE; }
#overlay {
    position:fixed; top:10px; left:10px;
    height:480px; width:750px;
  opacity:.8; background-color:white; }
#dialog {
    border: 3px groove blue;  text-align:left;
    padding:10px; position:fixed;
    top:100px; left:200px; background-color:white; }
#dialogTitle {
    text-align:center; font:20px bold;
    background-color:blue; color:white;
    margin:-10px -10px 5px -10px; }
#contentT, #titleT {
    border-radius: 3px;  width:150px; padding:5px;
  margin:10px; border:2px groove blue; }
label {
    font:bold italic 18px "Arial Black" }
```

```
$(document).ready(function(){
  for(var i=0; i< 10; i++){
    $("#equalizer").append($("<div></div>"));
  }
  $("#equalizer div").each(function (idx){
    $(this).append($("<p></p>").html(idx));
    for(var i=0; i< 2; i++){
      $(this).append($('<span class="red"></span>')); }
    for(var i=0; i< 2; i++){
      $(this).append($('<span class="orange"></span>')); }
    for(var i=0; i< 3; i++){
      $(this).append($('<span class="yellow"></span>')); }
    for(var i=0; i< 8; i++){
      $(this).append($('<span class="green"></span>')); }
    adjValues();
  });
});
```

```
function updateEqualizer(){
  $("span").css({opacity:.3});
  $("#equalizer div").each(function(i){
    $(this).children("span:gt("+ (15 - valueArr[i]) +")")
      .css({opacity:1});
    $(this).children("p:first").html(valueArr[i]);
  });
}
```

```
var valueArr = [10,8,3,12,12,15,15,3,4,5];
function randInt(max) {
  return Math.floor((Math.random()*max)-3); }
function adjValues(){
  for (var i=0; i<valueArr.length; i++) {
      var adj = valueArr[i] +
                Math.floor((Math.random()*7)-3);
      adj = Math.max(3, adj);
      adj = Math.min(15, adj);
      valueArr[i] = adj;
    }
  updateEqualizer();
  setTimeout(adjValues, 500);
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Listen to the Beat!</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/graphic_equalizer.js"></script>
    <link rel="stylesheet" type="text/css" href="css/graphic_equalizer.css">
  </head>
  <body>
    <div id="equalizer"></div>
  </body>
</html>
```

```
var valueArr = [10,8,3,12,12,15,15,3,4,5];
function randInt(max) {
  return Math.floor((Math.random()*max)-3); }
function adjValues(){
  for (var i=0; i<valueArr.length; i++) {
      var adj = valueArr[i] +
                Math.floor((Math.random()*7)-3);
      adj = Math.max(3, adj);
      adj = Math.min(15, adj);
      valueArr[i] = adj;
    }
  updateEqualizer();
  setTimeout(adjValues, 500);
}
function updateEqualizer(){
  $("span").css({opacity:.3});
  $("#equalizer div").each(function(i){
    $(this).children("span:gt("+ (15 - valueArr[i]) +")")
      .css({opacity:1});
    $(this).children("p:first").html(valueArr[i]);
  });
}
$(document).ready(function(){
  for(var i=0; i< 10; i++){
    $("#equalizer").append($("<div></div>"));
  }
  $("#equalizer div").each(function (idx){
    $(this).append($("<p></p>").html(idx));
    for(var i=0; i< 2; i++){
      $(this).append($('<span class="red"></span>')); }
    for(var i=0; i< 2; i++){
      $(this).append($('<span class="orange"></span>')); }
    for(var i=0; i< 3; i++){
      $(this).append($('<span class="yellow"></span>')); }
    for(var i=0; i< 8; i++){
      $(this).append($('<span class="green"></span>')); }
    adjValues();
  });
});
```

```
#equalizer {
    background-color:black; color:white;
  width:420px; height:160px; padding:20px;
  border: 3px ridge darkgrey;
}
div{
    display:inline-block; width:40px;  padding:1px; }
p{
    text-align:center; margin:0px;}
span{
    display:block; width:30px; height:7px;
  margin:2px; border-radius: 40%; }
.green{
    background-color:#00FF00; }
.yellow{
    background-color:#FFFF00; }
.orange{
    background-color:#FFAA00; }
.red{
    background-color:#FF0000; }
```

```
$(document).ready(function(){
  $("div").each(function(){
    $(this).data("valueArr", getRandomArray()); });
  adjValues();
});
```

```
function renderSpark(c, lineValues){
  c.width = c.width;
  var xAdj = c.width/lineValues.length;
  var ctx = c.getContext("2d");
  ctx.fillStyle = "#000000";
  ctx.strokeStyle = "#00ffff";
  ctx.lineWidth = 3;
  var x = 1;
  ctx.moveTo(x,(c.height));
  for (var idx in lineValues){
    var value = parseInt(lineValues[idx]);
    ctx.lineTo(x+xAdj, (c.height - value));
    x += xAdj;
  }
  ctx.stroke();
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dynamic Spark Line</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/dynamic_sparkline.js"></script>
    <link rel="stylesheet" type="text/css" href="css/dynamic_sparkline.css">
  </head>
  <body>
    <div>
      <label>Processes</label><span>1</span><canvas></canvas>
    </div>
    <div>
      <label>Speed</label><span></span><canvas></canvas>
    </div>
    <div>
      <label>Uploads</label><span></span><canvas></canvas>
    </div>
    <div>
      <label>Downloads</label><span></span><canvas></canvas>
    </div>
  </body>
</html>
```

```
function randInt(max) {
  return Math.floor((Math.random()*max)+1); }
function adjValues(){
  $("div").each(function(){
    var lineValues = $(this).data("valueArr");
    lineValues.shift();
    lineValues.push(randInt(100));
    $(this).children("span").html(lineValues[0]);
    renderSpark($(this).children("canvas").get(0), lineValues);
  });
  setTimeout(adjValues, 1000);
}
function renderSpark(c, lineValues){
  c.width = c.width;
  var xAdj = c.width/lineValues.length;
  var ctx = c.getContext("2d");
  ctx.fillStyle = "#000000";
  ctx.strokeStyle = "#00ffff";
  ctx.lineWidth = 3;
  var x = 1;
  ctx.moveTo(x,(c.height));
  for (var idx in lineValues){
    var value = parseInt(lineValues[idx]);
    ctx.lineTo(x+xAdj, (c.height - value));
    x += xAdj;
  }
  ctx.stroke();
}
function getRandomArray(){
  var arr = new Array();
  for(var x=0; x<20; x++){ arr.push(randInt(100)); }
  return arr;
}
$(document).ready(function(){
  $("div").each(function(){
    $(this).data("valueArr", getRandomArray()); });
  adjValues();
});
```

```
canvas{
    height:50px; width: 200px; vertical-align:bottom;
  border:3px solid black; background-color:black; margin:10px; }
label, span {
    display:inline-block; text-align:right; width:160px;
  font:bold 24px/50px "Arial Black"; border-bottom:2px dotted; }
span{
    width:50px; color:blue; }
```

```
<script type="text/javascript"
        src="http://new.domain.com/getData?jsonp=parseData">
</script>
```

```
{ "title": "photoA", "rating": 7 };
```

```
parseData({ "title": "photoA", "rating": 7 });
```

```
http://localhost/code/example1.html?first=Brad&last=Dayley
```

```
http://localhost/code/example1.html
```

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    { document.getElementById("email").innerHTML=xmlhttp.responseText;}
  }
xmlhttp.open("GET","/getUserEmail?userid=brad",true);
xmlhttp.send();
```

```
var xmlhttp = new XMLHttpRequest();
var params = "first=Brad&last=Dayley&email=brad@dayleycreations.com";
http.setRequestHeader("Content-type", "text/plain");
http.setRequestHeader("Content-length", params.length);
xmlhttp.onreadystatechange=function() {
  if (xmlhttp.readyState==4 && xmlhttp.status==200)
    { alert("Email Updated");
  }
xmlhttp.open("POST","/setUserEmail",true);
xmlhttp.send(params);
```

```
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
```

```
var obj ={"first":"Brad", "last":"Dayley"};
var objString = $.param(obj);
var formString = $("form").serialize();
```

```
$.get("/getEmail?first=Brad&last=Dayley", null, function (data, status, xObj){
  $("#email").html(data);
}));
```

```
$.get("/getUser?first=Brad&last=Dayley", null, function (data, status, xObj){
  $("#email").html(data.email);
}), "json");
```

```
        <span class="navItem"
              article="article1">Responsive Web Design</span>
```

```
$(document).ready(function(){
  $(".navItem").click(setArticle);
});
```

```
function setArticle(){
  $("#content").load("data/"+$(this).attr("article")+".html");
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Loading Content</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/load_content.js"></script>
    <link rel="stylesheet" type="text/css" href="css/load_content.css">
  </head>
  <body>
    <div id="banner">AngularJS, JavaScript and jQuery Articles</div>
    <br>
    <div>
      <div id="leftNav">
        <span class="navItem"
              article="article1">Responsive Web Design</span>
        <span class="navItem"
              article="article2">jQUery Under the Hood</span>
        <span class="navItem"
              article="article3">AngularJS Your New Best Friend</span>
        <span class="navItem"
              article="article4">Turbo Charged Web Sites</span>
      </div>
      <div id="content">
        <span id="title">&nbsp;</span>
        <div id="article"></div>
      </div>
    </div>
  </body>
</html>
```

```
function setArticle(){
  $("#content").load("data/"+$(this).attr("article")+".html");
}
$(document).ready(function(){
  $(".navItem").click(setArticle);
});
```

```css
div { margin:0px; display:inline-block; float:left; text-align:center; }
p { margin:2px; }
#banner { border-radius: 3px 3px 0px 0px;
  background-image: -moz-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -webkit-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -ms-linear-gradient(top , #0000FF, #88BBFF);
  color:white; height:30px; width:550px; font-size:25px; }
#leftNav { width:150px; height:404px; border:1px groove #000088; }
.navItem { border:1px dotted; display:block; margin:3px; }
.navItem:hover { border:1px solid; background-color:#00FF00; cursor:pointer; }
#content {border:1px solid blue;}
#article { width: 375px; height:350px; padding:10px; overflow-y:scroll; }
#title { font-weight:bold; font-size:25px; border-bottom: 1px blue solid;
  display:block; margin:5px; color:black; }
#by { text-align:right; font:bold italic 16px arial black; float:left;
  margin-bottom:20px; }
#date { text-align:right; font:italic 12px arial black; float:right;}
#article p {margin-top:20px; background-color:#EEEEEE; border-radius:5px;
  clear:both; padding:5px; }
```

```
<span id=title>Responsive Web Design</span>
<div id="article">
<span id="by">Brad Dayley</span>
<span id="date">5/25/2015</span>
<p>Lorem ipsum dolor ... </p>
<p>Lorem ipsum dolor sit amet, ...</p>
<p>...</p>
<p>...</p>
<p>....</p>
</div>
```

```
$.get("/getUser?first=Brad&last=Dayley", null, function (data, status, xObj){
  $("#email").html(data.email);
}), "json").fail(function(data, status, xObj){
  alert("Request Failed");
});
```

```
$(document).ready(function(){
  $("#requestButton").click(askYourQuestions);
});
```

```
function askYourQuestions(){
  $.get("/request",
   $("#requestForm").serialize()).done(success).fail(failure).
➥always(always);
  return false;
}
```

```
function failure(){ alert("You May Not Pass!"); }
function success(){ alert("You May Pass!"); }
function always(){ alert("Questions Answered."); }
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>AJAX Error Handling</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/
➥jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/ajax_response.js"></script>
    <link rel="stylesheet" type="text/css" href="css/ajax_response.css">
  </head>
  <body>
    <div id="request">
      <div id="title">Gorge of Eternal Peril</div>
      <form id="requestForm">
        <label>What is your name? </label>
        <input type="text" name="name" /><br>
        <label>What is your favorite color? </label>
        <input type="text" name="color" /><br>
        <input id="requestButton" type="button" value="Request" />
      </form>
    </div>
  </body>
</html>
```

```
function failure(){ alert("You May Not Pass!"); }
function success(){ alert("You May Pass!"); }
function always(){ alert("Questions Answered."); }
function askYourQuestions(){
  $.get("/request",
   $("#requestForm").serialize()).done(success).fail(failure).always(always);
  return false;
}
$(document).ready(function(){
  $("#requestButton").click(askYourQuestions);
});
```

```
#request {
  height:180px; width:350px; border: 3px ridge blue; }
#title {
    text-align:center;
  background-image: -moz-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -webkit-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -ms-linear-gradient(top , #0000FF, #88BBFF);
  height:30px; color:white;  font:bold 22px arial black; }
input {
    margin-top:10px;
  margin-left:30px;
  padding-left:10px; }
label {
    margin-left:10px;
    font:italic 18px arial black; }
```

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use('/', express.static('./'));
app.get('/request', function(req, res){
  var queryObj = req.query;
  if (queryObj.name == "Lancalot" && queryObj.color == "blue"){
    res.send(200, "Welcome To AJAX!");
  } else {
    res.send(400, "Invalid Answers!");
  }
});
app.listen(80);
```

```
{"first":"Brad", "last":"Dayley"}
```

```
var name  = data.first + " " + data.last;
var data = $.parseJSON('{"first":"Brad", "last":"Dayley"}');
var name  = data.first + " " + data.last;
```

```
$(document).ready(function(){
  $.get("data/images.json", updateImages);
});
```

```
function updateImages(data){
  for (i=0; i<data.length; i++){
    var imageInfo =data[i];
    var img = $('<img />').attr("src", "images/"+imageInfo.image);
    var title = $("<p></p>").html(imageInfo.title);
    var div = $("<div></div>").append(img, title);
    $("#images").append(div);
  }
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Loading JSON Data</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/load_json.js"></script>
    <link rel="stylesheet" type="text/css" href="css/load_json.css">
  </head>
  <body>
    <div id="images"></div>
  </body>
</html>
```

```
function updateImages(data){
  for (i=0; i<data.length; i++){
    var imageInfo =data[i];
    var img = $('<img />').attr("src", "/images/"+imageInfo.image);
    var title = $("<p></p>").html(imageInfo.title);
    var div = $("<div></div>").append(img, title);
    $("#images").append(div);
  }
}
$(document).ready(function(){
  $.get("data/images.json", updateImages);
});
```

```
[
  {"title":"Quiet Strength", "image":"img7.jpg"},
  {"title":"Great Heights", "image":"misty_mountains.jpg"},
  {"title":"Summer Fun", "image":"boy.jpg"},
  {"title":"Grandeur of Nature", "image":"falls2.jpg"},
  {"title":"Soft Perfection", "image":"flower.jpg"},
  {"title":"Courage", "image":"power.jpg"},
  {"title":"Joy of Finishing", "image":"shadow_jump.jpg"}
]
```

```
div {border:3px ridge white; box-shadow: 5px 5px 5px #888888;
  display:inline-block; margin:10px; }
p { background-image: -moz-linear-gradient(top , #B1B1B1, #FFFFFF);
  background-image: -webkit-linear-gradient(top , #B1B1B1, #FFFFFF);
  background-image: -ms-linear-gradient(top , #B1B1B1, #FFFFFF);
  margin:0px; padding:3px; text-align:center; }
img { height:130px; vertical-align:top;  }
#images { background-color:black; padding:20px; }
```

```
<person><first>Brad</first><last>Dayley</last></person>
```

```
var name  = $(data).find("first").text() + " " + $(data).find("last").text();
var data = $.parseXML("<person><first>Brad</first><last>Dayley</last></person>");
var name  = $(data).find("first").text() + " " + $(data).find("last").text();
```

```
$(document).ready(function(){
  $.get("data/parkdata.xml", updateTable);
});
```

```
function updateTable(data){
  var parks = $(data).find("park");
  parks.each(function(){
    var tr = $("<tr></tr>");
    tr.append($("<td></td>").html($(this).children("name").text()));
    tr.append($("<td></td>").html($(this).children("location").text()));
    tr.append($("<td></td>").html($(this).children("established").text()));
    var img = $('<img />').attr("src", "images/"+$(this).children("image").
➥text());
    tr.append($("<td></td>").append(img));
    $("tbody").append(tr);
  });
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Loading XML Data</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="js/load_xml.js"></script>
    <link rel="stylesheet" type="text/css" href="css/load_xml.css">
  </head>
  <body>
    <table>
      <caption>Favorite U.S. National Parks</caption>
      <thead><th>Park</th><th>Location</th><th>Established</th><th>&nbsp;</th></
➥thead>
      <tbody></tbody>
    </table>
    <p></p>
  </body>
</html>
```

```
function updateTable(data){
  var parks = $(data).find("park");
  parks.each(function(){
    var tr = $("<tr></tr>");
    tr.append($("<td></td>").html($(this).children("name").text()));
    tr.append($("<td></td>").html($(this).children("location").text()));
    tr.append($("<td></td>").html($(this).children("established").text()));
    var img = $('<img />').attr("src", "images/"+$(this).children("image").
➥text());
    tr.append($("<td></td>").append(img));
    $("tbody").append(tr);
  });
}
$(document).ready(function(){
  $.get("data/parkdata.xml", updateTable);
});
```

```xml
<parkinfo>
  <park>
    <name>Yellowstone</name>
    <location>Montana, Wyoming, Idaho</location>
    <established>March 1, 1872</established>
    <image>bison.jpg</image>
  </park>
  <park>
    <name>Yosemite</name>
    <location>California</location>
    <established>March 1, 1872</established>
    <image>falls.jpg</image>
  </park>
  <park>
    <name>Zion</name>
    <location>Utah</location>
    <established>November 19, 1919</established>
    <image>peak.jpg</image>
  </park>
</parkinfo>
```

```
img {width:80px;}
th {
  background-image: -moz-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -webkit-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -ms-linear-gradient(top , #0000FF, #88BBFF);
  color:white;  font:bold 18px arial black; }
caption { border-radius: 10px 10px 0px 0px; font-size:22px; height:30px; }
td { border:1px dotted; padding:2px; }
```

```
$(document).ready(function(){
  $.get("/getList", setList).done(function(){
     $("span:first").click(); return false; });
  $(".star").click(sendRating);
});
```

```
function setList(data){
  var items = [];
  $.each(data, function(key, val) {
    var item = $("<span></span>").html(val.title);
    item.click(function(){getTrip(val.idx)});
    $("#leftNav").append(item);
  });
}
```

```
function getTrip(idx){
  $.get("/getTrip", {idx: idx}, setTrip);
}
```

```
function sendRating(){
  var idx = $("#idx").html();
  var rating = $(".star").index($(this))+1;
  $.post("/setRating", {idx: idx, rating: rating}, setTrip);
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>AJAX Post</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="js/ajax_post.js"></script>
    <link rel="stylesheet" type="text/css" href="css/ajax_post.css">
  </head>
  <body>
    <div><div id="banner">Vacations</div>
    <div>
      <div id="leftNav"></div>
      <div id="content">
        <span id="idx" class="hidden"></span>
        <p id="title">Title</p>
        <img id="photo" src="/images/falls.jpg" width="200px"/>
        <p id="date">date</p>
        <p id="info">
        <label>5</label> days of fun in <label>Location</label></p>
        <img class="star" src="/images/star.png" />
        <img class="star" src="/images/star.png" />
        <img class="star" src="/images/star.png" />
        <img class="star" src="/images/star.png" />
        <img class="star" src="/images/star.png" />
      </div>
    </div>
  </body>
</html>
```

