jQuery

CD-ROM
Included!

OSWALD CAMPESATO

jQuery

CD-ROM
Included!

OSWALD CAMPESATO

jQuery
Pocket Primer

LICENSE, DISCLAIMER OF LIABILITY, AND LIMITED WARRANTY

By purchasing or using this book (the "Work"), you agree that this license grants permission to use the contents contained herein, but does not give you the right of ownership to any of the textual content in the book or ownership to any of the information or products contained in it. *This license does not permit uploading of the Work onto the Internet or on a network (of any kind) without the written consent of the Publisher.* Duplication or dissemination of any text, code, simulations, images, etc. contained herein is limited to and subject to licensing terms for the respective products, and permission must be obtained from the Publisher or the owner of the content, etc., in order to reproduce or network any portion of the textual material (in any media) that is contained in the Work.

Mercury Learning and Information ("MLI" or "the Publisher") and anyone involved in the creation, writing, or production of the companion disc, accompanying algorithms, code, or computer programs ("the software"), and any accompanying Web site or software of the Work, cannot and do not warrant the performance or results that might be obtained by using the contents of the Work. The author, developers, and the Publisher have used their best efforts to insure the accuracy and functionality of the textual material and/or programs contained in this package; we, however, make no warranty of any kind, express or implied, regarding the performance of these contents or programs. The Work is sold "as is" without warranty (except for defective materials used in manufacturing the book or due to faulty workmanship).

The author, developers, and the publisher of any accompanying content, and anyone involved in the composition, production, and manufacturing of this work will not be liable for damages of any kind arising out of the use of (or the inability to use) the algorithms, source code, computer programs, or textual material contained in this publication. This includes, but is not limited to, loss of revenue or profit, or other incidental, physical, or consequential damages arising out of the use of this Work.

The sole remedy in the event of a claim of any kind is expressly limited to replacement of the book, and only at the discretion of the Publisher. The use of "implied warranty" and certain "exclusions" vary from state to state, and might not apply to the purchaser of this product.

jQuery
Pocket Primer

Oswald Campesato

MERCURY LEARNING AND INFORMATION
Dulles, Virginia
Boston, Massachusetts
New Delhi

Publisher: David Pallai
MERCURY LEARNING AND INFORMATION
22841 Quicksilver Drive
Dulles, VA 20166
info@merclearning.com
www.merclearning.com
(800) 232-0223

O. Campesato. *jQuery Pocket Primer.*
ISBN: 978-1-938549-14-4

Library of Congress Control Number: 2013944477

1516321 Printed in the United States of America
This book is printed on acid-free paper.

I'd like to dedicate this book to my parents –
may this bring joy and happiness into their lives.

Contents

PREFACE

This book endeavors to provide you with as much up-to-date information as possible regarding jQuery that can be reasonably included in a book consisting of roughly 200 pages. You need some familiarity with HTML Web pages and JavaScript, but no prior knowledge of jQuery is required.

Which Version of jQuery is for this Book?

The code samples in this book were initially written using jQuery 1.7.1. These code samples were upgraded to jQuery 2.0 beta, along with jQuery Migrate plugin (version 1.9.1) to ensure that the code samples do not contain any deprecated jQuery methods.

There are two benefits to this approach. First, you can use these HTML Web pages with jQuery 2.0 beta and be assured that they do not contain any deprecated jQuery code. In addition, if you need to use a version of jQuery prior to version 2.0 (perhaps due to an existing code base), these HTML Web pages work correctly without modification.

What About jQuery 2.0?

The current production version of jQuery is version 1.9, and jQuery 2.0 beta is also available. As this book goes to print, jQuery 2.0 will probably become available as well. The key point to remember is that jQuery 2.0 removes support for IE 6, IE 7, and IE 8, and also deprecates some jQuery methods that are available in earlier versions of jQuery. Fortunately, version 2.0 will be backward compliant with jQuery 1.9.

Which Version of jQuery Should Readers Use?

This is an important question because there are at least five active versions of jQuery in HTML Web pages (back to version 1.6.x). The answer is simple: strive to write code that is compliant with jQuery 2.0.

There are two things that you can do to achieve jQuery 2.0 compliance. First, make sure that you do not use the jQuery methods that have been desupported, which are available here (along with other changes):

http://jquery.com/upgrade-guide/1.9/

Second, use the jQuery Migrate plugin, which you can use with either 1.9 or 2.0 to detect deprecated and removed features, or to restore old features for cases where you need old code to run with new jQuery. The plugin and the messages it generates are documented in the project README.

You can use the plugin simply by including it after your jQuery file and check for warning messages regarding deprecated code. For example, if you plan to use jQuery 1.9 in your HTML Web pages, include the following code snippet:

```
<script src="http://code.jquery.com/jquery-1.9.0.js"></script>

<script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
```

If you plan to use jQuery 2.0 in your HTML Web pages, include the following code snippet:

```
<script src="http://code.jquery.com/jquery-2.0.0b1.js"></script>

<script src="http://code.jquery.com/jquery-migrate-1.0.0.js"> </script>
```

When you add the preceding code snippet to an HTML Web page and then launch that Web page in a WebKit-based browser, you will see the following message in the Inspector if there are no errors:

```
JQMIGRATE: Logging is active
```

You can find additional details on the jQuery blog:

http://blog.jquery.com/

How Do I Actually Find the Deprecated Code?

Launch the HTML Web page in question in a WebKit-based browser and open the Web Inspector to check for errors or warnings. Unfortunately, the jQuery Migrate plugin can generate warning messages that are unintuitive. For example, consider the following code snippet in JQDragAndDrop1.html in Chapter 5:

```
<!—

<script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8.9/jquery-ui.min.js">

</script>

—>

<script src="http://ajax.googleapis.com/ajax/libs/jqueryui/1.10.1/jquery-ui.min.js">

</script>
```

You must use the second <script> element because the first <script> element does not work correctly. Listing 1 displays the output in Web Inspector when you launch the HTML Web page and you attempt to drag-and-drop any of the images.

Figure 1 An Error Message from jQuery Migration Plugin in Web Inspector.

If you launch an HTML Web page and you find “cryptic” error messages, check the jQuery Website for the latest version of the CDN-based JavaScript and CSS files and update your Web page accordingly. If this does not resolve the issue, perform a Google search to see if you can find a link with a solution for the error.