```
function setTrip(data){
  $("#idx").html(data.idx);
  $("#title").html(data.title);
  $("#photo").attr("src", "/images/"+data.image);
  $("#date").html(data.date);
  $("label:first").html(data.days);
  $("label:last").html(data.location);
  $(".star:gt("+(parseInt(data.rating)-1)+")").attr("src", "/images/empty.png");
  $(".star:lt("+(parseInt(data.rating))+")").attr("src", "/images/star.png");
}
function getTrip(idx){
  $.get("/getTrip", {idx: idx}, setTrip);
}
function setList(data){
  var items = [];
  $.each(data, function(key, val) {
    var item = $("<span></span>").html(val.title);
    item.click(function(){getTrip(val.idx)});
    $("#leftNav").append(item);
  });
}
function sendRating(){
  var idx = $("#idx").html();
  var rating = $(".star").index($(this))+1;
  $.post("/setRating", {idx: idx, rating: rating}, setTrip);
}
$(document).ready(function(){
  $.get("/getList", setList).done(function(){
     $("span:first").click(); return false; });
  $(".star").click(sendRating);
});
```

```
* { font-family:Georgia; }
div { margin:0px; display:inline-block;
  float:left; text-align:center; }
span { background-image: -moz-linear-gradient(top , #f1f1f1, #8F8F8F);
  background-image: -webkit-linear-gradient(top , #f1f1f1, #8F8F8F);
  background-image: -ms-linear-gradient(top , #f1f1f1, #8F8F8F);
  color:black; font-size:20px; float:left; cursor:pointer;
  width:150px; text-align:center; border-bottom:3px ridge; }
p { margin:0px; }
#banner {
  background-image: -moz-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -webkit-linear-gradient(top , #0000FF, #88BBFF);
  background-image: -ms-linear-gradient(top , #0000FF, #88BBFF);
  color:white; height:80px; width:550px; font-size:60px;
  border:3px ridge blue; }
#title { font-weight:bold; font-size:32px; }
#leftNav {width:150px; height:400px; font-size:20px;
border-right:3px ridge;}
#content { height:400px; width: 400px; }
#photo { margin:20px; border:5px ridge white;
  box-shadow: 10px 10px 5px #888888; border-radius:30px;}
#date { color:red; font-style:italic; font-size:24px; }
#info, label { font-size:24px; }
.star {width:30px;}
#idx { display: none; }
```

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use('/', express.static('./'));
var trips = [
 {
   "idx": 0,
   "title": "Lost in Paradise",
   "location": "Hawaii",
   "date": "November 15th",
   "days": "7",
   "image": "flower.jpg",
   "rating": "4"
 },
 {
   "idx": 1,
   "title": "Breathtaking Beauty",
   "location": "Yosemite",
   "date": "June 25th",
   "days": "4",
   "rating": "4",
   "image": "falls.jpg"
 },
 {
   "idx": 2,
   "title": "Wild Expanse",
   "location": "Yellowstone",
   "date": "August 11th",
   "days": "6",
   "rating": "2",
   "image": "bison.jpg"
 },
 {
   "idx": 3,
   "title": "Awe Inspiring",
   "location": "Zion",
   "date": "September 16th",
   "days": "4",
   "rating": "4",
   "image": "peak.jpg"
 }
];
app.get('/getList', function(req, res){
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(trips));
});
app.get('/getTrip', function(req, res){
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(trips[req.query.idx]));
});
app.post('/setRating', function(req, res){
  var test = 1;
  trips[req.body.idx].rating = req.body.rating;
  res.setHeader('Content-Type', 'application/json');
  res.end(JSON.stringify(trips[req.body.idx]));
});
app.listen(80);
```

```
$.ajaxSetup({url:"service.php", accepts:"json"})
```

```
$(document).ajaxStart(function(){
  $("form").addClass("processing");
});
$(document).ajaxComplete(function(){
  $("form").removeClass("processing");
});
```

```
$.ajax({
  url:"setEmail",
  type:"get",
  accepts:"json",
  contentType: 'application/x-www-form-urlencoded; charset=UTF-8',
  data: {"first":"Brad", "last":"Dayley"}
}).fail(function(){ alert("request Failed"); });
```

```
<script type="text/javascript" src="https://code.jquery.com/jquery-2.1.3.min.js"></
➥script>
<script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-ui.min.
➥js"></script>
<link rel="stylesheet" type="text/css" href="http://code.jquery.com/ui/1.11.2/
➥themes/smoothness/jquery-ui.css">
```

```
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/
➥jquery-ui.min.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
```

```
    <div id="datepicker"></div>
```

```
$(document).ready(function(){
  $( "#datepicker" ).datepicker();
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Calendar</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/
➥jquery-ui.min.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <script>
      $(document).ready(function(){
        $( "#datepicker" ).datepicker();
      });
    </script>
  </head>
  <body>
    <div id="datepicker"></div>
  </body>
</html>
```

```
var divs = $("div").uniqueId();
...
do something
...
$("div"). removeUniqueId ();
```

```
$("#timedInput").focus(500, function(){
  $(this).val("Enter Text Now");
});
```

```
$(p).data("color", "red");
$(span).data("color", "blue");
$(div).data("color", "green");
$(":data(color)").each(function(){
  $(this).css({color:$(this).data("color")});
});
```

```
$("form:focusable").each(function(){
  $(this).css({color:red});
});
```

```
$("form:tabbable").each(function(){
  $(this).css({color:red});
});
```

```
    <script type="text/javascript" src="https://code.jquery.com/
➥jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/
➥jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/jquery_image_adder.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/
jquery_image_adder.css">
```

```
$("#arch").data("image", "images/arch.jpg");
$("#volcano").data("image", "images/volcano.jpg");
$("#pyramid").data("image", "images/pyramid2.jpg");
```

```
$("#add").click(function(){
  $("div:data(image)").each(function(){
    $(this).prepend(
        $('<img></img>').attr("src", $(this).data("image")));
  });
  $("div:not(:data(image))").each(function(){
    $(this).prepend(
        $('<img></img>').attr("src", "/images/insert.png"));
  });
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Adding Images</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/
➥jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/
➥jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/jquery_image_adder.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/jquery_image_adder.css">
  </head>
  <body>
    <span id="add">Add Images</span>
    <div id="arch">Arch</div>
    <div id="river">River</div>
    <div id="volcano">Volcano</div>
    <div id="mountain">Mountain</div>
    <div id="pyramid">Pyramid</div>
  </body>
</html>
```

```
$(document).ready(function(){
  $("#arch").data("image", "/images/arch.jpg");
  $("#volcano").data("image", "/images/volcano.jpg");
  $("#pyramid").data("image", "/images/pyramid2.jpg");
  $("#add").click(function(){
    $("div:data(image)").each(function(){
      $(this).prepend(
          $('<img></img>').attr("src", $(this).data("image")));
    });
    $("div:not(:data(image))").each(function(){
      $(this).prepend(
          $('<img></img>').attr("src", "/images/insert.png"));
    });
  });
});
```

```
img {
    width:60px; margin-right:20px; vertical-align:middle; }
div {
    margin-top:15px; border:1px dotted;
    width:400px; font-size:35px;}
span {
    background-image: -moz-linear-gradient(top , #B1B1B1, #FFFFFF);
  background-image: -webkit-linear-gradient(top , #B1B1B1, #FFFFFF);
  background-image: -ms-linear-gradient(top , #B1B1B1, #FFFFFF);
  border:3px ridge white; box-shadow: 5px 5px 5px #888888;
  padding:3px; cursor:pointer; }
```

```
$("#div1").position("my:"left", at:"right", of:"#div2");
```

```
$("#img2").position(
    {my:"left top", at:"right bottom", of:"#img1"});
$("#img3").position(
    {my:"left top", at:"right bottom", of:"#img2"});
```

```
$("div").mousemove(function(e) {
  $("#img4").position({ my:"left top", at:"center", of:e,
                        collision:"flip", within:"div" });
})
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Dynamic Positioning</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-
➥ui.min.js"></script>
    <script type="text/javascript" src="js/dynamic_positioning.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/dynamic_positioning.css">
  </head>
  <body>
    <div>
      <img id="img1" src="/images/arch.jpg">
      <img id="img2" src="/images/pyramid2.jpg">
      <img id="img3" src="/images/volcano.jpg">
      <img id="img4" src="/images/jump.jpg">
    </div>
  </body>
</html>
```

```
$(window).load(function(){
  $("#img2").position(
      {my:"left top", at:"right bottom", of:"#img1"});
  $("#img3").position(
      {my:"left top", at:"right bottom", of:"#img2"});
  $("div").mousemove(function(e) {
    $("#img4").position({ my:"left top", at:"center", of:e,
                          collision:"flip", within:"div" });
  })
});
```

```css
img {
    position: absolute; height: 130px; width:auto; }
div {
    height:500px; width:500px; border:3px ridge; }
```

```
.effect( effect [, options ] [, duration ] [, complete ] )
```

```
("img").effect("size",
               {to:{height:100, width:100}, origin:["right","top"], scale:"box"},
               3000,
               function(){alert("effect complete");});
```

```
$("#img1").click(function(e) {
  $(this).effect("shake",
        { direction:"down", distance:20, times:5}, 3000);
});
```

```
$("#img2").click(function(e) {
  $(this).effect("scale",
    { direction:"both", origin:["middle", "right"],
      percent:40, scale:"box", easing:"easeInBounce"}, 3000);
});
```

```
$("#img3").click(function(e) {
  $(this).effect("slide",
    { direction:"down", distance:200}, 3000, function(){
        $(this).effect("slide",
            {direction:"right", distance:200}, 3000);
  });
});
```

```
$("#img4").click(function(e) {
  $(this).effect("explode", {pieces:16}, 3000);
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQeury Effects Showcase</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-
➥ui.min.js"></script>
    <script type="text/javascript" src="js/jquery_effects.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/jquery_effects.css">
  </head>
  <body>
      <div id="frame1">
        <img id="img1" src="/images/double.jpg"></div>
      <div id="frame2">
        <img id="img2" src="/images/arch.jpg"></div>
      <div id="frame3">
        <img id="img3" src="/images/cliff.jpg"></div>
      <div id="frame4">
        <img id="img4" src="/images/sunstar.jpg"></div>
  </body>
</html>
```

```
$(document).ready(function(){
  $("#img1").click(function(e) {
    $(this).effect("shake",
        { direction:"down", distance:20, times:5}, 3000);
  });
  $("#img2").click(function(e) {
    $(this).effect("scale",
      { direction:"both", origin:["middle", "right"],
        percent:40, scale:"box", easing:"easeInBounce"}, 3000);
  });
  $("#img3").click(function(e) {
    $(this).effect("slide",
      { direction:"down", distance:200}, 3000, function(){
          $(this).effect("slide",
              {direction:"right", distance:200}, 3000);
    });
  });
  $("#img4").click(function(e) {
    $(this).effect("explode", {pieces:16}, 3000);
  });
});
```

```
img {
    height:200px; }
div {
    height:200px; width:200px; border:1px dotted;
  display:inline-block; position:fixed; }
#frame1 {
    top:80px; left:20px; }
#frame2 {
    top:80px; left:240px; }
#frame3 {
    top:80px; left:460px; }
#frame4 {
    top:80px; left:720px; }
```

```
$(document).ready(function(){
  $("#btn1").click(function(e) {
    $(this).addClass( "round", 2000, "easeInElastic"); });
  $("#btn2").click(function(e) {
    $(this).switchClass( "active", "inactive", 2000,
                         "easeInOutElastic"); });
  $("#btn3").click(function(e) {
    $(this).toggleClass( "round", 2000, "easeOutQuart"); });
  $("#btn4").click(function(e) {
    $(this).removeClass( "round", 2000, "easeInCirc"); });
});
```

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Class Transitions</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-
➥ui.min.js"></script>
    <script type="text/javascript" src="js/class_transitions.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/class_transitions.css">
  </head>
  <body>
    <span id="btn1">Add Class</span><br>
    <span id="btn2" class="active">Switch Class</span><br>
    <span id="btn3" class="square">Toggle Class</span><br>
    <span id="btn4" class="round">Remove Class</span><br>
  </body>
</html>
```

```
$(document).ready(function(){
  $("#btn1").click(function(e) {
    $(this).addClass( "round", 2000, "easeInElastic"); });
  $("#btn2").click(function(e) {
    $(this).switchClass( "active", "inactive", 2000,
                         "easeInOutElastic"); });
  $("#btn3").click(function(e) {
    $(this).toggleClass( "round", 2000, "easeOutQuart"); });
  $("#btn4").click(function(e) {
    $(this).removeClass( "round", 2000, "easeInCirc"); });
});
```

```
span {
    display:inline-block; height:30px; width:200px;
    margin-top:20px; border:1px ridge;
  text-align:center; font:bold 20px/30px arial; }
.round {
    border-width:3px; border-radius:60px 30px;
    height:60px; width:320px; line-height:60px;
  background-color:steelblue; color:white; font-size:40px; }
.square {
    border-width:6px; background-color:gold; color:black; }
.active {
    background-image: -moz-linear-gradient(top , lightblue, gainsboro);
  background-image: -webkit-linear-gradient(top , lightblue, gainsboro);
  background-image: -ms-linear-gradient(top , lightblue, gainsboro);
  border-radius:4px 4px; border:3px outset slategrey; }
.inactive {
    background-image: -moz-linear-gradient(top , darkgrey, white);
  background-image: -webkit-linear-gradient(top , darkgrey, white);
  background-image: -ms-linear-gradient(top , darkgrey, white);
  border-radius:4px; color:#steelblue; }
```

```
$(document).ready(function(){
  $("#showMenu, #showMenu2, #toggleMenu").hide();
...
})
```

```
$("#show").click(function(e) {
    $("#showMenu").show("fold", {size:22}, 2000); });
$("#show2").click(function(e) {
    $("#showMenu2").show("scale",
        {origin:["top","left"]}, 2000); });
$("#showMenu").click(function(e) {
    $("#showMenu").hide("fold", {size:22}, 2000); });
$("#showMenu2").click(function(e) {
    $("#showMenu2").hide("explode", {pieces:9}, 2000); });
$("#toggle, #toggleMenu").click(function(e) {
    $("#toggleMenu").toggle("blind",
        {direction:"up", easing:"easeOutBounce"}, 1000); });
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Visibility Transitions</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-
➥ui.min.js"></script>
    <script type="text/javascript" src="js/visibility_transitions.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/visibility_transitions.
css">
  </head>
  <body>
    <span id="show">Show Fold</span><br>
    <span id="show2">Show Scale</span><br>
    <span id="toggle">Toggle Blind</span><br>
    <div id="showMenu">
      <span>Fold 1</span><br><span>Fold 2</span><br>
      <span>Fold 3</span><br><span>Fold 4</span><br>
    </div>
    <div id="showMenu2">
      <span>Explode 1</span><br><span>Explode 2</span><br>
      <span>Explode 3</span><br><span>Explode 4</span><br>
    </div>
    <div id="toggleMenu">
      <span>Toggle 1</span><br><span>Toggle 2</span><br>
      <span>Toggle 3</span><br><span>Toggle 4</span><br>
    </div>
  </body>
</html>
```

```
$(document).ready(function(){
  $("#showMenu, #showMenu2, #toggleMenu").hide();
  $("#show").click(function(e) {
      $("#showMenu").show("fold", {size:22}, 2000); });
  $("#show2").click(function(e) {
      $("#showMenu2").show("scale",
          {origin:["top","left"]}, 2000); });
  $("#showMenu").click(function(e) {
      $("#showMenu").hide("fold", {size:22}, 2000); });
  $("#showMenu2").click(function(e) {
      $("#showMenu2").hide("explode", {pieces:9}, 2000); });
  $("#toggle, #toggleMenu").click(function(e) {
      $("#toggleMenu").toggle("blind",
          {direction:"up", easing:"easeOutBounce"}, 1000); });
});
```

```
span {
  display:inline-block; width:130px; border:1px ridge;
  text-align:center; cursor:pointer;
  background-image: -moz-linear-gradient(top , lightblue, white);
  background-image: -webkit-linear-gradient(top , lightgray, white);
  background-image: -ms-linear-gradient(top , lightgray, white); }
div span{
  width:120px; margin-left:10px; }
#showMenu {
  position:fixed; left:130px; top:8px; }
#showMenu2 {
  position:fixed; left:130px; top:30px; }
```

```
$(document).ready(function(){
  $("#ball").click(reposition);
});
```

```
function reposition(){
  if (coords.length){
    coord = coords.pop();
    $(this).animate(coord, 1000, coord.easing, reposition);
  } else{
    $("#ball").effect("explode", {pieces:100}, 2000); }
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Adding Effects to Animations</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/jquery-
➥2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-
➥ui.min.js"></script>
    <script type="text/javascript" src="js/animation_effects.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/animation_effects.css">
  </head>
  <body>
      <img id="needle" src="/images/needle.jpg" />
    <img id="ball" src="/images/ball.png" />
  </body>
</html>
```

```
var coords = [{top:140, left:470, easing:"easeInBounce"},
              {top:100, left:200, easing:"easeOutElastic"},
              {top:300, left:200, easing:"easeInOutCirc"},
              {top:20, left:300, easing:"easeInBounce"},
              {top:10, left:10, easing:"easeOutExpo"},
              {top:200, left:100, easing:"easeInSine"}]
function reposition(){
  if (coords.length){
    coord = coords.pop();
    $(this).animate(coord, 1000, coord.easing, reposition);
  } else{
    $("#ball").effect("explode", {pieces:100}, 2000); }
}
$(document).ready(function(){
  $("#ball").click(reposition);
});
```

```
img {
  position:fixed; z-index: -1;}
#needle {
  left:450px; top:50px; width: 150px; }
#ball {
  left:50px; top:50px; width: 100px; }
```

```
$.easing.myCustom = function(tPercent, tMS, startValue, endValue, tTotal)
➥{
     var newValue= <your code here>>...
    return newValue;
}
```

```
{top:20, left:300, easing:"easeInBounce", duration:1500},
$( "#item1" ).mouse( "option", "cancel", ".label" );
$( "#item2" ).mouse( "option", "delay", 1000 );
$( "#item3" ).mouse( "option", "distance", 10 );
```

```
$("#img1").draggable({cursor:"move", opacity:.5});
```

```
$("#drag1").draggable({cursor:"move", opacity:.5});
$("#drag1").on("dragstop", function(){$(this).effect("bounce", 1000); });
```

```
$("#drag1").draggable({cursor:"move", opacity:.5});
$("#drag1").on("dragstop", function(){
  $(this).effect("bounce", 1000); });
```

```
$("#drag2").draggable({helper:"clone"});
$("#drag2").on("dragstop", function(e, ui){
  $("#drag2").animate(ui.offset); });
```

```
$("#drag3").draggable();
$("#drag3").on("drag", function(e){
  $(this).children("p").html(e.pageX+", "+e.pageY); });
$("#drag3").on("dragstop", function(e){
  $(this).children("p").html(""); });
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Making Images Drag-able</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/
➥jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/
➥jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/draggable_images.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/draggable_images.css">
  </head>
  <body>
      <div id="drag1">
      <img id="img1" src="/images/snowy_cliff.jpg"></div>
      <div id="drag2">
      <img id="img1" src="/images/shadow_jump.jpg"></div>
      <div id="drag3">
      <img id="img1" src="/images/sunstar.jpg"></div>
  </body>
</html>
```

```
$(window).load(function(){
  $("#drag1").draggable({cursor:"move", opacity:.5});
  $("#drag1").on("dragstop", function(){
    $(this).effect("bounce", 1000); });
  $("#drag2").draggable({helper:"clone"});
  $("#drag2").on("dragstop", function(e, ui){
    $("#drag2").animate(ui.offset); });
  $("#drag3").draggable();
  $("#drag3").on("drag", function(e){
    $(this).children("p").html(e.pageX+", "+e.pageY); });
  $("#drag3").on("dragstop", function(e){
    $(this).children("p").html(""); });
});
```

```
p {
    margin:0px; }
div {
    height:80px; width:100px; position:fixed; }
#drag2 {
    top:100px; }
#drag3 {
    top:200px; }
img {
    width:200px; }
img:hover{
    cursor:move; }
```

```
$("#div1"). droppable ({tolerance:"touch"});
```

```
$("#drop1").droppable({tollerance:"pointer"});
$("# drop1").on("dropactivate", function(){$(this).effect("shake", 1000); });
```

```
$("#drag1, #drag2, #drag3").draggable(
  {helper:"clone", cursor:"move", opacity:.7, zIndex:99});
```

```
$("#drop1").droppable(
    { accept:"img", tolerance:"fit"});
$("#drop1").on("dropover", function(e,ui){
  $(this).effect("pulsate"); });
$("#drop1").on("drop", function(e,ui){
  $(this).html($("<img></img>").attr("src",
      ui.draggable.attr("src")));
  $(this).effect("bounce");
});
```

```
$("#drop2").droppable(
    { accept:"img", tolerance:"intersect",
      hoverClass:"drop-hover"});
$("#drop2").on("drop", function(e,ui){
  var item = $("<div></div>");
  item.append($("<img></img>").attr("src",
      ui.draggable.attr("src")));
  item.append($("<span></span>").html(
      ui.draggable.attr("src")));
  $(this).append(item);
});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>drag'n'droppin</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/
➥jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/jquery-
➥ui.min.js"></script>
    <script type="text/javascript" src="js/drag_n_drop.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/drag_n_drop.css">
  </head>
  <body>
    <div id="images">
      <img id="drag1" src="/images/jungle.jpg" />
      <img id="drag2" src="/images/wheel.jpg" />
      <img id="drag3" src="/images/img7.jpg" />
    </div>
    <div id="drop1"></div>
    <div id="drop2"></div>
  </body>
</html>
```

```
$(document).ready(function(){
  $("#drag1, #drag2, #drag3").draggable(
    { helper:"clone", cursor:"move", opacity:.7, zIndex:99 });
  $("#drop1").droppable(
      { accept:"img", tolerance:"fit"});
  $("#drop1").on("dropover", function(e,ui){
    $(this).effect("pulsate"); });
  $("#drop1").on("drop", function(e,ui){
    $(this).html($("<img></img>").attr("src",
        ui.draggable.attr("src")));
    $(this).effect("bounce");
  });
  $("#drop2").droppable(
      { accept:"img", tolerance:"intersect",
        hoverClass:"drop-hover"});
  $("#drop2").on("drop", function(e,ui){
    var item = $("<div></div>");
    item.append($("<img></img>").attr("src",
        ui.draggable.attr("src")));
    item.append($("<span></span>").html(
        ui.draggable.attr("src")));
    $(this).append(item);
  });
});
```

```
div {
    display:inline-block; vertical-align:top; }
img {
    width:100px; margin:0px; }
#images {
    width:100px; height:300px; }
#drop1, #drop2 {
    width:300px; min-height:150px; padding:3px; margin:10px;
  border:3px ridge white; box-shadow: 5px 5px 5px #888888; }
#drop1 img {
    width:300px; }
#drop2 div{
    height:80px; width:280px; padding:4px;
    border:3px ridge darkblue; margin-top:5px; }
#drop2 div img {
    height:80px; margin-right:10px; }
#drop2 div span {
    display:inline-block; vertical-align:top;
    font:16px/70px arial; }
.drop-hover {
    background-color:#BBDDFF; }
```

```
$("#div1"). resizable ({aspectRatio:true});
```

```
$("#resize1"). resizable ({aspectRatio:true });
$("#resize1").on("dropactivate", function(){$(this).effect("pulsate"); });
```

```
$("#resize1, #resize2, #resize3").draggable();
```

```
$("#resize1").resizable(
     {aspectRatio:true, alsoResize:"#resize1 img" });
```

```
$("#resize2").resizable(
    {alsoResize:"#resize2 img"});
```

```
for(var i=0; i<100; i++){
  $("#list").append($("<p></p>").html("Item "+i)); }
$("#resize3").resizable(
    {alsoResize:"#resize3 #list", handles:"n,s,e,w"});
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Making Resizable Elements</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/
➥jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/
➥jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/resizable_elements.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/resizable_elements.css">
  </head>
  <body>
    <div id="resize1"><img src="/images/beachhouse.jpg" /></div>
    <div id="resize2">
      <p>Ocean View</p>
      <img src="/images/img7.jpg" />
    </div>
    <div id="resize3"><div id="list"></div></div>
  </body>
</html>
```

```
$(document).ready(function(){
  $("#resize1, #resize2, #resize3").draggable();
  $("#resize1").resizable(
      {aspectRatio:true, alsoResize:"#resize1 img" });
  $("#resize2").resizable(
      {alsoResize:"#resize2 img"});
  for(var i=0; i<100; i++){
    $("#list").append($("<p></p>").html("Item "+i)); }
  $("#resize3").resizable(
      {alsoResize:"#resize3 #list", handles:"n,s,e,w"});
});
```

```css
img {
  width:230px; }
#resize1, #resize2, #resize3 {
  width:250px; padding:10px; display:inline-block;
  margin:10px; vertical-align:top;
  border:3px ridge white; box-shadow: 5px 5px 5px #888888; }
p {
  margin:2px; border:1px dotted; text-align:center; }
#list {
  height:200px; overflow-y:auto; }
```

```
$("#ul1"). selectable ({tolerance:"fit"});
```

```
$("#list1").selectable();
$("#list1").on("selectableselected", function(e, ui){ ui.selected.effect("shake");
});
```

```
for(var i=0; i<100; i++){
  $("#set1").append($("<p></p>").html("Item "+i)); }
```

```
$("#set1").selectable({ filter:"p" });
$("#set1").on("selectablestart", function(e, ui){
  $("span").html("Selecting "); });
$("#set1").on("selectableselecting", function(e, ui){
  $("span").append(ui.selecting.innerHTML+", "); });
$("#set1").on("selectablestop", function(e, ui){
  $("span").html("Selection Complete"); });
```

```
$("#set2").selectable();
$("#set2").on("selectablestop", function(e, ui){
  var selection = $("#set2 .ui-selected");
  selection.effect("highlight");
  $("span").html("Selected "+ selection.length +
                 " Photos"); });
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sorting Elements</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/
➥jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/
➥jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/selectable_sets.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/selectable_sets.css">
  </head>
  <body>
    <span>Nothing Selected</span><br>
    <div id="set1"></div>
    <div id="set2">
      <img src="/images/cliff.jpg" /><img src="/images/flower2.jpg" />
      <img src="/images/lake.jpg" /><img src="/images/tiger.jpg" />
      <img src="/images/flower.jpg" /><img src="/images/volcano.jpg" />
    </div>
  </body>
</html>
```

```
$(document).ready(function(){
  for(var i=0; i<100; i++){
    $("#set1").append($("<p></p>").html("Item "+i)); }
  $("#set1").selectable({ filter:"p" });
  $("#set1").on("selectablestart", function(e, ui){
    $("span").html("Selecting "); });
  $("#set1").on("selectableselecting", function(e, ui){
    $("span").append(ui.selecting.innerHTML+", "); });
  $("#set1").on("selectablestop", function(e, ui){
    $("span").html("Selection Complete"); });
  $("#set2").selectable();
  $("#set2").on("selectablestop", function(e, ui){
    var selection = $("#set2 .ui-selected");
    selection.effect("highlight");
    $("span").html("Selected "+ selection.length +
                   " Photos"); });
});
```

```
span {
  display:inline-block; border:1px solid;
  font:bold 18px/26px arial;
  width:800px; text-align:center; margin:10px;}
div {
  display:inline-block; border:3px ridge white;
  vertical-align:top; margin:10px; }
p {
  border:1px dotted; margin:0px; }
#set1 {
  height:300px; width:200px; overflow-y:auto;
  text-align:center; }
#set1 .ui-selecting {
  background-image: -moz-linear-gradient(top , #88BBFF, #DDEEFF);
  background-image: -webkit-linear-gradient(top , #88BBFF, #DDEEFF);
  background-image: -ms-linear-gradient(top , #88BBFF, #DDEEFF); }
#set1 .ui-selected {
  background-color:blue; color:white; }
img {
  height:90px; border:3px ridge white; margin:15px;
  box-shadow: 5px 5px 5px #888888; opacity:.6; }
#set2 {
  width:500px; padding:25px; border-radius:15px; }
#set2 .ui-selecting{
  border:5px ridge green; box-shadow: 5px 5px 5px #558822; }
#set2 .ui-selected{
  border:5px ridge blue; box-shadow: 5px 5px 5px #225588;
  opacity:1; }
```

```
$("#ul1").sortable({tolerance:"fit"});
```

```
$("#list1").sortable({connectWith:"#list2"});
$("#list1").on("sortreceived", function(e, ui){
  ui.sender.effect("pulsate");
  $(this).effect("pulsate "); });
```

```
$("#sorter1").sortable(
    {cursor:"move", connectWith:"#sorter2"});
```

```
$("#sorter2").sortable(
     {cursor:"move", connectWith:"#sorter1"});
$("#sorter2").on("sortreceive", function(e, ui){
  ui.sender.effect("pulsate");
  $(this).effect("pulsate"); });
```

```
$("#sortTable").sortable(
    {axis:"y", cursor:"n-resize", });
$("#sortTable").on("sortupdate", function(e, ui){
  ui.item.effect("pulsate"); });
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sorting Things Out</title>
    <meta charset="utf-8" />
    <script type="text/javascript" src="https://code.jquery.com/
➥jquery-2.1.3.min.js"></script>
    <script type="text/javascript" src="http://code.jquery.com/ui/1.11.2/
➥jquery-ui.min.js"></script>
    <script type="text/javascript" src="js/sortable_elements.js"></script>
    <link rel="stylesheet" type="text/css" href="http://code.jquery.com/
➥ui/1.11.2/themes/smoothness/jquery-ui.css">
    <link rel="stylesheet" type="text/css" href="css/sortable_elements.css">
  </head>
  <body>
    <div id="sorter1"></div>
    <div id="sorter2"></div>
    <table border=1>
      <tbody>
        <tr>
          <th>Icon</th>
          <th>Number</th>
          <th>Name</th>
          <th>Source</th>
        </tr>
      </tbody>
      <tbody id="sortTable"></tbody>
    </table>
  </body>
</html>
```

```
var images = [
  {src:"/images/lake.jpg",name:"Lake"},
  {src:"/images/cliff.jpg",name:"Cliff"},
  {src:"/images/flower2.jpg",name:"Violet"},
  {src:"/images/tiger.jpg",name:"Tiger"},
  {src:"/images/volcano.jpg",name:"Volcano"},
  {src:"/images/flower.jpg",name:"Flower"}];
function buildLists(){
  $.each(images, function(i,item){
    var img = $("<img></img>").attr("src", item.src);
    var name = $("<p></p>").html(item.name);
    $("#sorter1").append($("<div></div>").append(img, name));
    var tr = $("<tr></tr>");
    tr.append($("<td></td>").append(
        $("<img></img>").attr("src", item.src)));
    tr.append($("<td></td>").html(i));
    tr.append($("<td></td>").html(item.name));
    tr.append($("<td></td>").html(item.src));
    $("#sortTable").append(tr);
  });
}
$(document).ready(function(){
  buildLists();
  $("#sorter1").sortable(
      {cursor:"move", connectWith:"#sorter2"});
  $("#sorter2").sortable(
      {cursor:"move", connectWith:"#sorter1"});
  $("#sorter2").on("sortreceive", function(e, ui){
    ui.sender.effect("pulsate");
    $(this).effect("pulsate"); });
  $("#sortTable").sortable(
      {axis:"y", cursor:"n-resize", });
  $("#sortTable").on("sortupdate", function(e, ui){
    ui.item.effect("pulsate"); });
});
```

```
#sorter1, #sorter2, table {
  display:inline-block; cursor:move;
 width:200px; padding:10px; vertical-align:top;
 margin:15px; height:auto; box-shadow: 5px 5px 5px #888888;
 border:3px ridge white;  }
#sorter1 div, #sorter2 div {
  width:180px; display:inline-block; height:50px;
 padding:5px; margin:5px; border:1px dotted;
 vertical-align:middle; }
p {
  float:right; margin:0px; display:inline-block; height:50px;
 font:bold 18px/50px arial; vertical-align:top;
 text-align:center; }
img {
  height:50px; }
table {
  width:auto; padding:5px; }
tr {
  background-color:white; }
td {
  min-width:80px; }
#sortTable img {
  height:20px; }
.ui-sortable-helper {
  background-color:blue; color:white; opacity:.5; }
table:hover{
  cursor:move}
```

```
$("#myList").mouse({ cancel:".notSelectable"});
```

```
$("#mySlider").slider({min:2, max:10});
```

```
var slider = $("#mySlider").slider({min:2, max:10});
slider.slider("option", "max", 100);
var currentMax = slider.slider("option", "max");
var value = $("#mySlider").slider("value");
$("#mySlider").slider("value", value+5 );
```

```
...jQuery...
$(window).load(function(){
  $( "#accordion" ).accordion(
      {header:"p", collapsable:true});
});
...CSS...
div { width:300px; }
img { width:300px; }
...HTML...
<div id="accordion">
  <p>Fort</p><div>
  <img src="/images/fort.jpg" /></div>
  <p>Bison</p><div>
  <img src="/images/bison.jpg" /></div>
  <p>Sunset</p><div>
  <img src="/images/sunset3.jpg" /></div>
  <p>tiger</p><div>
  <img src="/images/tiger.jpg" /></div>
</div>
```

```
...jQuery...
function dateChanged(dateStr, Object){
  $("span").html("Updated to" + dateStr); }
$(document).ready(function(){
  $( "#autocomplete" ).autocomplete({
    source: ["Monday", "Tuesday", "Wednesday",
             "Thursday", "Friday"]
  });
});
...CSS...
input {
  border:2px ridge steelblue; border-radius:5px;
  padding:3px; }
...HTML...
<label for="autocomplete">Day of Week: </label>
<input id="autocomplete">
```

```
...jQuery...
  $(window).ready(function(){
      $( "#button1" ).button(
          {icons: {primary: "ui-icon-gear"},text: true});
      $( "#check" ).button();
      $( "#format" ).buttonset();
      $( "#radio" ).buttonset();
    });
...CSS...
    div { margin:15px; }
...HTML...
    <div>
      <button id="button1">Configure</button>
    </div>
    <div>
      <input type="checkbox" id="check" />
      <label for="check">Toggle</label>
    </div>
    <div id="format">
      <input type="checkbox" id="check1" />
      <label for="check1">B</label>
      <input type="checkbox" id="check2" />
      <label for="check2">I</label>
      <input type="checkbox" id="check3" />
      <label for="check3">U</label>
    </div>
    <div id="radio">
      <input type="radio" id="radio1" name="radio" />
      <label for="radio1">option 1</label>
      <input type="radio" id="radio2" name="radio" />
      <label for="radio2">option 2</label>
      <input type="radio" id="radio3" name="radio" />
      <label for="radio3">option 3</label>
    </div>
```

```
...jQuery...
function dateChanged(dateStr, Object){
  $("span").html("Updated to" + dateStr); }
$(document).ready(function(){
  $( "#month" ).datepicker({
      onSelect:dateChanged,
      showOn: "button",
      buttonImage: "/images/calendar32.png",
      buttonImageOnly: true,
      numberOfMonths:2,
      showButtonPanel:true,
      dateFormat: "yy-mm-dd"
    });
});
...CSS...
input { border:2px ridge steelblue; border-radius:3px; }
...HTML...
<label>Start Date: </label>
<input type="text" id="month"></input>
<span></span>
```

```
...jQuery...
$(document).ready(function(){
   $( "#dialog" ).dialog({ modal: true,
     buttons: { Sweet: function() {
       $( this ).dialog( "close" ); }}});
});
...CSS...
img { height:60px; float:left; }
...HTML...
<div id="dialog" title="Upload Successful">
  <p>
    <img src="/images/sunset2.jpg" />
    <span class="ui-icon ui-icon-circle-check"></span>
    Image Uploaded Successfully.
  </p>
  <p>You are currently using <br>
    <b>32% of your storage space</b>.</p>
</div>
```

```
...jQuery...
$(document).ready(function(){
 $( "#menu" ).menu();
 $( "#menu" ).on("menuselect", function(e, ui){
  $("p").html("Selected " +
    ui.item.children("a:first").html());
 });
});
...CSS...
.ui-menu { width: 200px; }
p { box-shadow: 5px 5px 5px #888888;
  border:3px ridge steelblue; color:teal;
  display:inline-block; height:80px; width:100px; }
...HTML...
<ul id="menu">
 <li><a href="#">Open</a></li>
 <li><a href="#">Recent</a><ul>
  <li><a href="#">Some File</a></li>
  <li><a href="#">Another File</a></li>
  </ul></li>
 <li><a href="#">Save</a></li>
 <li class="ui-state-disabled">
  <a href="#"><span class="ui-icon ui-icon-print">
  </span>Print...</a></li>
 <li><a href="#">Slide Show</a><ul>
  <li>
   <a href="#">
    <span class="ui-icon ui-icon-seek-start"></span>Prev</a>
  </li>
  <li>
   <a href="#">
    <span class="ui-icon ui-icon-stop"></span>Stop</a>
  </li>
  <li>
   <a href="#">
    <span class="ui-icon ui-icon-play"></span>Play</a>
  </li>
  <li>
   <a href="#">
    <span class="ui-icon ui-icon-seek-end"></span>Next</a>
  </li>
  </ul></li>
</ul>
<p></p>
```

```
...jQuery...
function inc(){
  var value = $("#progressbar").progressbar("value") + 5;
  if (value <= 100){
    $("p").html("Progress: " + value + "%");
    $("#progressbar").progressbar("value", value);
    setTimeout(inc, 100);
   }
}
$(document).ready(function(){
  $("#progressbar").progressbar({ value: 0});
  inc();
});
...CSS...
#progressbar {
  box-shadow: 5px 5px 5px #888888; border:2px ridge;
  display:inline-block; height:20px; width:300px; }
#progressbar .ui-progressbar-value{
  background-image: -moz-linear-gradient(top, steelblue, skyblue);
  background-image: -webkit-linear-gradient(top, steelblue, skyblue);
  background-image: -ms-linear-gradient(top , steelblue, skyblue); }
...HTML...
<p></p>
<div id="progressbar"></div>
```