ABOUT THE TECHNICAL EDITOR

Richard Clark, M.A. (@rdclark) is an experienced software developer and instructor for Kaazing Corporation. He has taught for Apple and Hewlett-Packard, written immersive simulations, developed multiple high-performance web applications for the Fortune 100, and published Apple iOS applications. An in-demand speaker for international conferences, he has a special interest in using mobile, connected, real-time applications to help people live, work, and play better. In his spare time, Richard does Web development for non-profits, tends a garden full of California native plants, and cooks for family and charity events.

jQuery Concepts

This chapter introduces you to jQuery and provides examples of using jQuery APIs to manipulate elements in HTML5 Web pages. jQuery is an extremely popular open source JavaScript-based toolkit that provides a layer of abstraction over JavaScript. Moreover, jQuery supports multiple browsers, which simplifies the code in your HTML Web pages.

The jQuery homepage provides a download link for the source code, along with documentation, developer resources, and other useful links:

http://jquery.org

jQuery is an open source JavaScript toolkit that enables you to write cross-browser and cross-platform JavaScript code for managing elements in an HTML Web page, This includes finding, creating, updating, and deleting not only elements, but also element attributes, as well as the ability to add or remove style-related attributes of elements.

Some of the important and useful features of jQuery (in no particular order) include its support for cross-browser code, third-party jQuery plugins, themeable widgets, event handling, AJAX support, and simpler DOM traversal.

Using jQuery to Find Elements in Web Pages

A key point to remember is that the "$" prefix is the jQuery function. The $ prefix is a short-hand form of jQuery(), which means that the following two lines of code are the same:

```
var pElements1 = $("p");

var pElements2 = jQuery("p");
```

A third option is the use of window.jQuery, but this is less common than using jQuery or simply the $ sign.

One key point to remember is that a jQuery search actually returns a "result set," which is the set of elements that match the selection criteria. You can then apply an "action" to that set of elements. For example, you can find all the paragraphs in an HTML Web page and then set their text to red. After applying an action to a set of elements, a new set of elements is returned. In fact, you can apply a second action to that modified set, which returns yet another set. This process of applying multiple methods to a set is called *method chaining*, and the good news is that you "chain" together as many function invocations as you wish. Method chaining enables you to write very compact yet powerful code, as you will see in some examples in this chapter. As a preview, the following code snippet illustrates the use of jQuery method chaining:

```
$("ul li#item4").next().next().css({'font-size':24,
```

```
'background-color':'blue'});
```

As a simple example of how to use jQuery to select a set of elements in an HTML Web page, the following code snippet returns the set of <p> elements (if any) in an HTML Web page and assigns that set of elements to the JavaScript variable pElements:

```
var pElements = $("p");
```

The following code samples illustrate these and other jQuery concepts.

A "Hello World" Web Page with jQuery

The example in this section finds a *single* HTML <p> element and then changes its text. Later you will also see the modified code that enables you to manipulate an HTML Web page containing multiple HTML <p> elements.

Listing 1.1 displays the contents of HelloWorld1.html that illustrates how to add jQuery functionality to an HTML5 Web page that contains a single HTML <p> element.

NOTE *Listing 1.1 contains* console.log() *that is available in* WebKit*-based browsers, but might not be available without some type of plugin or extension for other browsers.*

LISTING 1.1 HelloWorld1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Hello World</title>

<script src="http://code.jquery.com/jquery-2.0.0b1.js">
</script>
<script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
</script>
</head>

<body>
  <p id="Steve">Hello World From a Paragraph</p>

  <script>
   $(document).ready(function(){
     // get the 'id' value
     var pId = $("p").attr("id");

     // get the text in the <p> element
```

```
    console.log (pId+" says "+pText);


    // update the text in the <p> element
    $("p").text("Goodbye World From a Paragraph");
    pText = $("p").text();
    console.log(pId+" says "+pText);
  });
 </script>
</body>
</html>
```

Listing 1.1 references two jQuery files with this code snippet:

```
<script src="http://code.jquery.com/jquery-2.0.0b1.js">
</script>
<script
 src="http://code.jquery.com/jquery-migrate-1.1.0.js">
</script>
```

Important: the first HTML <script> element in the HTML <body> element starts with this line:

```
$(document).ready(function(){
  // do something here
});
```

The preceding construct ensures that the DOM has been loaded into memory, so it's safe to access and manipulate DOM elements.

Remember that you can use the $ sign to represent jQuery, and you can get the value of an attribute of an HTML element (such as a <p> element) using the jQuery attr() function. For example, you can get the value of the id attribute of an HTML <p> element as follows:

```
var pId   = $("p").attr('id');
```

You can get the text string in an HTML <p> element using the jQuery text() function, as shown here:

```
// get the text in the <p> element
var pText = $("p").text();
console.log (pId+' says '+pText);
```

Finally, you can use the same text() function to update the text in an HTML <p> element, as shown here:

```
// update the text in the <p> element
$("p").text("Goodbye World From a Paragraph");
```

Launch the file in Listing 1.1 and open the Inspector that is available in your WebKit-

the Web page to see the output from the two console.log() statements in Listing 1.1. Keep in mind that the exact sequence of steps for using the Web Inspector is different for Chrome than for Safari, and the sequence will probably also change in future versions of these two browsers.

In case you don't already know, you can use Chrome Web Inspector to view the contents of variables, which can be very helpful for debugging purposes. You can experiment with the features of Chrome Web Inspector, and also read online tutorials about this excellent tool.

Querying and Modifying the DOM with jQuery

This section shows you how to use various jQuery modifiers that make it very easy to find and update elements in an HTML5 Web page. The code samples are short because they illustrate only one or two qualifiers, but you can combine them to perform very sophisticated DOM traversals and context-sensitive modifications to DOM elements.

Some of the qualifiers that are discussed in the code samples in this section includes :first, :last, :even, and :odd. A partial list of selectors includes :eq(), :lt(), :gt(), :has(), :contains(), and :eq().

Find and Modify Elements With :first and :last Qualifiers

The example in this section shows you how to use the jQuery :first and :last qualifiers to manipulate the text in HTML elements.

Listing 1.2 displays the contents of JQModifyElements1.html that illustrates how to switch the contents of two <p> elements.

LISTING 1.2 JQModifyElements1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>jQuery and Modifying Elements</title>

   <script src="http://code.jquery.com/jquery-2.0.0b1.js">
   </script>
   <script
     src="http://code.jquery.com/jquery-migrate-1.1.0.js">
   </script>
 </head>

 <body>
  <p style="color:red" id="Steve">Hello From Paragraph One</p>
  <p style="color:blue" id="Dave">Goodbye From Paragraph Two</p>
```

```
<script>
 $(document).ready(function(){
   // get information in first paragraph:
   var pId1   = $("p:first").attr('id');
   var pText1 = $("p:first").text();

   // get information in last paragraph:
   var pId2   = $("p:last").attr('id');
   var pText2 = $("p:last").text();

   $("p:first").html(pText2);
   $("p:last").html(pText1);
  //$("p:first").text(pText2);
  //$("p:last").text(pText1);
  });
 </script>
</body>
</html>
```

Listing 1.2 references the required jQuery file, adds two HTML <p> elements, and then extracts the value of the id attribute and the text in the first <p> element as shown here:

```
// get information in first paragraph:
var pId1   = $("p:first").attr('id');
var pText1 = $("p:first").text();
```

The next block of code performs the same thing with the second <p> element. Then, the text of the two <p> elements is switched with the following two lines of code:

```
$("p:first").html(pText2);
$("p:last").html(pText1);
```

Despite the simplicity of the jQuery code, this illustrates the ease with which you can manipulate HTML elements in an HTML Web page by means of the available jQuery functions.