```
...jQuery...
function cValue(selector){
  var v = $(selector).slider("value").toString( 16 );
  if (v.length ===1) { v = "0" + v;}
  return v;
}
function refreshSwatch() {
  $("#mix").css("background-color", "#" + cValue("#red") +
    cValue("#green") +  cValue("#blue"));
  $("#mix").html($("#mix").css("background-color"));
}
$(document).ready(function(){
   $( "#red, #green, #blue" ).slider({
    orientation: "horizontal",
    range: "min",
    max: 255,
    value: 127,
    slide: refreshSwatch,
    change: refreshSwatch
  });
  $("#red").slider("value", 128);
  $("#green").slider("value", 128);
  $("#blue").slider("value", 128);
});
...CSS...
#mix {
  width:160px; height:100px; text-align:center;
  font:18px/100px arial; }
#red, #green, #blue {
  float: left; clear: left; width: 150px; margin: 15px; }
#red .ui-slider-range { background:red; }
#red .ui-slider-handle { border-color:red; }
#green .ui-slider-range { background:green; }
#green .ui-slider-handle { border-color:green; }
#blue .ui-slider-range { background:blue; }
#blue .ui-slider-handle { border-color:blue; }
...HTML...
<div id="mix"></div>
<div id="red"></div>
<div id="green"></div>
<div id="blue"></div>
```

```
...jQuery...
$(document).ready(function(){
  $( "#spin1" ).spinner({step: 0.5});
  $( "#spin2" ).spinner({step: 1});
  $( "#spin3" ).spinner({step: 10});
});
...CSS...
label { display:inline-block; width:100px; }
...HTML...
<p>
  <label for="spin1">Count by .5s:</label>
  <input id="spin1" name="value" /><br>
  <label for="spin2">Count by 1s:</label>
  <input id="spin2" name="value" /><br>
  <label for="spin3">Count by 10s:</label>
  <input id="spin3" name="value" /><br>
</p>
```

```
...jQuery...
  $(document).ready(function(){
    $( "#tabs" ).tabs(
        { event: "mouseover", collapsible: true,
          active:"false"});
  });
...CSS...
  * { vertical-align:top; }
  img { height:120px; margin:5px; }
  .mini {height:23px; margin:0px; }
  #tabs { width:450px; }
  p { text-align:justify; }
...HTML...
<div id="tabs">
  <ul>
    <li>
      <a href="#tabs1">
        <img class="mini" src="/images/arch.jpg" />Images</a>
    </li>
    <li><a href="#tabs2">Content</a></li>
    <li><a href="#tabs3">Widgets</a></li>
  </ul>
  <div id="tabs1">
    <img src="/images/jail.jpg" />
    <img src="/images/sunset.jpg" />
    <img src="/images/cliff.jpg" />
    <img src="/images/sunset2.jpg" />
  </div>
```

```
    <div id="tabs2">
      <h3>jQuery UI Widgets</h3>
      <p><b><i>jQuery UI</i></b>
        provides a wide array of pre-built widgets that
        provide extended functionality to HTML elements.
        These widgets provide functionality that make forms
        and other input controls more intuitive
        and easy to use. For example a calendar
        view for choosing dates and expandable menus.
      </p>
    </div>
    <div id="tabs3">
      <ul>
        <li>Accordion</li>
        <li>Menu</li>
        <li>Button</li>
        <li>Slider</li>
        <li>Progress Bar</li>
        <li>Tabs</li>
      </ul>
    </div>
  </div>
```

```
position: {my: "left top+15", at: "left bottom", collision: "flipfit" }
```

```
...jQuery...
$(document).ready(function(){
  $(document).tooltip({
    items: "img, input",
    position: { my: "left+15 top", at: "left bottom",
               collision: "flipfit" },
    content: function() {
      var obj = $(this);
      if (obj.is("input")) { return obj.attr( "title" ); }
      if (obj.is("img")) {
        var img = $("<img></img>").addClass("mini")
                    .attr("src", obj.attr("src"));
        var span = $("<span></span>")
                     .html(obj.attr( "alt" ));
        return $("<div></div>").append(img, span); }
    }});
});
...CSS...
input {
  border:2px ridge blue; border-radius:5px; padding:3px; }
img { height:200px; margin:15px; }
.mini {height:30px; }
...HTML...
<label for="size">Who are You?</label>
<input id="size" title="Nosce Te Ipsum (Know Thyself)" /><br>
<img src="/images/someday.jpg" alt="I'm going there someday"/>
```

```
$.widget("custom.mywidget", {
  options : {
    // custom options
  },
  _create: function() {
    //creation code
  },
  _refresh: function() {
    //refresh code called when element refreshed
  },
  _destroy: function() {
    //cleanup code called when widget is destroyed
  },
  _setOptions: function() {
    // _super and _superApply handle keeping the right this-context
    this._superApply( arguments );
    this._refresh();
  },
  _setOption: function( key, value ) {
    // set individual option value override code
    this._super( key, value );
  }
});
```

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.5/angular.min.js">
➥</script>
```

```
<!doctype html>
<html ng-app="myApp">
  <body>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="/lib/myApp.js"></script>
  </body>
</html>
```

```
var myCopy = angular.copy(myObj);
var objArr = [{score: 95}, {score: 98}, {score: 92}];
var scores = [];
angular.forEach(objArr, function(value, key){
  this.push(key + '=' + value);
}, scores);
// scores == ['score=95', 'score=98', 'score=92']
```

```
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="/js/first.js"></script>
```

```
var firstApp = angular.module('firstApp', []);
```

```
<div ng-controller="FirstController">
```

```
firstApp.controller('FirstController', function($scope) {
});
```

```
        <input type="text" ng-model="first">
        <input type="text" ng-model="last">
```

```
$scope.first = 'Some';
$scope.last = 'One';
$scope.heading = 'Message: ';
```

```
      <button ng-click='updateMessage()'>Message</button>
```

```
$scope.updateMessage = function() {
  $scope.message = 'Hello ' + $scope.first +' '+ $scope.last + '!';
};
```

```html
<!doctype html>
<html ng-app="firstApp">
  <head>
    <title>First AngularJS App</title>
  </head>
  <body>
    <div ng-controller="FirstController">
      <span>Name:</span>
      <input type="text" ng-model="first">
      <input type="text" ng-model="last">
      <button ng-click='updateMessage()'>Message</button>
      <hr>
      {{heading + message}}
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/first.js"></script>
  </body>
</html>
```

```
var firstApp = angular.module('firstApp', []);
firstApp.controller('FirstController', function($scope) {
  $scope.first = 'Some';
  $scope.last = 'One';
  $scope.heading = 'Message: ';
  $scope.updateMessage = function() {
    $scope.message = 'Hello ' + $scope.first +' '+ $scope.last + '!';
  };
});
```

```
<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
<script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
```

```
angular.element() === jQuery() === $()
angular.module('myApp', [])
 .directive('myDirective', function() {
      . . .
      link: function(scope, elem, attrs, photosControl) {
        //elem is a jQuery lite object
        elem.addClass(...);
      }
    };
```

```
<div ng-click="clicked($event)">Click Me</div>
```

```
$scope.clicked = function(event){
  var jQueryElement = angular.element(event.target);
};
```

```
angular.module(name, [requires], [configFn])
```

```
var myModule = angular.module('myModule', ['$window', '$http'], function(){
    $provide.value('myValue', 'Some Value');
});
```

```
var myModule2 = angular.module('myModule', []);
var myModule3 = angular.module('myModule');
var mod = angular.module('myMod', []);
mod.controller('myController', function($scope) {
  $scope.someValue = 'Some Value';
});
```

```
var mod = angular.module('myMod', []);
mod.constant("cID", "ABC");
mod.value('counter', 0);
mod.value('image', {name:'box.jpg', height:12, width:20});
```

```
var myController = function($scope, appMsg) {
  $scope.message = appMsg;
};
controller['$inject'] = ['$scope', 'appMsg'];
myApp.myController('controllerA', controller);
```

```
[providerA, providerB, . . ., function(objectA, objectB, . . .) {} ]
```

```
myApp.controller('controllerA', ['$scope', 'appMsg', function($scope, appMsg) {
  $scope.message = appMsg;
}]);
```

```
myMod.controller('controllerA', ['$scope', '$window',
                                  function($scope, $window) {
```

```
$scope.message = "My Module Has Loaded!";
```

```
$window.alert($scope.message);
```

```
var myMod = angular.module('myApp', []);
myMod.controller('controllerA', ['$scope', '$window',
                                  function($scope, $window) {
  $scope.message = "My Module Has Loaded!";
  $window.alert($scope.message);
}]);
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Dependency Injection</title>
  </head>
  <body>
    <div ng-controller="controllerA">
      <h2>This Page has an Alert</h2>
    </div><hr>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/inject_builtin.js"></script>
  </body>
</html>
```

```
var myMod = angular.module('myMod', []);
```

```
myMod.value('modMsg', 'Hello from My Module');
myMod.controller('controllerB', ['$scope', 'modMsg',
                                 function($scope, msg) {
  $scope.message = msg;
}]);
```

```
var myApp = angular.module('myApp', ['myMod']);
```

```
02 <html ng-app="myApp">
. . .
07      <div ng-controller="controllerA">
. . .
11      <div ng-controller="controllerB">
```

```
var myMod = angular.module('myMod', []);
myMod.value('modMsg', 'Hello from My Module');
myMod.controller('controllerB', ['$scope', 'modMsg',
                                 function($scope, msg) {
  $scope.message = msg;
}]);
var myApp = angular.module('myApp', ['myMod']);
myApp.value('appMsg', 'Hello from My App');
myApp.controller('controllerA', ['$scope', 'appMsg',
                                 function($scope, msg) {
  $scope.message = msg;
}]);
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Dependency Injection</title>
  </head>
  <body>
    <div ng-controller="controllerA">
      <h2>Application Message:</h2>
      {{message}}
    </div><hr>
    <div ng-controller="controllerB">
      <h2>Module Message:</h2>
      {{message}}
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="/js/injector_custom.js"></script>
  </body>
</html>
```

```
config(function([injectable, . . .]))
```

```
var myModule = angular.module('myModule', []).
  config(function($provide, $filterProvider) {
    $provide.value("startTime", new Date());
    $filterProvider.register('myFilter', function(){});
});
```

```
run(function([injectable, . . .]))
```

```
myModule.run(function(startTime) {
  startTime.setTime((new Date()).getTime());
});
```

```
myModule.config(function($provide) {
    $provide.value("configTime", new Date());
    $provide.value("runTime", new Date());
    for(var x=0; x<1000000000; x++){
      var y = Math.sqrt(Math.log(x));
    }
});
```

```
myModule.run(function(configTime, runTime) {
  runTime.setTime((new Date()).getTime());
});
```

```
myModule.controller('controllerA',['$scope', 'configTime', 'runTime',
    function($scope, configTime, runTime){
  $scope.configTime = configTime;
  $scope.runTime = runTime;
}]);
```

```
var myModule = angular.module('myApp', []);
myModule.config(function($provide) {
    $provide.value("configTime", new Date());
    $provide.value("runTime", new Date());
    for(var x=0; x<1000000000; x++){
      var y = Math.sqrt(Math.log(x));
    }
});
myModule.run(function(configTime, runTime) {
  runTime.setTime((new Date()).getTime());
});
myModule.controller('controllerA',['$scope', 'configTime', 'runTime',
    function($scope, configTime, runTime){
  $scope.configTime = configTime;
  $scope.runTime = runTime;
}]);
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Configuration and Run Blocks</title>
  </head>
  <body>
    <div ng-controller="controllerA">
      <hr>
      <h2>Config Time:</h2>
      {{configTime}}
      <hr>
      <h2>Run Time:</h2>
      {{runTime}}
    </div><hr>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/config_run_blocks.js"></script>
  </body>
</html>
```

```
app.controller('myController', ['$scope', '$http'], function($scope, $http)
➥{ ...
```

```
+ ' ' + $window.screenX + 'x'+ $window.screenY
```

```
angular.module('myApp', [])
.run(function($rootScope) {
    $rootScope.rootValue = 5;
})
.controller('myController', function($scope, $rootScope) {
  $scope.value = 10;
  $scope.difference = function() {
        return $rootScope.rootValue - $scope.value;
    };
});
```

```
angular.module('myApp', []).
  value('start', 200).
  controller('Counter', ['$scope', 'start',
                          function($scope, startingValue) {
  }]);
```

```
angular.module('myApp', []).
  value('start', 200).
  controller('Counter', ['$scope', 'start',
                                function($scope, start) {
```

```
    $scope.start = start;
    $scope.current = start;
    $scope.difference = 0;
    $scope.change = 1;
    $scope.inc = function() {
      $scope.current += $scope.change;
      $scope.calcDiff();
    };
    $scope.dec = function() {
      $scope.current -= $scope.change;
      $scope.calcDiff();
    };
    $scope.calcDiff = function() {
      $scope.difference = $scope.current - $scope.start;
    };
```

```
angular.module('myApp', []).
  value('start', 200).
  controller('Counter', ['$scope', 'start',
                            function($scope, start) {
    $scope.start = start;
    $scope.current = start;
    $scope.difference = 0;
    $scope.change = 1;
    $scope.inc = function() {
      $scope.current += $scope.change;
      $scope.calcDiff();
    };
    $scope.dec = function() {
      $scope.current -= $scope.change;
      $scope.calcDiff();
    };
    $scope.calcDiff = function() {
      $scope.difference = $scope.current - $scope.start;
    };
  }]);
```

```html
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Basic Scope</title>
  </head>
  <body>
    <div ng-controller="Counter">
      <span>Change Amount:</span>
      <input type="number" ng-model="change"><hr>
      <span>Starting Value:</span>
      {{start}}
      <br>
      <span>CurrentValue:</span>
      {{current}}
      <button ng-click='inc()'>+</button>
      <button ng-click='dec()'>-</button><hr>
      <span>Difference:</span>
      {{difference}}
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/scope_controller.js"></script>
  </body>
</html>
```

```
<input type="number" ng-model="valueA" />
```

```
<span ng-click="addValues(valueA, valueB")>Add Values{{valueA}} & {{valueB}}</span>
```

```
    $scope.addValues = function(v1, v2) {
      $scope.valueC = v1 + v2;
    };
```

```
        ValueA: <input type="number" ng-model="valueA" /><br>
        ValueB: <input type="number" ng-model="valueB" /><br><br>
```

```
       Expression: {{valueA}} + {{valueB}}<br><br>
```

```
    Live Expression Value: {{valueA + valueB}}<br><br>
```

```
        <input type="button" ng-click="addValues(valueA, valueB)"
          value ="Click to Add Values {{valueA}} & {{valueB}}" /><br>
        Clicked Expression Value: {{valueC}}<br>
```

```
angular.module('myApp', []).
  controller('SimpleTemplate', function($scope) {
    $scope.valueA = 5;
    $scope.valueB = 7;
    $scope.valueC = 12;
    $scope.addValues = function(v1, v2) {
      $scope.valueC = v1 + v2;
    };
  });
```

```html
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Scope and Templates</title>
  </head>
  <body>
    <div ng-controller="SimpleTemplate">
      ValueA: <input type="number" ng-model="valueA" /><br>
      ValueB: <input type="number" ng-model="valueB" /><br><br>
      Expression: {{valueA}} + {{valueB}}<br><br>
      Live Expression Value: {{valueA + valueB}}<br><br>
      <input type="button" ng-click="addValues(valueA, valueB)"
        value ="Click to Add Values {{valueA}} & {{valueB}}" /><br>
      Clicked Expression Value: {{valueC}}<br>
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/scope_template.js"></script>
  </body>
</html>
```

```
$scope.watchedItem = 'myItem';
$scope.counter = 0;
$scope.$watch('watchedItem', function(newValue, oldValue) {
  $scope.watchedItem = $scope.counter + 1;
});
```

```
<div ng-controller="controllerA"> . . . </div>
<div ng-controller="controllerB"> . . . </div>
```

```
<div ng-controller="controllerA">
  <div ng-controller="controllerB"> . . . </div>
</div>
```

```
controller('LevelA', function($scope) {
  $scope.title = "Level A"
  $scope.valueA = 1;
  $scope.inc = function() {
    $scope.valueA++;
  };
}).
```

```
<div ng-controller="LevelA">
. . .
  <div ng-controller="LevelB"><hr>
. . .
    <div ng-controller="LevelC"><hr>
. . .
    </div>
  </div>
</div>
```

```
    <h3>{{title}}</h3>
    ValueA = {{valueA}} <input type="button" ng-click="inc()" value="+" />
```

```
        <h3>{{title}}</h3>
        ValueA = {{valueA}}<br>
        ValueB = {{valueB}}
        <input type="button" ng-click="inc()" value="+" />
        <h3>{{title}}</h3>
         ValueA = {{valueA}}<br>
         ValueB = {{valueB}}<br>
         ValueC = {{valueC}}
         <input type="button" ng-click="inc()" value="+" />
```

```
angular.module('myApp', []).
  controller('LevelA', function($scope) {
    $scope.title = "Level A"
    $scope.valueA = 1;
    $scope.inc = function() {
      $scope.valueA++;
    };
  }).
  controller('LevelB', function($scope) {
    $scope.title = "Level B"
    $scope.valueB = 1;
    $scope.inc = function() {
      $scope.valueB++;
    };
  }).
  controller('LevelC', function($scope) {
    $scope.title = "Level C"
    $scope.valueC = 1;
    $scope.inc = function() {
      $scope.valueC++;
    };
  });
```

```
<!doctype html>
<html ng-app="myApp">
<head>
<title>AngularJS Scope Hierarchy</title>
</head>
<body>
  <div ng-controller="LevelA">
    <h3>{{title}}</h3>
    ValueA = {{valueA}} <input type="button" ng-click="inc()" value="+" />
    <div ng-controller="LevelB"><hr>
      <h3>{{title}}</h3>
      ValueA = {{valueA}}<br>
      ValueB = {{valueB}}
      <input type="button" ng-click="inc()" value="+" />
      <div ng-controller="LevelC"><hr>
       <h3>{{title}}</h3>
        ValueA = {{valueA}}<br>
        ValueB = {{valueB}}<br>
        ValueC = {{valueC}}
        <input type="button" ng-click="inc()" value="+" />
      </div>
    </div>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/scope_hierarchy.js"></script>
</body>
</html>
```

```
<div ng-controller="controllerA">
  <div ng-controller="controllerB">
    <div ng-controller="controllerC">. . . </div>
  </div>
</div>
```

```
<p>{{1+2}}</p>
href="/myPage.html/{{hash}}"
```

```
<span ng-click="scopeFunction()"></span>
<span ng-click="scopeFunction(scopeVariable, 'stringParameter')"></span>
<span ng-click="scopeFunction(5*scopeVariable)"></span>
```

```
<span ng-click="msg='clicked'"></span>
```

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
  });
```

```
{{'My String'}}<hr>
```

```
{{'String1' + ' ' + 'String2'}}<hr>
```

```
22          {{5 + '+' + 5 + '='}}{{5+5}}<hr>
```

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
  });
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Expressions</title>
    <style>
      p{margin:0px;}
      p:after{color:red;}
    </style>
  </head>
  <body>
    <div ng-controller="myController">
      <h1>Expressions</h1>
      Number:<br>
      {{5}}<hr>
      String:<br>
      {{'My String'}}<hr>
      Adding two strings together:<br>
      {{'String1' + ' ' + 'String2'}}<hr>
      Adding two numbers together:<br>
      {{5+5}}<hr>
      Adding strings and numbers together:<br>
      {{5 + '+' + 5 + '='}}{{5+5}}<hr>
      Comparing two numbers with each other:<br>
      {{5===5}}<hr>
    <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
    <script src="js/expressions_basic.js"></script>
  </body>
</html>
```

```
  $scope.speed = 'Slow';
  $scope.vehicle = 'Train';
  $scope.newSpeed = 'Hypersonic';
  $scope.newVehicle = 'Plane';
```

```
    $scope.upper = function(aString){
      return angular.uppercase(aString);
    };
    $scope.lower = function(aString){
      return angular.lowercase(aString);
    };
```

```
  $scope.setValues = function(speed, vehicle){
    $scope.speed = speed;
    $scope.vehicle = vehicle;
  };
```

{{speed}} {{vehicle}}<hr>

```
        {{speed + ' ' + vehicle}}<hr>
        {{lower(speed)}} {{upper('Jeep')}}<hr>
```

```
    <a ng-click="setValues('Fast', newVehicle)">
      Click to change to Fast {{newVehicle}}</a><hr>
      <a ng-click="setValues(newSpeed, 'Rocket')">
        Click to change to {{newSpeed}} Rocket</a><hr>
        <a ng-click="vehicle='Car'">
          Click to change the vehicle to a Car</a><hr>
        <a ng-click="vehicle='Enhanced ' + vehicle">
          Click to Enhance Vehicle</a><hr>
```

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
    $scope.speed = 'Slow';
    $scope.vehicle = 'Train';
    $scope.newSpeed = 'Hypersonic';
    $scope.newVehicle = 'Plane';
    $scope.upper = function(aString){
      return angular.uppercase(aString);
    };
    $scope.lower = function(aString){
      return angular.lowercase(aString);
    };
    $scope.setValues = function(speed, vehicle){
      $scope.speed = speed;
      $scope.vehicle = vehicle;
    };
  });
```

```html
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Expressions</title>
    <style>
      a{color: blue; text-decoration: underline; cursor: pointer}
    </style>
  </head>
  <body>
    <div ng-controller="myController">
      Directly accessing variables in the scope:<br>
      {{speed}} {{vehicle}}<hr>
      Adding variables in the scope:<br>
      {{speed + ' ' + vehicle}}<hr>
      Calling function in the scope:<br>
      {{lower(speed)}} {{upper('Jeep')}}<hr>
      <a ng-click="setValues('Fast', newVehicle)">
        Click to change to Fast {{newVehicle}}</a><hr>
      <a ng-click="setValues(newSpeed, 'Rocket')">
        Click to change to {{newSpeed}} Rocket</a><hr>
      <a ng-click="vehicle='Car'">
        Click to change the vehicle to a Car</a><hr>
      <a ng-click="vehicle='Enhanced ' + vehicle">
        Click to Enhance Vehicle</a><hr>
    <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
    <script src="js/expressions_scope.js"></script>
  </body>
</html>
```

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
    $scope.Math = window.Math;
    $scope.myArr = [1];
    $scope.removedArr = [];
  });
```

```
13            {{myArr}}<hr>
. . .
15            {{removedArr}}<hr>
```

```
      <a ng-click="myArr.push(Math.floor(Math.random()*100 + 1))">
        Click to append a value to the array</a><hr>
      <a ng-click="removedArr.push(myArr.shift())">
        Click to remove the first value from the array</a><hr>
```

```
          {{myArr.length}}<hr>
          {{Math.cos(removedArr.length)}}<hr>
```

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
    $scope.Math = window.Math;
    $scope.myArr = [1];
    $scope.removedArr = [];
  });
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Expressions</title>
    <style>
      a{color: blue; text-decoration: underline; cursor: pointer}
    </style>
  </head>
  <body>
    <div ng-controller="myController">
      <h1>Expressions</h1>
      Array:<br>
        {{myArr}}<hr>
      Elements removed from array:<br>
        {{removedArr}}<hr>
      <a ng-click="myArr.push(Math.floor(Math.random()*100 + 1))">
        Click to append a value to the array</a><hr>
      <a ng-click="removedArr.push(myArr.shift())">
        Click to remove the first value from the array</a><hr>
      Size of Array:<br>
        {{myArr.length}}<hr>
      Cosine of the length of the removed array:<br>
        {{Math.cos(removedArr.length)}}<hr>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/expressions_javascript.js"></script>
  </body>
</html>
```

```
{{ expression | filter | filter }}
{{ expression | filter:parameter1:parameter2 }}
```

```
controller('myController', ['$scope', 'currencyFilter',
                            function($scope, myCurrencyFilter){
  $scope.getCurrencyValue = function(value){
    return myCurrencyFilter(value, "$USD");
  };
}]);
```

```
.controller('myController', function($scope) {
  $scope.currentDate = new Date();
  $scope.JSONObj = { title: "myTitle" };
  $scope.word="Supercalifragilisticexpialidocious";
  $scope.days=['Monday', 'Tuesday', 'Wednesday',
                  'Thursday', 'Friday'];
});
```

```
          Number: {{123.45678|number:3}}<br>
```

```
    Currency: {{123.45678|currency:"$"}}<br>
```

```
      Date: {{ currentDate | date:'yyyy-MM-dd HH:mm:ss Z'}}<br>
```

```
        JSON: {{ JSONObj | json }}<br>
```

```
    Limit Array: {{ days | limitTo:3 }}<br>
```

```
        Limit String: {{ word | limitTo:9 }}<br>
        Uppercase: {{ word | uppercase | limitTo:9 }}<br>
        Lowercase: {{ word | lowercase | limitTo:9 }}
```

```
angular.module('myApp', [])
  .controller('myController', function($scope) {
    $scope.currentDate = new Date();
    $scope.JSONObj = { title: "myTitle" };
    $scope.word="Supercalifragilisticexpialidocious";
    $scope.days=['Monday', 'Tuesday', 'Wednesday',
                 'Thursday', 'Friday'];
  });
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Filters</title>
  </head>
  <body>
    <div ng-controller="myController">
      <h2>Basic Filters</h2>
      Number: {{123.45678|number:3}}<br>
      Currency: {{123.45678|currency:"$"}}<br>
      Date: {{ currentDate | date:'yyyy-MM-dd HH:mm:ss Z'}}<br>
      JSON: {{ JSONObj | json }}<br>
      Limit Array: {{ days | limitTo:3 }}<br>
      Limit String: {{ word | limitTo:9 }}<br>
      Uppercase: {{ word | uppercase | limitTo:9 }}<br>
      Lowercase: {{ word | lowercase | limitTo:9 }}
    <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
    <script src="js/filters.js"></script>
  </body>
</html>
```

```
.controller('myController', ['$scope', 'filterFilter',
                              function($scope, filterFilter) {
```

```
$scope.planes = [
  {make: 'Boeing', model: '777', capacity: 440},
. . .
$scope.filteredPlanes = $scope.planes;
$scope.reverse = true;
$scope.column = 'make';
```

```
    $scope.setSort = function(column){
      $scope.column = column;
      $scope.reverse = !$scope.reverse;
    };
```

```
  $scope.filterString = '';
  $scope.setFilter = function(value){
    $scope.filteredPlanes =
      filterFilter($scope.planes, $scope.filterString);
  };
```

```
      <input type="text" ng-model="filterString">
      <input type="button" ng-click="setFilter()" value="Filter">
```

```
          <th ng-click="setSort('make')">Make</th>
          <th ng-click="setSort('model')">Model</th>
          <th ng-click="setSort('capacity')">Capacity</th>
```

```
        <tr ng-repeat=
            "plane in filteredPlanes | orderBy:column:reverse">
          <td>{{plane.make}}</td>
          <td>{{plane.model}}</td>
          <td>{{plane.capacity}}</td>
        </tr>
```

```
angular.module('myApp', [])
  .controller('myController', ['$scope', 'filterFilter',
                              function($scope, filterFilter) {
    $scope.planes = [
      {make: 'Boeing', model: '777', capacity: 440},
      {make: 'Boeing', model: '747', capacity: 700},
      {make: 'Airbus', model: 'A380', capacity: 850},
      {make: 'Airbus', model: 'A340', capacity: 420},
      {make: 'McDonnell Douglas', model: 'DC10', capacity: 380},
      {make: 'McDonnell Douglas', model: 'MD11', capacity: 410},
      {make: 'Lockheed', model: 'L1011', capacity: 380}];
    $scope.filteredPlanes = $scope.planes;
    $scope.reverse = true;
    $scope.column = 'make';
    $scope.setSort = function(column){
      $scope.column = column;
      $scope.reverse = !$scope.reverse;
    };
    $scope.filterString = '';
    $scope.setFilter = function(value){
      $scope.filteredPlanes =
        filterFilter($scope.planes, $scope.filterString);
    };
  }]);
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Sorting and Filtering</title>
    <style>
      table{text-align:right;}
      td,th{padding:3px;}
      th{cursor:pointer;}
    </style>
  </head>
  <body>
    <div ng-controller="myController">
      <h2>Sorting and Filtering</h2>
      <input type="text" ng-model="filterString">
      <input type="button" ng-click="setFilter()" value="Filter">
      <table>
      <tr>
        <th ng-click="setSort('make')">Make</th>
        <th ng-click="setSort('model')">Model</th>
        <th ng-click="setSort('capacity')">Capacity</th>
      </tr>
      <tr ng-repeat=
          "plane in filteredPlanes | orderBy:column:reverse">
        <td>{{plane.make}}</td>
        <td>{{plane.model}}</td>
        <td>{{plane.capacity}}</td>
      </tr>
      </table>
    <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
    <script src="js/filter_sort.js"></script>
  </body>
</html>
```

```
filter('myFilter', function(){
  return function(input, param1, param2){
    return <<modified input>>;
  };
});
```

```
  .filter('censor', function() {
    return function(input, replacement) {
      var cWords = ['bad', 'evil', 'dark'];
      var out = input;
      for(var i=0; i<cWords.length; i++){
        var regex = new RegExp(cWords[i], "gi");
        out = out.replace(regex, replacement);
      }
      return out;
    };
  })
```

```
.controller('myController', ['$scope', 'censorFilter',
                              function($scope, myCensorFilter) {
  $scope.censorText = "***";
  $scope.phrase="This is a bad phrase.";
  $scope.txt = "Click to filter out dark and evil.";
  $scope.filterText = function(){
    $scope.txt = myCensorFilter($scope.txt, '<<censored>>');
  };
}]);
```

```
        {{phrase | censor:censorText}}<br>
```

```
{{"This is some bad, dark, evil text." | censor:"happy"}}
```

```
        <p ng-click="filterText()">{{txt}}</p>
```

```
angular.module('myApp', [])
  .filter('censor', function() {
    return function(input, replacement) {
      var cWords = ['bad', 'evil', 'dark'];
      var out = input;
      for(var i=0; i<cWords.length; i++){
        var regex = new RegExp(cWords[i], "gi");
        out = out.replace(regex, replacement);
      }
      return out;
    };
  })
  .controller('myController', ['$scope', 'censorFilter',
                               function($scope, myCensorFilter) {
    $scope.censorText = "***";
    $scope.phrase="This is a bad phrase.";
    $scope.txt = "Click to filter out dark and evil.";
    $scope.filterText = function(){
      $scope.txt = myCensorFilter($scope.txt, '<<censored>>');
    };
  }]);
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Custom Filter</title>
  </head>
  <body>
    <div ng-controller="myController">
      <h2>Sorting and Filtering</h2>
      {{phrase | censor:censorText}}<br>
      {{"This is some bad, dark, evil text." | censor:"happy"}}
      <p ng-click="filterText()">{{txt}}</p>
    <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
    <script src="js/filter_custom.js"></script>
  </body>
</html>
```

```
{{myNewPhrase | censor:"XXX" }}<br>
```

```
[<a ng-click="titleBar='large_title.html'">Make Title Large</a>]
[<a ng-click="titleBar='small_title.html'">Make Title Small</a>]
```

```
<div ng-include="titleBar"></div>
```

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.titleBar = "small_title.html";
  });
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Data Include Directive</title>
  <style>
    .large{
      background-color: blue; color: white;
      text-align: center;
      font: bold 50px/80px verdana, serif;
      border: 6px black ridge; }
    .small{
      background-color: lightgrey;
      text-align: center;
      border: 1px black solid; }
    a{
     color: blue; text-decoration: underline;
     cursor: pointer; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    [<a ng-click="titleBar='large_title.html'">Make Title Large</a>]
    [<a ng-click="titleBar='small_title.html'">Make Title Small</a>]
    <div ng-include="titleBar"></div>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_angular_include.js"></script>
</body>
</html>
```

```
<input type="text" ng-model="someText"> {{someText}}<hr>
```

```
<input type="checkbox" ng-model="cbValue"
       ng-true-value="'Checked'" ng-false-value="'Not Checked'">
Checkbox: {{cbValue}}<hr>
```

```
<input type="radio"
  ng-model="cameraName" value="Canon"> Canon<br/>
<input type="radio"
  ng-model="cameraName" value="Nikon"> Nikon<br/>
Selected Camera: {{cameraName}} <hr>
```

```
<select ng-model="camera"
  ng-options="c.model group by c.make for c in cameras">
</select>
{{camera|json}}
```

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.cameras = [
      {make:'Canon', model:'70D', mp:20.2},
      {make:'Canon', model:'6D', mp:20},
      {make:'Nikon', model:'D7100', mp:24.1},
      {make:'Nikon', model:'D5200', mp:24.1}];
    $scope.cameraObj=$scope.cameras[0];
    $scope.cameraName = 'Canon';
    $scope.cbValue = '';
    $scope.someText = '';
  });
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Form Directives</title>
</head>
<body>
  <div ng-controller="myController">
    <h2>Forms Directives</h2>
    <input type="text" ng-model="someText"> {{someText}}<hr>
    <input type="checkbox" ng-model="cbValue"
           ng-true-value="'Checked'" ng-false-value="'Not Checked'">
    Checkbox: {{cbValue}}<hr>
    <input type="radio"
      ng-model="cameraName" value="Canon"> Canon<br/>
    <input type="radio"
      ng-model="cameraName" value="Nikon"> Nikon<br/>
    Selected Camera: {{cameraName}} <hr>
    <select ng-model="camera"
      ng-options="c.model group by c.make for c in cameras">
    </select>
    {{camera|json}}
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_form.js"></script>
</body>
</html>
```

```
<a ng-href="/{{hash}}/index.html">{{hash}}</a>.
```

```
<label ng-repeat="color in colors">
  {{color}}
  <input type="radio" ng-model="myStyle['background-color']"
         ng-value="color" id="{{color}}" name="mColor">
</label>
```

```
<span class="rect" ng-style="myStyle"></span><hr>
```

```
<li ng-repeat="day in days">
  <span ng-class-even="'even'">{{day}}</span>
</li><hr>
```

```
Show Message: <input type="checkbox" ng-model="checked" />
<p ng-if="checked" ng-bind="msg"> </p>
```

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.colors=['red','green','blue'];
    $scope.myStyle = { "background-color": 'blue' };
    $scope.days=['Monday', 'Tuesday', 'Wednesday',
                 'Thursday', 'Friday'];
    $scope.msg="Dynamic Message from the model";
  });
```

```html
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Data Binding Directives</title>
  <style>
    .even{background-color:lightgrey;}
    .rect{display:inline-block; height:40px; width:100px;}
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h2>Data Binding Directives</h2>
    <label ng-repeat="color in colors">
      {{color}}
      <input type="radio" ng-model="myStyle['background-color']"
             ng-value="color" id="{{color}}" name="mColor">
    </label>
    <span class="rect" ng-style="myStyle"></span><hr>
    <li ng-repeat="day in days">
      <span ng-class-even="'even'">{{day}}</span>
    </li><hr>
    Show Message: <input type="checkbox" ng-model="checked" />
    <p ng-if="checked" ng-bind="msg"> </p>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_bind.js"></script>
</body>
</html>
```

```
$scope.setTitle = function(title){
  $scope.title = title;
};
```

```
<input type="button" ng-click="setTitle('New Title')">
```

```
<input type="button" ng-click="myClick($event)">
```

```
    $scope.inputData = { input1: {value: "", state: ""},
                         input2: {value: "", state: ""} };
    $scope.focusGained = function(input){
      $scope.inputData[input]['value'] = '';
      $scope.inputData[input]['state'] = 'Focus Gained';
    };
    $scope.focusLost = function(event, input){
      var element = angular.element(event.target);
      var value = element.val();
      $scope.inputData[input]['value'] = value.toUpperCase();
      $scope.inputData[input]['state'] = "Focus Lost";
    };
```

```
<input type="text"
  ng-blur="focusLost($event, 'input1')"
  ng-focus="focusGained('input1')"><br>
. . .
<input type="text"
  ng-blur="focusLost($event, 'input2')"
  ng-focus="focusGained('input2')"><hr>
```

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.inputData = { input1: {value: "", state: ""},
                         input2: {value: "", state: ""} };
    $scope.focusGained = function(input){
      $scope.inputData[input]['value'] = '';
      $scope.inputData[input]['state'] = 'Focus Gained';
    };
    $scope.focusLost = function(event, input){
      var element = angular.element(event.target);
      var value = element.val();
      $scope.inputData[input]['value'] = value.toUpperCase();
      $scope.inputData[input]['state'] = "Focus Lost";
    };
  });
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Focus Event Directives</title>
</head>
<body>
  <div ng-controller="myController">
    <h2>Focus Event Directives</h2>
    Input 1:<br>
    <input type="text"
      ng-blur="focusLost($event, 'input1')"
      ng-focus="focusGained('input1')"><br>
    Input 2:<br>
    <input type="text"
      ng-blur="focusLost($event, 'input2')"
      ng-focus="focusGained('input2')"><hr>
    Input Data: {{inputData|json}}<br/>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_focus_events.js"></script>
</body>
</html>
```

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.storedString = '';
    $scope.keyInfo = {};
    $scope.keyStrokes = [];
    $scope.keyState = 'Not Pressed';
    $scope.keyPressed = function(event){
      if (event.keyCode == 13){
        var element = angular.element(event.target);
        $scope.storedString = element.val();
        element.val('');
        $scope.keyInfo.keyCode = event.keyCode;
        $scope.keyStrokes = [];
        $scope.keyState = 'Enter Pressed';
      } else {
        $scope.keyInfo.keyCode = event.keyCode;
        $scope.keyStrokes.push(event.keyCode);
        $scope.keyState = 'Not Pressed';
      }
    };
  });
```

```html
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Keyboard Event Directives</title>
</head>
<body>
  <div ng-controller="myController">
    <h2>Keyboard Event Directives</h2>
    <input type="text"
        ng-keydown="keyState='Pressed'"
        ng-keyup="keyPressed($event)"><hr>
    Keyboard State:<br>
      {{keyState}}<hr>
    Last Key:<br>
      {{keyInfo|json}}<hr>
    Stored String:<br>
      {{storedString}}<hr>
    Recorded Key Strokes:<br>
      {{keyStrokes}}
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_keyboard_events.js"></script>
</body>
</html>
```

```
  $scope.mouseClick = function(event){
    $scope.lastClickInfo.clientX = event.clientX;
    $scope.lastClickInfo.clientY = event.clientY;
    $scope.lastClickInfo.screenX = event.screenX;
    $scope.lastClickInfo.screenY = event.screenY;
  };
```

```
    $scope.mouseMove = function(event){
      $scope.mouseInfo.clientX = event.clientX;
      $scope.mouseInfo.clientY = event.clientY;
      $scope.mouseInfo.screenX = event.screenX;
      $scope.mouseInfo.screenY = event.screenY;
    };
```

```
<img
    src="/images/img3.jpg"
    ng-mouseenter="mouseState='Entered'"
    ng-mouseleave="mouseState='Left'"
    ng-mouseclick="mouseState='Clicked'"
    ng-mousedown="mouseState='Down'"
    ng-mouseup="mouseState='Up'"
    ng-click="mouseClick($event)"
    ng-mousemove="mouseMove($event)"></img><hr>
```