Incidentally, you can get and set the value of an HTML <input> field (whose id attribute has value myInput) with the following two lines of code:

```
$("#myInput").val()
$("#myInput).text("new input value");
```

Figure 1.1 displays the result of rendering the page JQModifyElements1.html in a landscape-mode screenshot taken from an Asus Prime tablet with Android ICS.

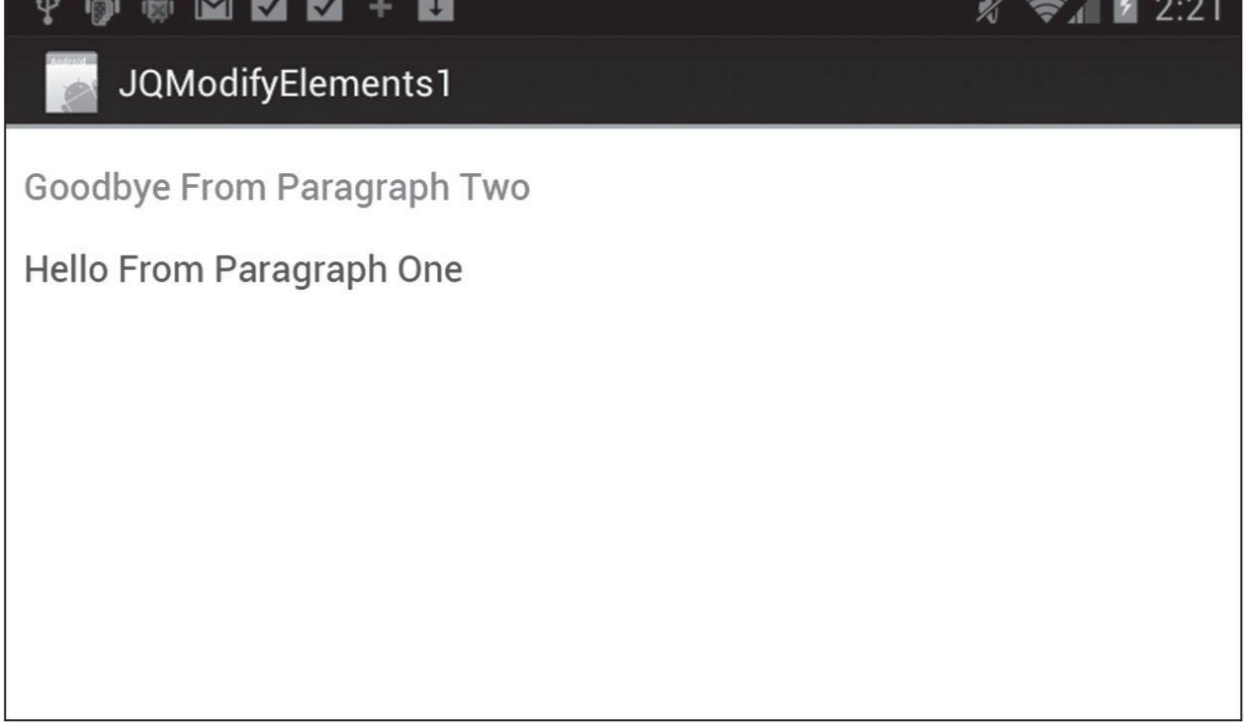


Figure 1.1 Modifying element in jQuery on an Asus Prime tablet with Android ICS.

The next section shows you how to use jQuery methods that can set collection of elements with the jQuery qualifiers :even() and :odd().

Finding Elements with **:eq, :lt,** and **:gt** Qualifiers

There are many jQuery functions available to perform sophisticated manipulations of HTML elements with relative ease. This section contains some useful code snippets that illustrate some of the other jQuery functions that are available.

For example, the jQuery qualifiers :eq(), :lt(), and :gt() respectively match elements whose position is equal to, less than, or greater than, in a list of items. Recall that since lists in jQuery start from index 0, the first item in a list has index zero of the list.

An example of finding the <p> element with index 3:

```
$('p:eq(3)').text('index equals three');
```

An example of finding the <p> element with index greater than 3:

```
$('p:gt(3)').text('index is greater than three');
```

An example of finding the <p> element with index less than 3:

```
$('p:lt(3)').text('index is less than three');
```

The preceding code snippets show you some of the things that are possible with jQuery functions. Also, there are jQuery functions that perform conditional tests on HTML elements.

For example, jQuery provides custom selectors, such as :has(), :contains(), and :eq(). You can use these selectors to select elements, as in the following example:

```
$("div:contains('foo')"))
```

```
$("div").contains('foo')
```

In addition, you can search for elements based on the value of their id attribute or by a specific class attribute, as shown in the next section.

Properties versus Attributes in jQuery

In general terms, an *attribute* is descriptive information attached to a DOM node that is not the node's contents or child nodes. For example, in the following code snippet:

```
<p class="intro">Hello</p>
```

class is an attribute with the value *"intro."*

A *property* is a value derived from the node (attributes and all), and is often writeable. For example, every node has an attributes property with a reference to all the attributes. In the preceding code snippet, there will be a className property that initialized from the *class* attribute.

In most cases you want the property, but if you need to see the original value, then use the attribute.

Additional information regarding properties and attributes is here:

https://developer.mozilla.org/en-US/docs/DOM/element

http://stackoverflow.com/questions/5874652/prop-vs-attr

jQuery provides the .prop() method for properties and the jQuery .attr() method for attributes. Unfortunately, the jQuery documentation describes the .prop() method and the .attr() method with an identical statement:

"Get the value of a property for the first element in the set of matched elements or set one or more properties for every matched element."