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.mouseInfo = {};
    $scope.lastClickInfo = {};
    $scope.mouseClick = function(event){
      $scope.lastClickInfo.clientX = event.clientX;
      $scope.lastClickInfo.clientY = event.clientY;
      $scope.lastClickInfo.screenX = event.screenX;
      $scope.lastClickInfo.screenY = event.screenY;
    };
    $scope.mouseMove = function(event){
      $scope.mouseInfo.clientX = event.clientX;
      $scope.mouseInfo.clientY = event.clientY;
      $scope.mouseInfo.screenX = event.screenX;
      $scope.mouseInfo.screenY = event.screenY;
    };
  });
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Event Directives</title>
  <style>
    img {
      border: 3px ridge black;
      height: 200px; width: 200px;
      display: inline-block;
    }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h2>Event Directives</h2>
    <img
        src="/images/img3.jpg"
        ng-mouseenter="mouseState='Entered'"
        ng-mouseleave="mouseState='Left'"
        ng-mouseclick="mouseState='Clicked'"
        ng-mousedown="mouseState='Down'"
        ng-mouseup="mouseState='Up'"
        ng-click="mouseClick($event)"
        ng-mousemove="mouseMove($event)"></img><hr>
    Mouse State: {{mouseState}}<br/>
    Mouse Position Info: {{mouseInfo|json}}<br/>
    Last Click Info: {{lastClickInfo|json}}<br/>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_mouse_events.js"></script>
</body>
</html>
```

```
angular.module('myApp', []).
  directive('myDirective', function() {
    return {
      template: 'Name: {{name}} Score: {{score}}'
    };
  });
```

```
directive('myDirective', function() {
  return {
    template: 'Name: {{name}} Score: {{score}}'
  };
});
```

```
directive('myDirective', function() {
  return {
    transclude: true,
    template: '<div ng-transclude></div>'
  };
});
```

```
directive('myDirective', function() {
  return {
    templateUrl: '/myDirective.html'
  };
});
```

```
<div my-directive="expression"></div>
```

```
<div class="my-directive: expression;"></div>
```

```
<!-- directive: my-directive expression -->
```

```
directive('myDirective', function() {
  return {
    restrict: 'AE',
    templateUrl: '/myDirective.html'
  };
});
```

```
directive('myDirective', function() {
  return {
    scope: {title: '='},
    controller: function ($scope){
      $scope.title = "new";
      $scope.myFunction = function(){
      });
    }
  };
});
```

```
directive('myDirective', function() {
  return {
    require: '^myController'
  };
});
```

```
directive('myDirective', function() {
  return {
    require: '^myController',
    link: function(scope, element, attrs, injectedMyController){
          }
  };
});
```

```
 directive('myDirective', function() {
  return {
    require: ['^myControllerA', '^myControllerB'],
    link: function(scope, element, attrs, requiredControllers){
            var controllerA = requiredControllers[0];
            var controller = requiredControllers[1];
          }
  };
});
```

```
directive('myDirective', function() {
  return {
    require: '^myOtherDirective',
    link: function(scope, element, attrs, otherDirectiveController){
          }
  };
});
```

```
directive('myDirective', function() {
  return {
    scope: true
  };
});
```

```
directive('myDirective', function() {
  return {
    scope: { },
    templateUrl: '/myDirective.html'
  };
});
```

```
angular.module('myApp', []).
  controller('myController', function($scope) {
    $scope.title="myApplication";
    $scope.myFunc = function(){
      console.log("out");
    };
  }).
  directive('myDirective', function() {
    return {
      scope: {title: '=', newFunc:"&myFunc", info: '@'},
      template: '<div ng-click="newFunc()">{{title}}: {{info}}</div>'
    };
  });
```

```
angular.module('myApp', []).
  directive('myDirective', function() {
    return {
      transclude: true,
      scope: {},
      template: '<div ng-transclude>{{title}}</div>'
    };
  }).
  controller('myController', function($scope) {
    $scope.title="myApplication";
  });
```

```
link: function(scope, element, attributes, [controller], [transclude])
```

```
link: function link(scope, elem, attr, controller, transcludeFn){
        var transcludedElement = transcludeFn();
      }
```

```
link: function link(scope, elem, attr, controller, transcludeFn){
        transcludeFn(function(clone){
          //access clone here . . .
        });
      }
```

```
link: function link(scope, elem, attr, controller, transcludeFn){
        transcludeFn(scope, function(clone){
          //access clone here . . .
        });
      }
```

```
directive('myDirective', function() {
  return {
    scope: {title: '='},
    require: '^otherDirective',
    link: function link(scope, elem, attr, controller, transclude){
      scope.title = "new";
      elem.append("Linked");
      elem.on('$destroy', function() {
        //cleanup code
      });
      scope.$watch('title', function(newVal){
        //watch code
      });
    }
  };
```

```
directive('myDirective', function() {
  return {
  link: {
    pre: function preLink(scope, elem, attr, controller){
           //prelink code
         },
    post: function postLink(scope, elem, attr, controller){
           //postlink code
         },
  }
};
```

```
directive('myDirective', function() {
  return {
  compile: function compile(scope, elem, attr, controller){
             //postlink code
           }
};
```

```
directive('myDirective', function() {
  return {
  compile: {
    pre: function preLink(elem, attr){
           //prelink code
         },
    post: function postLink(elem, attr){
           //postlink code
         },
  }
};
```

```
angular.module('myApp', [])
.controller('myController', function($scope) {
    $scope.title="myApplication";
  })
.directive('mybox', function() {
  return {
. . .
    };
  });
```

```
    scope: {title: '@', bwidth: '@bwidth'},
    template: '<div><span class="titleBar">{{title}}' +
              '</span><div ng-transclude></div></div>',
    link: function (scope, elem, attr, controller, transclude){
        elem.append('<span class="footer">' + scope.$parent.title + '</
➥span>');
        elem.css('display', 'inline-block');
        elem.css('width', scope.bwidth);
      },
angular.module('myApp', [])
.controller('myController', function($scope) {
    $scope.title="myApplication";
  })
.directive('mybox', function() {
  return {
    transclude: true,
    restrict: 'E',
    scope: {title: '@', bwidth: '@bwidth'},
    template: '<div><span class="titleBar">{{title}}' +
              '</span><div ng-transclude></div></div>',
    link: function (scope, elem, attr, controller, transclude){
        elem.append('<span class="footer">' + scope.$parent.title + '</span>');
        elem.css('display', 'inline-block');
        elem.css('width', scope.bwidth);
      },
    };
  });
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Directive</title>
  <style>
    * { text-align: center; }
    .titleBar { color: white; background-color: grey;
                font: bold 14px/18px arial; display: block;
                border-bottom: 4px ridge grey; }
    .footer {  color: white; background-color: grey;
               font: italic 10px/14px arial; display: block;
               border-top: 4px ridge grey; }
    mybox { border: 4px ridge grey; margin: 10px; }
    img { display: block; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h2>Custom Directive Manipulating the DOM</h2>
    <mybox title="Simple Text" bwidth="100px">
      Using AngularJS to build a box around text.
    </mybox>
    <mybox title="Paragraph" bwidth="200px">
      <p>Using AngularJS to build a box around a paragraph.</p>
    </mybox>
    <mybox title="List" bwidth="200px">
      <ul>
        <li>Using AngularJS</li>
        <li>to build a box</li>
        <li>around a list.
        </li>
      </ul>
    </mybox>
    <mybox title="Image" bwidth="400px">
      <img src="/images/sunset2.jpg" width="400px" />
    </mybox>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_custom_dom.js"></script>
</body>
</html>
```

```
angular.module('myApp', [])
.directive('zoomit', function() {
  return {
    link: function (scope, elem, attr){
. . .
})
.directive('fadeit', function() {
  return {
    link: function (scope, elem, attr){
. . .
    }
  };
});
```

```
    elem.on('mouseup', function(){
      dragging = false;
    });
    elem.on('mouseleave', function(){
      dragging = false;
    });
```

```
elem.on('mousemove', function(event){
  if(dragging){
    var adjustment = null;
    if ( event.screenX > lastX+tolerance &&
        elem.width() < 300){
      adjustment = 1.1;
    } else if ( event.screenX < lastX-tolerance &&
        elem.width() > 100){
      adjustment = .9;
    }
    if(adjustment){
      //requires full jQuery library
      elem.width(elem.width()*adjustment);
      elem.height(elem.height()*adjustment);
      lastX = event.screenX;
    }
  }
});
```

```
<img src="/images/wheel.jpg" zoomit></img>
<img src="/images/wheel.jpg" fadeit></img>
<img src="/images/wheel.jpg" zoomit fadeit></img>
```

```
angular.module('myApp', [])
.directive('zoomit', function() {
  return {
    link: function (scope, elem, attr){
      var dragging = false;
      var tolerance = 10;
      var lastX = 0;
      elem.on('mousedown', function(event){
        lastX = event.screenX;
        event.preventDefault();
        dragging = true;
      });
      elem.on('mouseup', function(){
        dragging = false;
      });
      elem.on('mouseleave', function(){
        dragging = false;
      });
      elem.on('mousemove', function(event){
        if(dragging){
          var adjustment = null;
          if ( event.screenX > lastX+tolerance &&
              elem.width() < 300){
            adjustment = 1.1;
          } else if ( event.screenX < lastX-tolerance &&
              elem.width() > 100){
            adjustment = .9;
          }
          if(adjustment){
            //requires full jQuery library
            elem.width(elem.width()*adjustment);
            elem.height(elem.height()*adjustment);
            lastX = event.screenX;
          }
        }
      });
    }
  };
})
.directive('fadeit', function() {
  return {
    link: function (scope, elem, attr){
      var dragging = false;
      var tolerance = 10;
      var lastY = 0;
      elem.on('mousedown', function(event){
        lastY = event.screenY;
        event.preventDefault();
        dragging = true;
      });
      elem.on('mouseup', function(){
        dragging = false;
      });
      elem.on('mouseleave', function(){
        dragging = false;
      });
      elem.on('mousemove', function(event){
        if(dragging){
          var adjustment = null;
          var currentOpacity = parseFloat(elem.css("opacity"));
          if ( event.screenY > lastY+tolerance &&
              currentOpacity < 1){
            adjustment = 1.1;
          } else if ( event.screenY < lastY-tolerance &&
              currentOpacity > 0.5){
            adjustment = .9;
          }
          if(adjustment){
            //requires full jQuery library
            elem.css("opacity", currentOpacity*adjustment);
            lastY = event.screenY;
          }
        }
      });
    }
  };
});
```

```html
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Directive</title>
  <style>
    img { width: 200px; }
  </style>
</head>
<body>
  <h2>Custom Directive Zoom and Fade</h2>
  <img src="/images/wheel.jpg" zoomit></img>
  <img src="/images/wheel.jpg" fadeit></img>
  <img src="/images/wheel.jpg" zoomit fadeit></img>
  <p>Drag up to fade out and down to fade in.</p>
  <p>Drag left to zoom out and right to zoom in.</p>
  <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_custom_zoom.js"></script>
</body>
</html>
```

```
.directive('myPhotos', function() {
  return {
    restrict: 'E',
    transclude: true,
    scope: {},
    controller: function($scope) {
      var photos = $scope.photos = [];
      $scope.select = function(photo) {
        angular.forEach(photos, function(photo) {
          photo.selected = false;
        });
        photo.selected = true;
      };
      this.addPhoto = function(photo) {
        photos.push(photo);
      };
    },
    templateUrl: 'my_photos.html'
  };
})
```

```
    templateUrl: 'my_photos.html'
```

```
<div>
  <div  class="imgList" >
      <li ng-repeat="photo in photos"
            ng-class="{active:photo.selected}">
        <a href="" ng-click="select(photo)">{{photo.title}}</a>
      </li>
  </div>
  <div class="imgView" ng-transclude></div>
</div>
```

```
.directive('myPhoto', function() {
  return {
    require: '^myPhotos',
    restrict: 'E',
    transclude: true,
    scope: { title: '@'},
    link: function(scope, elem, attrs, photosControl) {
      photosControl.addPhoto(scope);
    },
    template: '<div ng-show="selected" ng-transclude></div>'
  };
});
```

```
  <my-photos>
    <my-photo title="Leap">
      <img src="/images/power.jpg"/>
    </my-photo>
. . .
```

```
angular.module('myApp', [])
.directive('myPhotos', function() {
  return {
    restrict: 'E',
    transclude: true,
    scope: {},
    controller: function($scope) {
      var photos = $scope.photos = [];
      $scope.select = function(photo) {
        angular.forEach(photos, function(photo) {
          photo.selected = false;
        });
        photo.selected = true;
      };
      this.addPhoto = function(photo) {
        photos.push(photo);
      };
    },
    templateUrl: 'my_photos.html'
  };
})
.directive('myPhoto', function() {
  return {
    require: '^myPhotos',
    restrict: 'E',
    transclude: true,
    scope: { title: '@'},
    link: function(scope, elem, attrs, photosControl) {
      photosControl.addPhoto(scope);
    },
    template: '<div ng-show="selected" ng-transclude></div>'
  };
});
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Directive</title>
  <style>
    img {
      width: 300px }
    .imgView, .imgList {
      vertical-align: top; display:inline-block; }
  </style>
</head>
<body>
  <h2>Custom Directive Photo Flip</h2>
   <my-photos>
     <my-photo title="Leap">
       <img src="/images/power.jpg"/>
     </my-photo>
     <my-photo title="Washington">
       <img src="/images/washington.jpg"/>
     </my-photo>
     <my-photo title="Bridge">
       <img src="/images/bridge.jpg"/>
     </my-photo>
     <my-photo title="Liberty">
       <img src="/images/liberty.jpg"/>
     </my-photo>
     <my-photo title="Falls">
       <img src="/images/falls2.jpg"/>
     </my-photo>
   </my-photos>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/directive_custom_photos.js"></script>
</body>
</html>
```

```
<div>
  <div  class="imgList" >
      <li ng-repeat="photo in photos"
            ng-class="{active:photo.selected}">
        <a href="" ng-click="select(photo)">{{photo.title}}</a>
      </li>
  </div>
  <div class="imgView" ng-transclude></div>
</div>
```

```
<span
  ng-mouseenter="mouseEntered(event)"
  ng-mouseleave="mouseLeft(event)"
  ng-click="clicked(event)">
<span>
```

```
$watch(watchExpression, listener, [objectEquality])
```

```
$scope.score = 0;
$scope.$watch('score', function(newValue, oldValue) {
  if(newValue > 10){
    $scope.status = 'win';
  }
});
```

```
$scope.score = 0;
$scope.time = 0;
$scope.$watchGroup(['score', 'time'], function(newValues, oldValues) {
  if(newValues[0] > 10){
    $scope.status = 'win';
  } else if (newValues[1] > 5{
    $scope.status = 'times up';
});
```

```
$scope.scores = [5, 10, 15, 20];
$scope.$watchGroup('scores', function(newValue, oldValue) {
  $scope.newScores = newValue;
});
```

```
angular.module('myApp', [])
.controller('myController', function ($scope) {
  $scope.mColors = ['red', 'green', 'blue'];
  $scope.myColor = '';
  $scope.hits = 0;
  $scope.misses = 0;
  $scope.changes = 0;
  $scope.myObj = {color: '', hits: '', misses: ''};
  $scope.setColor = function (color){
    $scope.myColor = color;
  };
  $scope.hit = function (){
    $scope.hits += 1;
  };
  $scope.miss = function (){
    $scope.misses += 1;
  };
  $scope.$watch('myColor', function (newValue, oldValue){
    $scope.myObj.color = newValue;
  });
  $scope.$watchGroup(['hits', 'misses'], function (newValue, oldValue){
    $scope.myObj.hits = newValue[0];
    $scope.myObj.misses = newValue[1];
  });
  $scope.$watchCollection('myObj', function (newValue, oldValue){
    $scope.changes += 1;
  });
});
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Scope Variable Watch</title>
    <style>
      span { cursor: pointer; }
    </style>
  </head>
  <body>
    <h2>Watching Values in the AngularJS Scope</h2>
    <div ng-controller="myController">
      Select Color:
      <span ng-repeat="mColor in mColors">
        <span ng-style="{color: mColor}"
              ng-click="setColor(mColor)">
          {{mColor}}</span>
      </span><hr>
      <span ng-click="hit()">[+]</span>
      Hits: {{hits}}<br>
      <span ng-click="miss()">[+]</span>
      misses: {{misses}}<hr>
      Object: {{myObj|json}} <br>
      Number of Changes: {{changes}}
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/scope_watch.js"></script>
  </body>
</html>
```

```
scope.$emit(name, [args, . . .])
scope.$broadcast(name, [args, . . .])
```

```
controller('Characters', function($scope) {
  $scope.names = ['Frodo', 'Aragorn', 'Legolas', 'Gimli'];
  $scope.currentName = $scope.names[0];
...
}).
```

```
$scope.changeName = function() {
  $scope.currentName = this.name;
  $scope.$broadcast('CharacterChanged', this.name);
};
```

```
$scope.changeName = function() {
$scope.$on('CharacterDeleted', function(event, removeName){
  var i = $scope.names.indexOf(removeName);
  $scope.names.splice(i, 1);
  $scope.currentName = $scope.names[0];
  $scope.$broadcast('CharacterChanged', $scope.currentName);
});
```

```
controller('Character', function($scope) {
  $scope.info = {'Frodo': { weapon: 'Sting',
                            race: 'Hobbit'},
                 'Aragorn': { weapon: 'Sword',
                              race: 'Man'},
                 'Legolas': { weapon: 'Bow',
                              race: 'Elf'},
                 'Gimli': { weapon: 'Axe',
                            race: 'Dwarf'}};
  $scope.currentInfo = $scope.info['Frodo'];
  $scope.$on('CharacterChanged', function(event, newCharacter){
    $scope.currentInfo = $scope.info[newCharacter];
  });
  $scope.deleteChar = function() {
    delete $scope.info[$scope.currentName];
    $scope.$emit('CharacterDeleted', $scope.currentName);
  };
});
```

```
angular.module('myApp', []).
  controller('Characters', function($scope) {
    $scope.names = ['Frodo', 'Aragorn', 'Legolas', 'Gimli'];
    $scope.currentName = $scope.names[0];
    $scope.changeName = function() {
      $scope.currentName = this.name;
      $scope.$broadcast('CharacterChanged', this.name);
    };
    $scope.$on('CharacterDeleted', function(event, removeName){
      var i = $scope.names.indexOf(removeName);
      $scope.names.splice(i, 1);
      $scope.currentName = $scope.names[0];
      $scope.$broadcast('CharacterChanged', $scope.currentName);
    });
  }).
  controller('Character', function($scope) {
    $scope.info = {'Frodo': { weapon: 'Sting',
                              race: 'Hobbit'},
                   'Aragorn': { weapon: 'Sword',
                                race: 'Man'},
                   'Legolas': { weapon: 'Bow',
                                race: 'Elf'},
                   'Gimli': { weapon: 'Axe',
                              race: 'Dwarf'}};
    $scope.currentInfo = $scope.info['Frodo'];
    $scope.$on('CharacterChanged', function(event, newCharacter){
      $scope.currentInfo = $scope.info[newCharacter];
    });
    $scope.deleteChar = function() {
      delete $scope.info[$scope.currentName];
      $scope.$emit('CharacterDeleted', $scope.currentName);
    };
  });
```

```
<!doctype html>
<html ng-app="myApp">
  <head>
    <title>AngularJS Scope Events</title>
    <style>
      span{
        padding: 3px; border: 3px ridge;
        cursor: pointer; width: 100px; display: inline-block;
        font: bold 18px/22px Georgia; text-align: center;
        color: white; background-color: blue }
      label{
        padding: 2px; margin: 5px 10px; font: 15px bold;
        display: inline-block; width: 50px; text-align: right; }
      .lList {
        vertical-align: top;
        display: inline-block; width: 130px; }
      .cInfo {
        display: inline-block; width: 175px;
        border: 3px blue ridge; padding: 3px; }
    </style>
  </head>
  <body>
    <h2>Custom Events in Nested Controllers</h2>
    <div ng-controller="Characters">
      <div class="lList">
          <span ng-repeat="name in names"
                 ng-click="changeName()">{{name}}
           </span>
      </div>
      <div class="cInfo">
           <div ng-controller="Character">
             <label>Name: </label>{{currentName}}<br>
             <label>Race: </label>{{currentInfo.race}}<br>
             <label>Weapon: </label>{{currentInfo.weapon}}<br>
             <span ng-click="deleteChar()">Delete</span>
           </div>
      </div>
    </div>
    <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
    <script src="js/scope_events.js"></script>
  </body>
</html>
```

```
$http.get('/myUrl');
$http({method: 'GET', url:'/myUrl'});
```

```
$http({method: 'GET', url: '/myUrl'}).
  success(function(data, status, headers, config) {
    // handle success
  }).
error(function(data, status, headers, config) {
  // handle failure
});
```

```
./service_server.js
./lesson28/service_http.html
./lesson28/js/service_http.js
```

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use('/', express.static('./'));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
function initStore(){
  var items = ['eggs', 'toast', 'bacon', 'juice'];
  var storeObj = {};
  for (var itemIDX in items){
    storeObj[items[itemIDX]] =
      Math.floor(Math.random() * 5 + 1);
  }
  return storeObj;
}
var storeItems = initStore();
app.get('/reset/data', function(req, res){
  storeItems = initStore();
  res.json(storeItems);
});
app.post('/buy/item', function(req, res){
  if (storeItems[req.body.item] > 0){
    storeItems[req.body.item] =
      storeItems[req.body.item] - 1;
    res.json(storeItems);
  }else {
    res.json(400, { msg: 'Sorry ' + req.body.item +
                         ' is out of stock.' });
  }
});
app.listen(80);
```

```
angular.module('myApp', []).
  controller('myController', ['$scope', '$http',
                              function($scope, $http) {
    $scope.storeItems = {};
    $scope.kitchenItems = {};
    $scope.status = "";
    $scope.resetStore = function(){
      $scope.status = "";
      $http.get('/reset/data')
            .success(function(data, status, headers, config) {
               $scope.storeItems = data;
             })
            .error(function(data, status, headers, config) {
               $scope.status = data;
             });
    };
    $scope.buyItem = function(buyItem){
      $http.post('/buy/item', {item:buyItem})
            .success(function(data, status, headers, config) {
               $scope.storeItems = data;
               if($scope.kitchenItems.hasOwnProperty(buyItem)){
                 $scope.kitchenItems[buyItem] += 1;
               } else {
                 $scope.kitchenItems[buyItem] = 1;
               }
               $scope.status = "Purchased " + buyItem;
             })
            .error(function(data, status, headers, config) {
               $scope.status = data.msg;
             });
    };
    $scope.useItem = function(useItem){
      if($scope.kitchenItems[useItem] > 0){
        $scope.kitchenItems[useItem] -= 1;
      }
    };
  }]);
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS $http Service</title>
  <style>
    span {
      color:red; cursor: pointer; }
    .myList {
      display: inline-block; width: 200px;
      vertical-align: top; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h2>GET and POST Using $http Service</h2>
    <input type="button" ng-click="resetStore()"
           value="Restock Store"/>
    {{status}}
    <hr>
    <div class="myList">
         <h3>The Store</h3>
         <div ng-repeat="(item, count) in storeItems">
           {{item}} ({{count}})
           [<span ng-click="buyItem(item)">buy</span>]
         </div>
    </div>
    <div class="myList">
      <h3>My Kitchen</h3>
      <div ng-repeat="(item, count) in kitchenItems">
        {{item}} ({{count}})
        [<span ng-click="useItem(item)">use</span>]
      </div>
    </div>
  </div>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
  <script src="js/service_http.js"></script>
</body>
</html>
```

```
var app = angular.module('myApp', []);
app.factory('MyCache', function($cacheFactory) {
  return $cacheFactory('myCache', {capacity:5});
});
app.controller('myController', ['$scope', 'MyCache',
                                function($scope, cache) {
    cache.put('myValue', 55);
  }]);
app.controller('myController2', ['$scope', 'MyCache',
                                 function($scope, cache) {
  $scope.value = cache.get('myValue');
}]);
```

```
var app = angular.module('myApp', []);
app.controller('myController', ['$scope', '$window',
                                function($scope, window) {
    window.alert("Your Screen is: \n" +
        window.screen.availWidth + "X" + window.screen.availHeight);
  }]);
```

```
var cookie = $cookies.appCookie;
$cookies.appCookie = 'New Value';
```

```
<script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
<script src="http://code.angularjs.org/1.3.0/angular-cookies.min.js"></script>
```

```
var app = angular.module('myApp', ['ngCookies']);
```

```
app.controller('myController', ['$scope', '$cookieStore',
                                function($scope, cookieStore) {
}]);
```

```
app.controller('myController', ['$scope', '$cookieStore',
                                 function($scope, cookieStore) {
```

```
  $scope.setCookie = function(){
    if ($scope.favCookie === 'None'){
      cookieStore.remove('myAppCookie');
    }else{
      cookieStore.put('myAppCookie', {flavor:$scope.favCookie});
    }
    $scope.myFavCookie = cookieStore.get('myAppCookie');
  };
  $scope.initCookieValue = function(){
    var cookie = cookieStore.get('myAppCookie');
    if (cookie){
      $scope.favCookie = cookie.flavor;
    } else {
      $scope.favCookie = 'None';
    }
    $scope.myFavCookie = $scope.favCookie;
  };
  $scope.initCookieValue();
```

```
var app = angular.module('myApp', ['ngCookies']);
app.controller('myController', ['$scope', '$cookieStore',
                                function($scope, cookieStore) {
    $scope.setCookie = function(){
      if ($scope.favCookie === 'None'){
        cookieStore.remove('myAppCookie');
      }else{
        cookieStore.put('myAppCookie', {flavor:$scope.favCookie});
      }
      $scope.myFavCookie = cookieStore.get('myAppCookie');
    };
    $scope.initCookieValue = function(){
      var cookie = cookieStore.get('myAppCookie');
      if (cookie){
        $scope.favCookie = cookie.flavor;
      } else {
        $scope.favCookie = 'None';
      }
      $scope.myFavCookie = $scope.favCookie;
    };
    $scope.initCookieValue();
  }]);
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS $cookie Service</title>
</head>
<body>
  <div ng-controller="myController">
    <h3>Favorite Cookie:</h3>
    <input type="radio" value="Chocolate Chip" ng-model="favCookie"
           ng-change="setCookie()">Chocolate Chip</input><br>
    <input type="radio" value="Oatmeal" ng-model="favCookie"
           ng-change="setCookie()">Oatmeal</input><br>
    <input type="radio" value="Frosted" ng-model="favCookie"
           ng-change="setCookie()">Frosted</input><br>
    <input type="radio" value="None" ng-model="favCookie"
           ng-change="setCookie()">None</input>
    <hr>Cookies: {{myFavCookie}}
  </div>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
  <script src="http://code.angularjs.org/1.3.0/angular-cookies.min.js"></script>
  <script src="js/service_cookie.js"></script>
</body>
</html>
```

```
$interval(callback, delay, [count], [invokeApply]);
$timeout(callback, delay, [invokeApply]);
```

```
var myInterval = $interval(function(){$scope.seconds++;}, 1000, 10, true);
. . .
$interval.cancel(myInterval);
```

```
$scope.$on('$destroy', function(){
  $scope.cancel(myInterval);
});
```

```
.img-fade-add, .img-fade-remove {
  -webkit-transition:all ease 2s;
  -moz-transition:all ease 2s;
  -o-transition:all ease 2s;
  transition:all ease 2s;
}
.img-fade, .img-fade-add.img-fade-add-active {
   opacity:.1;
}
```