However, the good news is in the following statement:

"To maintain backwards compatability [sic], the .attr() method in jQuery 1.6.1+ will retrieve and update the property for you so no code for Boolean attributes is required to be changed to .prop()."

The safest thing to do is to use the jQuery .attr() method on primitive values (such as Boolean or single-valued strings) and to use the jQuery .prop() method for multi-valued strings (such as "style"). With this point in mind, the next section shows you how to work with attributes in jQuery.

Finding and Setting Element Attributes

You've seen how to use various jQuery functions to manipulate elements, and also how to find the value of an attribute. This section contains examples of updating the attributes of elements.

The following snippet gets the value of the src attribute of an element:

```
var $source = $("img").attr("src");
```

The next code snippet shows how to set the value of one attribute:

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

(displayed over multiple lines for convenience):

```
$(“img”).attr({
    src: “/images/MyHouse.jpg”,
    title: “House”,
    alt: “House”
});
```

The preceding code snippet sets the values of the attributes src, title, and alt to their respective values.

Notice that the syntax of the jQuery attr() method is very similar to the jQuery css() method, so when you understand one function, you will understand the other one as well.

Working with Custom Attributes

HTML5 supports custom attributes, provided that the attribute name starts with the string data- followed by an attribute name. Although you might not need to use custom attributes right now, jQuery Mobile relies heavily on custom attributes, so this functionality is extremely useful.

You can retrieve the values of custom attributes using the jQuery .data() method. For example, suppose your HTML Web page contains this snippet:

```
<div data-role=“page” data-value=“99” data-status=“new”></div>
```

You can retrieve the values of these custom attributes as follows:

$(“div”).data(“role”) returns the value page

$(“div”).data(“value”) returns the value 99

$(“div”).data(“status”) returns the value true

You will see more examples of manipulating custom data attributes in jQuery Mobile code samples, and if you’re really ambitious, you can find more examples in the jQuery Mobile source code.

Using jQuery to Remove Elements

As you can probably guess, jQuery enables you to remove elements in addition to finding and modifying elements in an HTML Web page. Listing 1.3 contains a portion of the HTML Web page JQRemovingElements1.html that illustrates how to remove elements via jQuery.

LISTING 1.3 JQRemovingElements1.html

```
<script>
  $(document).ready(function(){
    // remove the <p> element Dave
    $(“#Dave”).remove();


    // remove the <p> element Michelle
```

```
    // remove <p> elements containing “Goodbye”
    $(“p”).filter(“:contains(‘Goodbye’)”).remove();
  });
</script>
```

Listing 1.3 contains three lines of code for removing elements, the first of which is shown here (and the second is similar):

```
// remove the <p> element Dave
$(“#Dave”).remove();
```

Although the preceding code snippet performs just as you would expect, an example that illustrates the real power of jQuery is shown in the following code snippet, which uses the jQuery filter() method to find and then remove all the HTML <p> elements that contain the string Goodbye:

```
$(“p”).filter(“:contains(‘Goodbye’)”).remove();
```

Compare the simple and intuitive nature of the preceding single line of jQuery code with the corresponding JavaScript code that is required to perform the same functionality.

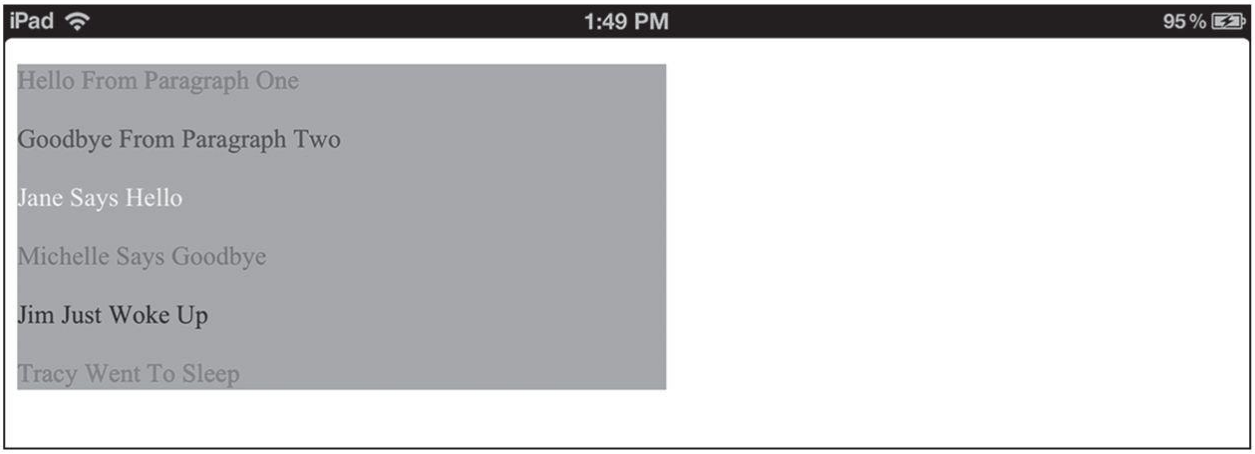


Figure 1.2 Removing elements with jQuery on an iPad3.

Figure 1.2 displays the result of rendering RemovingElements1.htmlin a portrait-mode screenshot taken from an iOS application running on an iPad3.

Creating DOM Elements

jQuery provides the clone() method and the append() method for creating new DOM elements. The clone() method creates a true copy of an element. On the other hand, the append() method operates on the specified element. Listing 1.4 displays the contents of JQCreatingElements1.html that illustrates how to use both of these jQuery methods in order to create new elements in an HTML Web page.

LISTING 1.4 JQCreatingElements1.html

```
<!DOCTYPE html>
<html lang=“en”>
```

```
<script>
 $(document).ready(function(){
   // append a clone of the #Dave element to “#div2”:
   $(“#Dave”).clone().css({color:”#000”}).appendTo(“#div2”);

   // append another clone of the #Dave element to “#div2”:
   $(“#Dave”).clone().css({color:”#00f”})
     .appendTo(“#div2”);

   // move the red #Dave to the end of “#div4”:
   $(“#Dave”).appendTo(“#div4”);

   // prepend #Dave to all the ‘div’ elements:
  //$(“#Dave”).clone().prependTo(“div”);
 });
</script>
</body>
</html>
```

Listing 1.3 introduces the jQuery clone() method, an example of which is shown here:

```
// append a clone of the #Dave element to “#div2”:
$(“#Dave”).clone().css({color:”#000”}).appendTo(“#div2”);
```

The purpose of the preceding code snippet is clear, and you can even read it from left to right to grasp its purpose: clone the element whose id is Dave, set its color to black, and append this cloned element to the element whose id is div2.