<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>

```
var app = angular.module('myApp', ['ngAnimate']);
```

```
animate(cssProperties, [duration], [easing], [callback])
```

```
app.animation('.fadeClass', function() {
  return {
    addClass : function(element, className, done) {
      jQuery(element).animate({ opacity: 0}, 3000);
    },
  };
});
```

```
var app = angular.module('myApp', ['ngAnimate']);
```

```
app.animation('.fadeOut', function() {
  return {
    enter : function(element, parentElement, afterElement, doneCallback) {},
    leave : function(element, doneCallback) {},
    move : function(element, parentElement, afterElement, doneCallback) {},
    addClass : function(element, className, done) {
      jQuery(element).animate({ opacity: 0}, 3000);
    },
    removeClass : function(element, className, done) {
      jQuery(element).animate({ opacity: 1}, 3000);
    }
  };
```

```
.shrink-add, .grow-add {
  -webkit-transition:all ease 2.5s;
  -moz-transition:all ease 2.5s;
  -o-transition:all ease 2.5s;
  transition:all ease 2.5s;
}
.shrink,
.shrink-add.shrink-add-active {
  width:100px;
}
.start-class,
.grow,
.grow-add.grow-add-active {
  width:400px;
}
```

```
var app = angular.module('myApp', ['ngAnimate']);
app.controller('myController', function($scope ) {
  $scope.myImgClass = 'start-class';
  });
app.animation('.fadeOut', function() {
  return {
    enter : function(element, parentElement, afterElement, doneCallback) {},
    leave : function(element, doneCallback) {},
    move : function(element, parentElement, afterElement, doneCallback) {},
    addClass : function(element, className, done) {
      jQuery(element).animate({ opacity: 0}, 3000);
    },
    removeClass : function(element, className, done) {
      jQuery(element).animate({ opacity: 1}, 3000);
    }
  };
});
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS $animate Service</title>
  <link rel="stylesheet" href="css/animate.css">
</head>
<body>
  <div ng-controller="myController">
    <h3>AngularJS Image Animation:</h3>
    <input type="button"
           ng-click="myImgClass='fadeOut'" value="Fade Out"/>
    <input type="button"
           ng-click="myImgClass=''" value="Fade In"/>
    <input type="button"
           ng-click="myImgClass='shrink'" value="Small"/>
    <input type="button"
           ng-click="myImgClass='grow'" value="Big"/>
    <hr>
    <img ng-class="myImgClass" src="/images/canyon.jpg" />
  </div>
  <script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
  <script
➥src="http://code.angularjs.org/1.3.0/angular-animate.min.js"></script>
  <script src="js/service_animate.js"></script>
</body>
</html>
```

```
.shrink-add, .grow-add {
  -webkit-transition:all ease 2.5s;
  -moz-transition:all ease 2.5s;
  -o-transition:all ease 2.5s;
  transition:all ease 2.5s;
}
.shrink,
.shrink-add.shrink-add-active {
  width:100px;
}
.start-class,
.grow,
.grow-add.grow-add-active {
  width:400px;
}
img{
  width:100px;
}
```

```
app.controller('myController', ['$scope', '$location',
                                function($scope, location) {
    . . .
  }]);
```

```
app.controller('myController', ['$scope', '$location',
                                function($scope, location) {
```

```
$scope.changePath = function(){
  location.path("/new/path");
  $scope.updateLocationInfo();
};
```

```
$scope.changeHash = function(){
  location.hash("newHash");
  $scope.updateLocationInfo();
};
```

```
$scope.changeSearch = function(){
  location.search("p1", "newA");
  $scope.updateLocationInfo();
};
```

```
var app = angular.module('myApp', []);
app.controller('myController', ['$scope', '$location',
                                function($scope, location) {
  $scope.updateLocationInfo = function() {
    $scope.url = location.url();
    $scope.absUrl = location.absUrl();
    $scope.host = location.host();
    $scope.port = location.port();
    $scope.protocol = location.protocol();
    $scope.path = location.path();
    $scope.search = location.search();
    $scope.hash = location.hash();
  };
  $scope.changePath = function(){
    location.path("/new/path");
    $scope.updateLocationInfo();
  };
  $scope.changeHash = function(){
    location.hash("newHash");
    $scope.updateLocationInfo();
  };
  $scope.changeSearch = function(){
    location.search("p1", "newA");
    $scope.updateLocationInfo();
  };
  $scope.updateLocationInfo();
}]);
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS $location Service</title>
  <style>
    span {
      color: red; text-decoration: underline;
      cursor: pointer; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h3>Location Service:</h3>
    [<span ng-click="changePath()">Change Path</span>]
    [<span ng-click="changeHash()">Change Hash</span>]
    [<span ng-click="changeSearch()">Change Search</span>]
    <hr>
    <h4>URL Info</h4>
    url: {{url}}<br>
    absUrl: {{absUrl}}<br>
    host: {{host}}<br>
    port: {{port}}<br>
    protocol: {{protocol}}<br>
    path: {{path}}<br>
    search: {{search}}<br>
    hash: {{hash}}<br>
  </div>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
  <script src="js/service_location.js"></script>
</body>
</html>
```

```
function makeDeferredRequest(){
  var deferred = $q.defer();
  return deferred.promise;
}
```

```
promise.then(successCallback, [errorCallback], [notifyCallback])
```

```
var promise = makeDeferredRequest();
promise.then(
  function successCallback(value){
    //handle success
  },
  function errorCallback(value){
    //handle error
  },
  function notifyCallback(value){
    //handle notify
  },
var app = angular.module('myApp', []);
app.value('myValue', {color:'blue', value:'17'});
```

```
var app = angular.module('myApp', []);
app.constant('myConst', "Constant String");
```

```
factory(name, factoryProvider)
```

```
var app = angular.module('myApp', []);
app.constant('myConst', 10);
app.factory('multiplier', ['myConst', function (myConst) {
  return function(value) { return value + myConst; };
}]);
```

```
var app = angular.module('myApp', []);
app.constant('myConst', 10);
function ConstMathObj(myConst) {
  this.add = function(value){ return value + myConst; };
  this.multiply = function(value){ return value * myConst; };
}
app.service('constMath', ['myConst', ConstMathObj] );
app.value('censorWords', ["can't", "quit", "fail"]);
app.constant('repString', "*****");
app.factory('censorF', ['censorWords', 'repString',
                        function (cWords, repString) {
  return function(inString) {
    var outString = inString;
    for(i in cWords){
      var regex = new RegExp(cWords[i], "ig");
      outString = outString.replace(regex, repString);
    }
    return outString;
  };
}]);
```

```
function CensorObj(cWords, repString) {
  this.censor = function(inString){
    var outString = inString;
    for(i in cWords){
      var regex = new RegExp(cWords[i], "ig");
      outString = outString.replace(regex, repString);
    }
    return outString;
  };
  this.censoredWords = function(){
    return cWords;
  };
}
app.service('censorS', ['censorWords', 'repString', CensorObj]);
```

```
    $scope.censoredByFactory = censorF(newValue);
    $scope.censoredByService = censorS.censor(newValue);
```

```
var app = angular.module('myApp', []);
app.value('censorWords', ["can't", "quit", "fail"]);
app.constant('repString', "****");
app.factory('censorF', ['censorWords', 'repString',
                        function (cWords, repString) {
  return function(inString) {
    var outString = inString;
    for(i in cWords){
      var regex = new RegExp(cWords[i], "ig");
      outString = outString.replace(regex, repString);
    }
    return outString;
  };
}]);
function CensorObj(cWords, repString) {
  this.censor = function(inString){
    var outString = inString;
    for(i in cWords){
      var regex = new RegExp(cWords[i], "ig");
      outString = outString.replace(regex, repString);
    }
    return outString;
  };
  this.censoredWords = function(){
    return cWords;
  };
}
app.service('censorS', ['censorWords', 'repString', CensorObj]);
app.controller('myController', ['$scope', 'censorF', 'censorS',
                                function($scope, censorF, censorS) {
  $scope.censoredWords = censorS.censoredWords();
  $scope.inPhrase = "";
  $scope.censoredByFactory = censorF("");
  $scope.censoredByService = censorS.censor("");;
  $scope.$watch('inPhrase', function(newValue, oldValue){
    $scope.censoredByFactory = censorF(newValue);
    $scope.censoredByService = censorS.censor(newValue);
  });
}]);
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Censor Service</title>
  <style>
    p { color: red; margin-left: 15px; }
    input { width: 250px; }
  </style>
</head>
<body>
  <div ng-controller="myController">
    <h3>Custom Censor Service:</h3>
    Censored Words:<br>
    <p>{{censoredWords|json}}</p>
    <hr>
    Enter Phrase:<br>
    <input type="text" ng-model="inPhrase" /><hr>
    Filtered by Factory:
    <p>{{censoredByFactory}}</p>
    Filtered by Service:
    <p>{{censoredByService}}</p>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/service_custom_censor.js"></script>
</body>
</html>
```

```
var app = angular.module('myApp', []);
function TimeService() {
  var cities = { 'Los Angeles': -8,
                 'New York': -5,
                 'London': 0,
                 'Paris': 1,
                 'Tokyo': 9 };
  this.getTZDate = function(city){
    var localDate = new Date();
    var utcTime = localDate.getTime() +
                  localDate.getTimezoneOffset() *
                  60*1000;
    return new Date(utcTime +
                     (60*60*1000 *
                      cities[city]));
  };
  this.getCities = function(){
    var cList = [];
    for (var key in cities){
      cList.push(key);
    }
    return cList;
  };
}
app.service('TimeService', [TimeService]);
app.controller('LAController', ['$scope', 'TimeService',
                                function($scope, timeS) {
  $scope.myTime = timeS.getTZDate("Los Angeles").toLocaleTimeString();
}]);
app.controller('NYController', ['$scope', 'TimeService',
                                function($scope, timeS) {
  $scope.myTime = timeS.getTZDate("New York").toLocaleTimeString();
}]);
app.controller('LondonController', ['$scope', 'TimeService',
                                    function($scope, timeS) {
  $scope.myTime = timeS.getTZDate("London").toLocaleTimeString();
}]);
app.controller('TimeController', ['$scope', 'TimeService',
                                  function($scope, timeS) {
  $scope.cities = timeS.getCities();
  $scope.getTime = function(cityName){
    return timeS.getTZDate(cityName).toLocaleTimeString();
  };
}]);
```

```html
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Time Service</title>
  <style>
    span {
      color: lightgreen; background-color: black;
      border: 3px ridge; padding: 2px;
      font: 14px/18px arial, serif; }
  </style>
</head>
<body>
  <h2>Custom Time Service:</h2><hr>
  <div ng-controller="LAController">
    Los Angeles Time:
    <span>{{myTime}}</span>
  </div><hr>
  <div ng-controller="NYController">
    New York Time:
    <span>{{myTime}}</span>
  </div><hr>
  <div ng-controller="LondonController">
    London Time:
    <span>{{myTime}}</span>
  </div><hr>
  <div ng-controller="TimeController">
    All Times:
    <table>
    <tr>
         <th ng-repeat="city in cities">
           {{city}}
         </th>
    </tr>
    <tr>
         <td ng-repeat="city in cities">
           <span>{{getTime(city)}}</span>
         </td>
    </tr>
    </table>
  </div><hr>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/service_custom_time.js"></script>
</body>
</html>
```

```
./service_db_server.js
./lesson28/service_custom_db.html
./lesson28/js/service_custom_db_access.js
./lesson28/js/service_custom_db.js
```

```
var express = require('express');
var bodyParser = require('body-parser');
var app = express();
app.use('/', express.static('./'));
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
var user = {
             first: 'Christopher',
             last: 'Columbus',
             username: 'cc1492',
             title: 'Admiral',
             home: 'Genoa'
           };
var data = [];
function r(min, max){
  var n = Math.floor(Math.random() * (max - min + 1)) + min;
  if (n<10){ return '0' + n; }
  else { return n; }
}
function p(start, end, total, current){
  return Math.floor((end-start)*(current/total)) + start;
}
function d(plusDays){
  var start = new Date(1492, 7, 3);
  var current = new Date(1492, 7, 3);
  current.setDate(start.getDate()+plusDays);
  return current.toDateString();
}
function makeData(){
  var t = 70;
  for (var x=0; x < t; x++){
    var entry = {
      day: d(x),
      time: r(0, 23) + ':' + r(0, 59),
      longitude: p(37, 25, t, x) + '\u00B0 '+ r(0,59) + ' N',
      latitude: p(6, 77, t, x) + '\u00B0 '+ r(0,59) + ' W'
    };
    data.push(entry);
  }
}
```

```
makeData();
app.get('/get/user', function(req, res){
  res.json(user);
});
app.get('/get/data', function(req, res){
  res.json(data);
});
app.post('/set/user', function(req, res){
  console.log(req.body.userData);
  user = req.body.userData;
  res.json({ data: user, status: "User Updated." });
});
app.post('/set/data', function(req, res){
  data = req.body.data;
  res.json({ data: data, status: "Data Updated." });
});
app.listen(80);
```

```
var app = angular.module('dbAccess', []);
function DBAccessObj($http, $q) {
  this.getUserData = function(){
    var deferred = $q.defer();
    $http.get('/get/user')
    .success(function(response, status, headers, config) {
      deferred.resolve(response);
    });
    return deferred.promise;
  };
  this.updateUser = function(userInfo){
    var deferred = $q.defer();
    $http.post('/set/user', { userData: userInfo}).
    success(function(response, status, headers, config) {
      deferred.resolve(response);
    });
    return deferred.promise;
  };
  this.getData = function(){
    var deferred = $q.defer();
    $http.get('/get/data')
    .success(function(response, status, headers, config) {
      deferred.resolve(response);
    });
    return deferred.promise;
  };
  this.updateData = function(data){
    var deferred = $q.defer();
    $http.post('/set/data', { data: data}).
    success(function(response, status, headers, config) {
      deferred.resolve(response);
    });
    return deferred.promise;
  };
}
app.service('DBService', ['$http', '$q', DBAccessObj]);
```

```
var app = angular.module('myApp', ['dbAccess']);
app.controller('myController', ['$scope', 'DBService',
                                function($scope, db) {
  $scope.status = "";
  $scope.getUser = function(){
    db.getUserData().then(function(response){
      $scope.userInfo = response;
      $scope.status = "User Data Retrieve.";
    });
  };
  $scope.getData = function(){
    db.getData().then(function(response){
      $scope.data = response;
      $scope.status = "User Data Retrieve.";
    });
  };
  $scope.updateUser = function(){
    db.updateUser($scope.userInfo).then(function(response){
      $scope.status = response.status;
    });
  };
  $scope.updateData = function(){
    db.updateData($scope.data).then(function(response){
      $scope.status = response.status;
    });
  };
  $scope.getUser();
  $scope.getData();
}]);
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>AngularJS Custom Database Service</title>
     <style>
       label {
         display: inline-block; width: 75px; text-align: right; }
       td, tr {
         width: 125px; text-align: right; }
       p {
         color: red; font: italic 12px/14px; margin: 0px;}
       h3 {
         margin: 5px; }
     </style>
</head>
<body>
  <h2>Custom Database Service:</h2>
  <div ng-controller="myController">
    <h3>User Info:</h3>
    <label>First:</label>
      <input type="text" ng-model="userInfo.first" /><br>
    <label>Last:</label>
      <input type="text" ng-model="userInfo.last" /><br>
    <label>Username:</label>
      <input type="text" ng-model="userInfo.username" /><br>
    <label>Title:</label>
      <input type="text" ng-model="userInfo.title" /><br>
    <label>Home:</label>
      <input type="text" ng-model="userInfo.home" /><br>
    <input type= button ng-click="updateUser()" value="Update User" />
    <input type= button ng-click="getUser()" value="Refresh User Info" />
    <hr>
    <p>{{status}}</p>
    <hr>
    <h3>Data:</h3>
    <input type= button ng-click="updateData()" value="Update Data" />
    <input type= button ng-click="getData()" value="Refresh Data Table" /><br>
    <table>
      <tr><th>Day</th><th>Time</th><th>Latitude</th><th>Longitude</th></tr>
      <tr ng-repeat="datum in data">
        <th>{{datum.day}}</th>
        <td><input type="text" ng-model="datum.time" /></td>
        <td><input type="text" ng-model="datum.latitude" /></td>
        <td><input type="text" ng-model="datum.longitude" /></td>
      </tr>
    </table>
    <hr>
  </div>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/service_custom_db_access.js"></script>
  <script src="js/service_custom_db.js"></script>
</body>
</html>
```

```
var app = angular.module('myApp', []);
app.directive('myTabs', function() {
  return {
    restrict: 'E',
    transclude: true,
    scope: {},
    controller: function($scope) {
      var panes = $scope.panes = [];
      $scope.select = function(pane) {
        angular.forEach(panes, function(pane) {
          pane.selected = false;
        });
        pane.selected = true;
      };
      this.addPane = function(pane) {
        if (panes.length == 0) {
          $scope.select(pane);
        }
        panes.push(pane);
      };
    },
    templateUrl: 'tabs.html'
  };
});
app.directive('myPane', function() {
  return { require: '^myTabs', restrict: 'E',
    templateUrl: 'pane.html',
    transclude: true, scope: { title: '@' },
    link: function(scope, element, attrs, tabsCtrl) {
      tabsCtrl.addPane(scope);
    }
  };
});
```

```
<div class="tabbable">
  <div class="tabs">
    <span class="tab" ng-repeat="pane in panes"
        ng-class="{activeTab:pane.selected}"
        ng-click="select(pane)">{{pane.title}}
    </span>
  </div>
  <div class="tabcontent" ng-transclude></div>
</div>?
```

```html
<!doctype html>
<html ng-app="myApp">
<head>
  <title>Tab and Tab Pane Directives</title>
  <style>
    .tab{
      display:inline-block; width:100px;
      border-radius: .5em .5em 0 0; border:1px solid black;
      text-align:center; font: 15px/28px Helvetica, sans-serif;
      background-image: linear-gradient(#CCCCCC, #EEEEEE);
      cursor: pointer; }
    .activeTab{
      border-bottom: none;
      background-image: linear-gradient(#66CCFF, #CCFFFF); }
    .pane{
      border:1px solid black; background-color: #CCFFFF;
      height:300px; width:400px;
      padding:10px;  margin-top:-2px;
      overflow: scroll; }
  </style>
</head>
<body>
  <h2>AngularJS Custom Tabs</h2>
  <my-tabs>
    <my-pane title="Canyon">
      <img src="/images/canyon.jpg" />
    </my-pane>
    <my-pane title="Lake">
      <img src="/images/lake.jpg" />
    </my-pane>
    <my-pane title="Sunset">
      <img src="/images/jump.jpg" />
    </my-pane>
  </my-tabs>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
  <script src="js/tabbable.js"></script>
</body>
</html>
```

```
var app = angular.module('myApp', []);
app.controller('myController', function($scope) {
  $scope.dragStatus = "none";
  $scope.dropStatus = "none";
  $scope.dropValue = "";
})
.directive('dragit', function($document, $window) {
  function makeDraggable(scope, element, attr) {
    angular.element(element).attr("draggable", "true");
    element.on('dragstart', function(event) {
      element.addClass('dragItem');
      scope.$apply(function(){
        scope.dragStatus = "Dragging " + element.html();
        scope.dropValue = element.html();
      });
      event.dataTransfer.setData('Text', element.html());
    });
    element.on('drag', function(event) {
      scope.$apply(function(){
        scope.dragStatus = "X: " + event.pageX +
                           " Y: " + event.pageY;
      });
    });
    element.on('dragend', function(event) {
      event.preventDefault();
      element.removeClass('dragItem');
    });
  }
  return {
    link: makeDraggable
  };
})
.directive('dropit', function($document, $window) {
  return {
    restrict: 'E',
    link: function makeDroppable(scope, element, attr){
      element.on('dragover', function(event) {
        event.preventDefault();
        scope.$apply(function(){
          scope.dropStatus = "Drag Over";
        });
      });
      element.on('dragleave', function(event) {
        event.preventDefault();
        element.removeClass('dropItem');
        scope.$apply(function(){
          scope.dropStatus = "Drag Leave";
        });
      });
      element.on('dragenter', function(event) {
        event.preventDefault();
        element.addClass('dropItem');
        scope.$apply(function(){
          scope.dropStatus = "Drag Enter";
        });
      });
      element.on('drop', function(event) {
        event.preventDefault();
        element.removeClass('dropItem');
        scope.$apply(function(){
          element.append('<p>' +
              event.dataTransfer.getData('Text') + '</p>');
          scope.dropStatus = "Dropped " + scope.dropValue;
        });
      });
    }
  };
});
```

```
<!doctype html>
<html ng-app="myApp">
<head>
  <title>HTML5 Draggable and Droppable Directives</title>
  <style>
    dropit, img, p{
      vertical-align: top; text-align: center;
            width: 100px;
            display: inline-block;
          }
          p {
            color: white; background-color: black;
            font: bold 14px/16px arial;
      margin: 0px; width: 96px;
      border: 2px ridge grey;
      background: linear-gradient(#888888, #000000);
          }
          span{
            display:inline-block; width: 100px;
            font: 16px/18px Georgia, serif; text-align: center;
            padding: 2px;
            background: linear-gradient(#FFFFFF, #888888);
          }
          .dragItem {
            color: red;
      opacity: .5;
          }
    .dropItem {
      border: 3px solid red;
      opacity: .5;
    }
    #dragItems {
      width: 400px;
    }
  </style>
</head>
<body>
  <h2>HTML5 Drag and Drop Components</h2>
  <div ng-controller="myController">
    Drag Status: {{dragStatus}}<br>
    Drop Status: {{dropStatus}}
    <hr>
    <div id="dragItems">
          <span dragit>Nature</span>
          <span dragit>Landscape</span>
          <span dragit>Flora</span>
          <span dragit>Sunset</span>
          <span dragit>Arch</span>
          <span dragit>Beauty</span>
          <span dragit>Inspiring</span>
      <span dragit>Summer</span>
      <span dragit>Fun</span>
    </div>
    <hr>
    <dropit><img src="/images/arch.jpg" /></dropit>
    <dropit><img src="/images/flower.jpg" /></dropit>
    <dropit><img src="/images/cliff.jpg" /></dropit>
    <dropit><img src="/images/jump.jpg" /></dropit>
  </div>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
  <script src="js/dragdrop.js"></script>
</body>
</html>
```

```
angular.module('myApp', [])
.controller('myController', ['$scope', function($scope) {
}])
.directive('zoomit', function() {
  return {
    restrict: 'E',
    scope: { src: '@'},
    controller: function($scope) {
        $scope.zInfo = {
            "background-image": "url(" + $scope.src + ")",
            "background-position": "top right"
        };
        $scope.imageClick= function(event){
          event.preventDefault();
          //Using full jQuery to get offset, width and height
          var elem = angular.element(event.target);
          var posX = Math.ceil((event.pageX - elem.offset().left) /
                               elem.width() * 100);
          var posY = Math.ceil((event.pageY - elem.offset().top) /
                               elem.height() * 100);
          $scope.pos = posX + "% " + posY + "%";
          $scope.zInfo["background-position"] = posX + "% " +
                                                posY + "%";
        };
      },
    link: function(scope, element, attrs) {
      },
    templateUrl: 'zoomit.html'
  };
});
```

```
<div>
  <img src="{{src}}"
       ng-click="imageClick($event)"/>
  <div class="zoombox"
       ng-style="zInfo"></div>
</div>
```

```
<!DOCTYPE html>
<html  ng-app="myApp">
  <head>
    <title>Magnify</title>
    <style>
      .zoombox {
        display: inline-block;
        border: 3px ridge black;
        width: 100px; height: 100px; }
      img {
        height: 200px;
        vertical-align: top; }
    </style>
  </head>
  <body>
  <h2>Image Zoom Window</h2>
  <div ng-controller="myController">
    <zoomit  src="/images/flower.jpg"></zoomit>
    <hr>
    <zoomit  src="/images/tiger.jpg"></zoomit>
  </div>
  </body>
  <script
src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
  <script src="js/zooming.js"></script>
</html>
```

```
angular.module('myApp', [])
.controller('myController', ['$scope', function($scope) {
  $scope.items = [1,2,3,4,5];
}])
.directive('expandList', function() {
  return {
    restrict: 'E',
    transclude: true,
    scope: {title: '@', listWidth: '@exwidth'},
    controller: function($scope) {
      $scope.collapsed = false;
      $scope.expandHandle = "-";
      items = $scope.items = [];
      $scope.collapse = function() {
        if ($scope.collapsed){
          $scope.collapsed = false;
          $scope.expandHandle = "-";
        } else {
          $scope.collapsed = true;
          $scope.expandHandle = "+";
        }
        angular.forEach($scope.items, function(item) {
          item.myHide = $scope.collapsed;
        });
      };
      this.addItem = function(item) {
        item.myStyle.width = $scope.listWidth;
        items.push(item);
        item.myHide=false;
      };
    },
    link: function(scope, element, attrs, expandCtrl) {
      element.css("display", "inline-block");
      element.css("width", scope.listWidth);
    },
    templateUrl: 'expand_list.html',
  };
})
```

```
.directive('expandItem', function() {
  return {
    require: '^expandList',
    restrict: 'E',
    transclude: true,
    scope: {},
    controller: function($scope){
        $scope.myHide = false;
        $scope.myStyle = { width: "100px", "display": "inline-block" };
      },
    link: function(scope, element, attrs, expandCtrl) {
      expandCtrl.addItem(scope);
    },
    templateUrl: 'expand_item.html',
  };
});
```

```
<div>
     <div class="expand-header">
          <span class="expand-button"
                ng-click="collapse()">{{expandHandle}}</span>
          {{title}}
     </div>
     <div ng-transclude></div>
</div>
```

```
<!DOCTYPE html>
<html  ng-app="myApp">
  <head>
    <title>Expandable and Collapsible Lists</title>
    <style>
      * { vertical-align: top; }
      expand-list{
        border: 2px ridge black; }
      .expand-header{
        text-align: center;
        font: bold 16px/24px arial;
         background-image: linear-gradient(#CCCCCC, #EEEEEE);
        }
      .expand-button{
        float: left; padding: 2px 4px;
        font: bold 22px/16px courier;
        color: white; background-color: black;
        cursor: pointer;
        border: 3px groove grey; }
      .expand-item {
        border: 1px ridge black;}
      p { margin: 0px; padding: 2px;}
      label { display: inline-block; width: 80px; padding: 2px; }
      .small { width: 100px; padding: 2px; }
      .large { width: 300px; }
  </style>
</head>
<body>
<h2>Expandable and Collapsible Lists</h2>
<hr>
<div ng-controller="myController">
  <expand-list title="Companion" exwidth="120px">
    <expand-item>Rose</expand-item>
    <expand-item>Donna</expand-item>
    <expand-item>Martha</expand-item>
    <expand-item>Amy</expand-item>
    <expand-item>Rory</expand-item>
  </expand-list>
  <expand-list title="Form" exwidth="280px">
    <expand-item>
      <label>Name</label>
      <input type="text" /><br>
      <label>Phone</label>
      <input type="text" /><br>
      <label>Address</label>
      <input type="text" /><br>
      <label>Comment</label>
      <textarea type="text"></textarea>
    </expand-item>
  </expand-list>
  <hr>
```

```
    <expand-list title="Mixed List" exwidth="300px">
      <expand-item>Text Item</expand-item>
      <expand-item><p>I think therefore I am.</p></expand-item>
      <expand-item>
        <img class="small" src="/images/jump.jpg" />Sunset
      </expand-item>
      <expand-item>
        <ul>
          <li>AngularJS</li>
          <li>jQuery</li>
          <li>JavaScript</li>
        </ul>
      </expand-item>
    </expand-list>
    <expand-list title="Image" exwidth="300px">
      <expand-item>
        <img class="large" src="/images/falls.jpg" />
      </expand-item>
    </expand-list>
  </div>
  </body>
  <script src="http://code.angularjs.org/1.3.0/angular.min.js"></script>
  <script src="js/expand.js"></script>
</html>
```

```
angular.module('myApp', [])
.controller('myController', ['$scope', function($scope) {
  $scope.stars = [1,2,3,4,5];
  $scope.items = [
      {
        description: "Mysty Mountains",
        img: "/images/misty_mountains.jpg",
        rating: 3},
      {
        description: "Wheel",
        img: "/images/wheel.jpg",
        rating: 4},
      {
        description: "Pool",
        img: "/images/pool.jpg",
        rating: 4}
    ];
  $scope.adjustRating = function(item, value){
    item.rating = value;
  };
}]);
```

```html
<!DOCTYPE html>
<html  ng-app="myApp">
  <head>
    <title>Ratings</title>
    <style>
      img {
        width: 200px; }
      .star {
        display: inline-block;
        width: 15px;
        background-image: url("/images/star.png");
        background-repeat: no-repeat;
        background-size: 15px 15px;
      }
      .empty {
        display: inline-block;
        width: 15px;
        background-image: url("/images/empty.png");
        background-repeat: no-repeat;
        background-size: 15px 15px;
      }
    </style>
  </head>
  <body>
  <h2>Images With Ratings</h2>
  <hr>
  <div ng-controller="myController">
    <div ng-repeat="item in items">
      <img ng-src="{{item.img}}" />
      {{item.description}}<br>
      Rating: {{item.rating}} stars<br>
      <span ng-repeat="idx in stars"
            ng-class=
              "{true: 'star', false: 'empty'}[idx <= item.rating]"
            ng-click="adjustRating(item, idx)">&nbsp;
      </span>
      <hr>
    </div>
  </div>
  </body>
  <script src="http://code.angularjs.org/1.3.8/angular.min.js"></script>
  <script src="js/rating.js"></script>
</html>
```