The only other new functionality in Listing 1.3 is the jQuery prependTo() function, which inserts an element before (instead of after) a specified element, as shown here:

```
// prepend #Dave to all the ‘div’ elements:
//$(“#Dave”).clone().prependTo(“div”);
```

One other point to remember: clone(true) will also propagate the event handlers of the source element. There are other jQuery methods for inserting DOM elements, some of which are described in a later section.

Figure 1.3 displays the result of rendering JQCreatingElements1.html in a landscape-mode screenshot taken from an iOS application running on an iPad3.

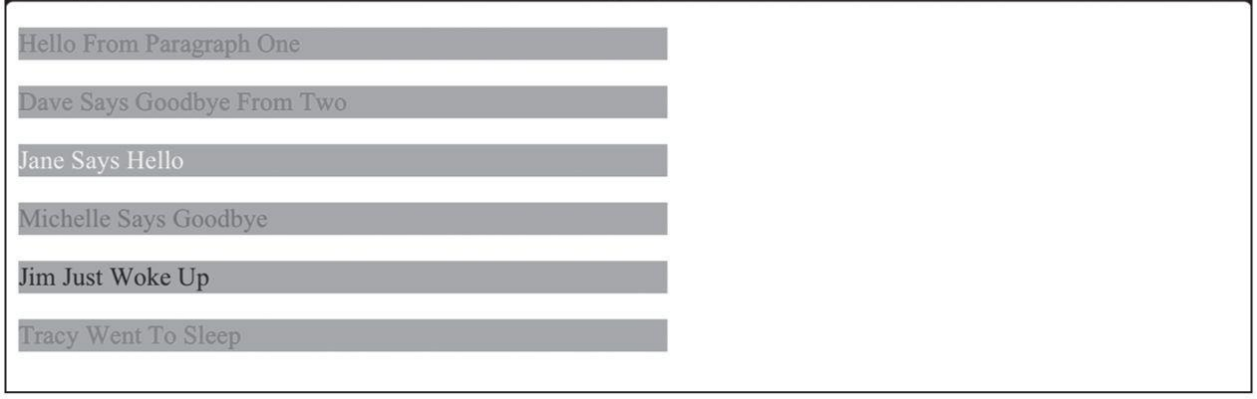


Figure 1.3 Creating elements with jQuery on an iPad3.

The jQuery **append()** and **appendTo()** methods

The jQuery documentation provides a succinct explanation about these two methods:

"The.append()and .appendTo()methods perform the same task. The major difference is in the syntax-specifically, in the placement of the content and target. With .append(), the selector expression preceding the method is the container into which the content is inserted. With .appendTo(), on the other hand, the content precedes the method, either as a selector expression or as markup created on the fly, and it is inserted into the target container."

Perhaps this casual explanation will clarify the difference:

{select-something-here}.append(and-append-stuff-from-here)

{specify-stuff-here}.appendTo(and-append-that-stuff-here)

This chapter contains code samples that use these methods so that you will remember the difference between these two jQuery methods.

Useful jQuery Code Blocks

This section contains a set of code snippets that enable you to perform conditional logic and then execute your custom code. The code samples in this section are straightforward, and the comments explain their purpose.

Check if jQuery is loaded:

```
if (typeof jQuery == 'undefined') {
  // jQuery is not loaded
}
```

Check if an element exists:

```
if ( $('#myElement').length > 0 ) {
  // the element exists
}
```

Note that the ">" symbol in the preceding code snippet is often omitted in the "truthy" style of programming, but it's good to be explicit in your code.

Checking for empty elements:

```
$('*').each(function() {
  if ($(this).text() == "") {
    //do something here
  }
});
```

Returns true or false based on content of a element:

```
var emptyTest = $('#myDiv').is(':empty');
```

Determine if a checkbox is checked (returns true/false):

```
$('#checkBox').attr('checked');
```

Find all checked checkboxes:

```
$('input[type=checkbox]:checked');
```

Disable/enable inputs for a button element:

```
$("#submit-button").attr("disabled", true);
```

Remove an attribute from a button element:

```
$("#submit-button").removeAttr("disabled");
```

The preceding code snippets give you an idea of the compact manner in which you can check various conditions. As you become more proficient with jQuery, you will develop your own set of useful code snippets.

Now let's take a look at another important class of jQuery functions that enable you to navigate around the DOM, some of which are discussed in the next section.

Handling Click Events in jQuery

jQuery provides support for various types of events and user gestures that you can "bind" to custom code (written by you) that is executed whenever those events or gestures take place. The events that you can detect and bind in jQuery Mobile are discussed in the jQuery Mobile chapter.

The `click()` function enables you to handle click events using the following syntax:

```
$("#button1").click(function() {
    // do something
}
```

There are several techniques for handling events, and the recommended technique for doing so is shown here:

```
$("#button1").on("click"), function() {
    // do something
}
```

The `dblclick()` function enables you to handle double click events, and an example of the syntax is here:

```
    // do something
  }
```

Incidentally, the focus() function provides focus on selected elements. Although it is not covered here, you can get more information by consulting the online documentation.

Listing 1.5 displays most of the contents of JQClickDivs1.html that illustrates how to detect click events and then update the contents of both elements in this HTML5 Web page.

LISTING 1.5 JQClickDivs1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Detecting Click Events with jQuery</title>

  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script
   src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>
</head>

<body>
 <div id="div1">The first div element </div>
 <div id="div2">The second div element </div>

 <script>
  var click1=0, click2=0, total=0;

  $(document).ready(function() {
    $("#div1").click(function() {
      ++click1;
      ++total;
      $(this).text("Clicked: "+click1+" total: "+total);
     $("#div2").text("Clicked: "+click2+" total:
                 "+total);
    });

    $("#div2").click(function() {
```

```
        ++total;
        $(this).text("Clicked: "+click2+" total: "+total);
       $("#div1").text("Clicked: "+click1+" total: "+total);
      });
    });
  </script>
 </body>
</html>
```

Listing 1.5 references the required jQuery file, followed by some CSS styling definitions, along with two HTML elements. The code for adding a click event listener to the first HTML element is shown here (with similar jQuery code for the second HTML element):

```
$("#div1").click(function() {
    ++click1;
    ++total;
    $(this).text("Clicked: "+click1+" total: "+total);
    $("#div2").text("Clicked: "+click2+" total: "+total);
});
```

Whenever users click on the preceding HTML element, its click count and the total click count are incremented, and the text of both HTML elements are updated with the click count for the individual elements as well as the sum of the click counts for both elements.

Although the example in Listing 1.4 is simplistic, it does illustrate how to keep track of events in different HTML elements in an HTML Web page. A more realistic example could involve an HTML Web page with an HTML Form that has inter-dependencies between elements in the form.

Figure 1.4 displays the result of rendering the HTML page JQClickDivs1.html in a landscape-mode screenshot taken from an iOS application running on an Asus Prime tablet with Android ICS.

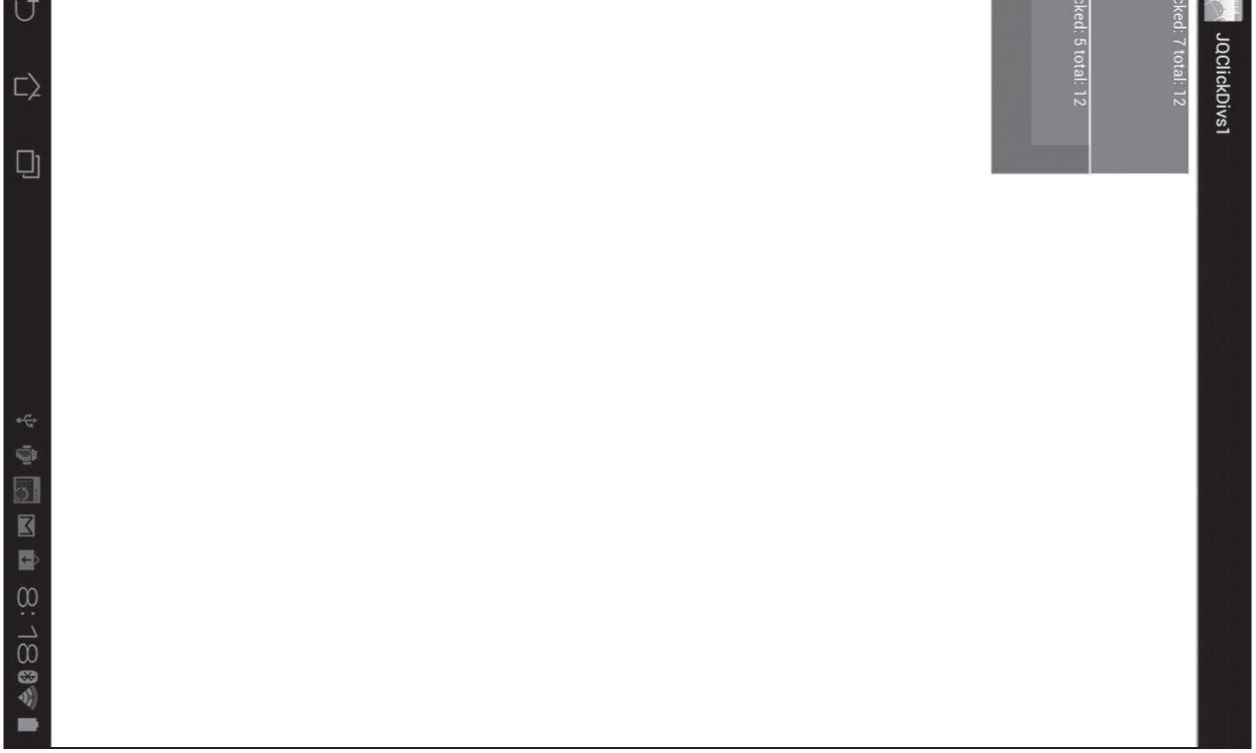


Figure 1.4 Counting click events on an Asus Prime Tablet with Android ICS.

Handling Events in jQuery 1.7 and Beyond

In jQuery 1.7 and beyond, the preferred method for defining an event handler uses the following construct:

```
$(“some-element”).on(some-event)
```

However, versions of jQuery prior to version 1.7 provide several techniques to bind events to elements, and three of these techniques have been deprecated. Since version 1.7 was introduced recently, you will probably be exposed to HTML Web pages containing earlier versions of jQuery for quite some time. Consequently, you need to be aware of those other coding techniques so that you will be able to read the jQuery code in HTML Web pages that use earlier versions of jQuery.

For the purpose of illustration, suppose that you need to bind a “click” event to an HTML `<div>` element whose `id` attribute is `div1`.

The preferred method for defining an event handler with this code block (which you saw earlier in this chapter) is shown here:

```
$(“#div1”).on(“click”), function() {
  // do something
}
```

However, HTML Web pages using older versions of jQuery also use an event handler defined like this:

```
$(“#div1”).click(function() {
  // do something
```

A third method for defining an event handler is to use the bind() method, which has been deprecated in version 1.7:

```
$(“#button1”).bind(“click”), function() {
  // do something
}
```

A fourth method for defining an event handler is to use the live() method, which has also been deprecated in version 1.7:

```
$(“#div1”).live(“click”), function() {
  // do something
}
```

The bind() method and the live() method attach a handler to an event to any element that matches the current selector. In addition, the live() method attaches the same handler to elements created later, which match the current selector.

A fifth method for defining an event handler is to use the delegate() method, which has also been deprecated in version 1.7:

```
$(“#div1”).delegate(“click”), function() {
  // do something
}
```

In the preceding code blocks, an event handler was defined in order to handle a click event, but similar comments apply to other user-initiated events, such as swipeleft and swiperight. If you want to learn more about other changes in jQuery 1.7, you can find a summary of the changes (with links) here:

http://api.jquery.com/category/version/1.7/

You can also get detailed information regarding new functionality and changes in jQuery 1.7 on this Web page:

http://blog.jquery.com/2011/11/03/jquery-1-7-released/

Additional information regarding the most recent versions of jQuery is available in the Preface of this book.

Chaining jQuery Functions

You have already seen examples of chaining jQuery commands, and you might have used chained commands in Java (especially with JAXB) such as the following:

```
myList().getFirstElem().getCustomer().setFirstName(“Dave”);
```

jQuery chaining supports more sophisticated operations than the preceding code snippet. By default, jQuery references the first element in a chain, but you can change the default behavior. For example, you can instruct jQuery to reference the current element using the .parent() method, perform some manipulation, and then use the jQuery .end() method to reference the first element in the chain again.

here:

http://blog.pengoworks.com/index.cfm/2007/10/26/jQuery-Understanding-the-chain

Keep in mind the following caveats: use caution when you change the default behavior in method chaining because the actual behavior might not be the behavior that you expect in your code. In addition, make sure that you clearly document such code, lest you confuse people who are less knowledgeable than you about jQuery.

Accelerometer Values with jQuery

The example in this section illustrates how you can use jQuery to obtain accelerometer values for a mobile device.

Listing 1.6 displays the contents of JQAccelerometer1.html that illustrates how to display the accelerometer values of a mobile device whenever the device undergoes acceleration in any direction. The CSS stylesheet JQAccelerometer1.css contains simple selectors that are not shown here, but you can find the complete listing on the CD.

LISTING 1.6 JQAccelerometer1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>jQuery and Accelerometer</title>

  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script
   src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>

 <script>
  var colorX = "", colorY = "", colorZ = "";
  var intx = 0, inty = 0, intz = 0;
  var colors = ['#f00', '#ff0', '#00f'];

  $('document').ready(function(){
   $(window).bind("devicemotion", function(e){
     var accelEvent = e.originalEvent,
       acceler = accelEvent.accelerationIncluding
           Gravity,
```

```
        if(x < 0)      { intx = 0; }
        else if(x < 1) { intx = 1; }
        else           { intx = 2; }

        if(y < 0)      { inty = 0; }
        else if(y < 1) { inty = 1; }
        else           { inty = 2; }

        if(z < 0)      { intz = 0; }
        else if(z < 1) { intz = 1; }
        else           { intz = 2; }

        colorX = colors[intx];
        colorY = colors[inty];
        colorZ = colors[intz];

        $("#valueX").css("backgroundColor", colorX);
        $("#valueY").css("backgroundColor", colorY);
        $("#valueZ").css("backgroundColor", colorZ);

       $("#valueX").html("<p>Acceleration x: <b>" + x +
                                  "</b></p>");
       $("#valueY").html("<p>Acceleration y: <b>" + x +
                                  "</b></p>");
       $("#valueZ").html("<p>Acceleration z: <b>" + x +
                                  "</b></p>");
     });
   });
  </script>
</head>

<body>
   <h2>Accelerometer Values</h2>
   <div id="outer">
     <div id="valueX"></div>
     <div id="valueY"></div>
```

```
  </div>
 </body>
</html>
```

The code in Listing 1.6 obtains accelerometer values for three directions (all perpendicular to each other) for a mobile device, and then performs some arithmetic calculations in order to compute integer values to be used as indexes into an array of color values. After determining the color associated with each direction, the associated rectangular element is updated with the corresponding color.

After binding the window object to the devicemotion event, we can use the event object (in this case called e) to obtain a JavaScript reference to the acceleration object (which is called acceler). We can then extract current values for the three different axes, as shown here:

```
$('document').ready(function(){
  $(window).bind("devicemotion", function(e){
    var accelEvent = e.originalEvent,
        acceler = accelEvent.accelerationIncludingGravity,
        x = acceler.x, y = acceler.y, z = acceler.z;
```

For simplicity, the array of colors contains only three colors, and the following code computes a number between 0 and 2 in order to determine the color for the x direction:

```
if(x < 0)      { intx = 0; }
else if(x < 1) { intx = 1; }
else           { intx = 2; }
```

The color for the x direction is calculated like this:

```
colorX = colors[intx];
```

The background color of the HTML element that is associated with the x direction is updated with the following code:

```
$("#valueX").css("backgroundColor", colorX);
```

Finally, the current value of the acceleration in the x direction is displayed using the following code snippet:

```
$("#valueX").html("<p>Acceleration x: <b>" + x + "</b></p>");
```

The corresponding values for the y direction and the z direction are computed in a similar fashion.

Figure 1.5 displays the result of rendering the HTML Web page in Listing 1.5 in a landscape-mode screenshot taken from an Android application running on an Asus Prime tablet with Android ICS.

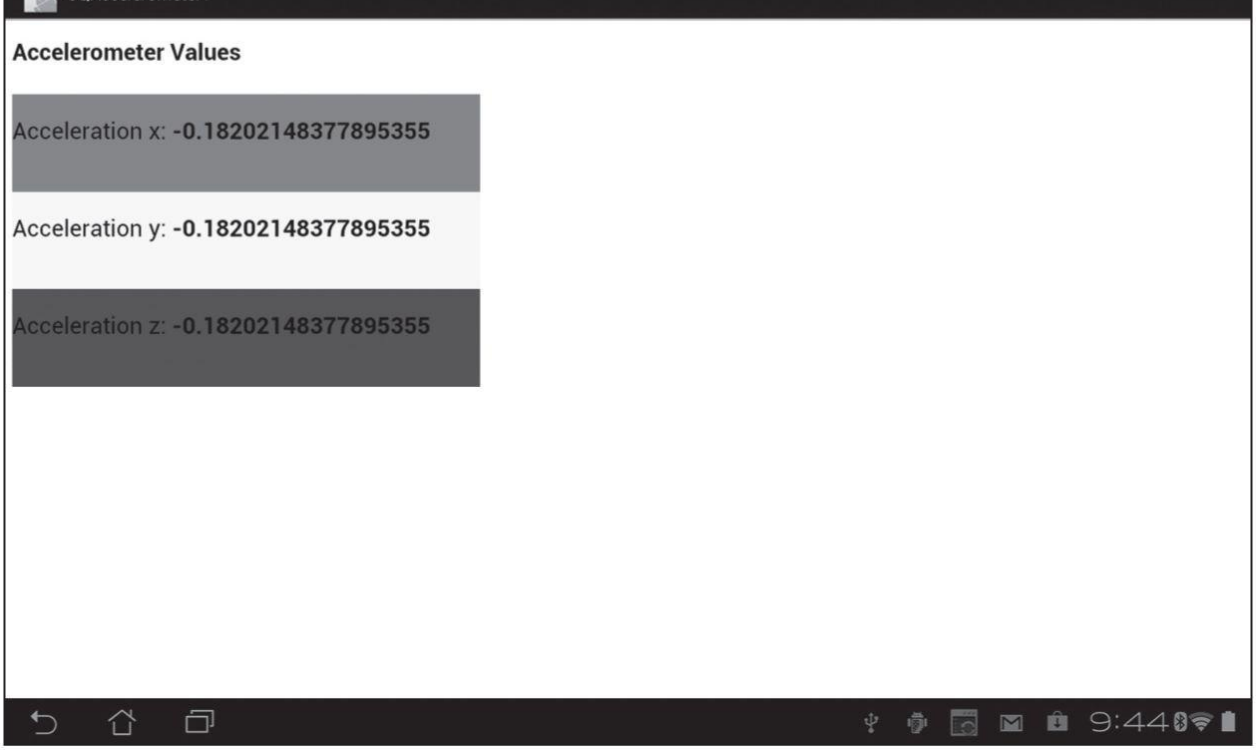


Figure 1.5 Accelerometer on an Asus Prime tablet with Android ICS.

Summary

This chapter introduced you to jQuery, along with code samples that illustrated how to use jQuery functions to manipulate an HTML Web page. You saw code samples that showed how to do the following:

- Create a simple jQuery-based HTML5 Web page
- Find and modify Elements With :first and :last qualifiers
- Find elements with :even and :odd qualifiers
- Find elements with :eq, :lt, and :gt qualifiers
- Find elements by class or id
- Find/set element Attributes
- Find form elements and their attributes
- CSS3-style expressions for finding elements
- Remove DOM elements
- Create DOM elements
- Handle events in jQuery
- Use the click() function
- Chaining jQuery functions
- Accelerometer with jQuery

INTRODUCTION TO CSS3

This chapter introduces various aspects of CSS3, such as 2D/3D graphics and 2D/3D animation. In some cases, CSS3 concepts are presented without code samples due to space limitations; however, those concepts are included because it's important for you to be aware of their existence. By necessity, this chapter assumes that you have a moderate understanding of CSS, which means that you can write a basic stylesheet with selectors and properties. If you are unfamiliar with CSS selectors, there are many introductory articles available through an Internet search. If you are convinced that CSS operates under confusing and seemingly arcane rules, then it's probably worth your while to read an online article about the CSS box model, after which you will have a better understanding of the underlying logic of CSS.

The first part of this chapter contains code samples that illustrate how to create shadow effects, how to render rectangles with rounded corners, and also how to use linear and radial gradients. The second part of this chapter covers CSS3 transforms (scale, rotate, skew, and translate), along with code samples that illustrate how to apply transforms to HTML elements and to JPG files.

The third part of this chapter covers CSS3 2D graphics, such as linear gradients, radial gradients, and CSS3 transforms.

You can launch the code samples in this chapter in a Webkit-based browser on a desktop or a laptop. You can also view them on mobile devices, provided that you launch them in a browser that supports the CSS3 features that are used in the code samples. For your convenience, many of the code samples in this chapter are accompanied by screenshots of the code samples on a Sprint Nexus S 4G and an Asus Prime Android ICS 10" tablet (both on Android ICS), which enables you to compare those screenshots with the corresponding images that are rendered on Webkit-based browsers on desktops and laptops. In Chapter 10, you will learn the process of creating Android applications that can launch HTML5 Web pages.

CSS3 Support and Browser-Specific Prefixes for CSS3 Properties

Before we delve into the details of CSS3, there are two important details that you need to know about defining CSS3-based selectors for HTML pages. First, you need to know the CSS3 features that are available in different browsers. One of the best Websites for determining browser support for CSS3 features is here:

http://caniuse.com/

The preceding link contains tabular information regarding CSS3 support in IE, Firefox, Safari, Chrome, and Opera, as well as several mobile browsers.

Another highly useful tool that checks for CSS3 feature support is Enhance.js. It tests browsers to determine whether or not they can support a set of essential CSS and JavaScript properties, and then delivers features to those browsers that satisfy the test. You can download Enhance.js here:

https://github.com/filamentgroup/EnhanceJS

A third useful tool is Modernizr, which checks for HTML5-related feature detection in various browsers, and its homepage is here:

http://www.modernizr.com/

At some point you will start using JavaScript in your HTML5 Web pages (indeed, you probably do so already), and Modernizr provides a programmatic way to check for many HTML5 and CSS3 features in different browsers.

In order to use Modernizr, include the following code snippet in the <head> element of your Web pages:

```
<script src="modernizr.min.js" type="text/javascript"></script>
```

Navigate to the Modernizr homepage where you can read the documentation, tutorials, and details regarding the set of feature detection.

The second detail that you need to know is that many CSS3 properties currently require browser-specific prefixes in order for them to work correctly. The prefix applies to "work in progress" for individual browsers, and the final specification drops browser-specific prefixes. The prefixes –ie-, -moz-, and -o- are for Internet Explorer, Firefox, and Opera, respectively. Note that Opera also supports –webkit- prefixes, and it's possible that other browsers will do the same (check the respective Websites for updates).

As an illustration, the following code block shows examples of these prefixes:

```
-ie-webkit-border-radius: 8px;

-moz-webkit-border-radius: 8px;

-o-webkit-border-radius: 8px;

border-radius: 8px;
```

In your CSS selectors, specify the attributes with browser-specific prefixes before the "generic" property, which serves as a default choice in the event that the browser-specific attributes are not selected. The CSS3 code samples in this book contain WebKit-specific prefixes, which helps us keep the CSS stylesheets manageable in terms of size. If you need CSS stylesheets that work on multiple browsers (for current versions as well as older versions), there are essentially two options available. One option involves manually adding the CSS3 code with all the required browser-specific prefixes, which can be tedious to maintain and also error-prone. Another option is to use CSS toolkits or frameworks (discussed in the next chapter), which can programmatically generate the CSS3 code that contains all browser-specific prefixes.

Finally, an extensive list of browser-prefixed CSS properties is here:

http://peter.sh/experiments/vendor-prefixed-css-property-overview/

An extensive list of prefix-free CSS properties is here:

Quick Overview of CSS3 Features

CSS3 adopts a modularized approach involving multiple sub-specifications for extending existing CSS2 functionality as well as supporting new functionality. As such, CSS3 can be logically divided into the following categories:

- Backgrounds/borders
- Color
- Media queries
- Multi-column layout
- Selectors

With CSS3, you can create boxes with rounded corners and shadow effects; create rich graphics effects using linear and radial gradients; detect portrait and landscape mode; detect the type of mobile device using media query selectors; and produce multi-column text rendering and formatting.

In addition, CSS3 enables you to define sophisticated node selection rules in selectors using pseudo-classes (described in the next section), first or last child (:first-child, :last-child, :first-of-type, and :last-of-type), and also pattern-matching tests for attributes of elements. Several sections in this chapter contain examples of how to create such selection rules.

CSS3 Pseudo Classes and Attribute Selection

This brief section contains examples of some pseudo-classes, followed by snippets that show you how to select elements based on the relative position of text strings in various attributes of those elements.

Recall that the class attribute in CSS provides a way to mark a group of elements as having a certain property, such as the following code snippet:

```
<p class="details">
```

On the other hand, pseudo-classes enable you to reference an arbitrary group of elements by some identifying feature that they possess. For example, the following code snippet collects the first paragraph in each HTML <section> element in an HTML Web page:

```
section:p:first-of-type
```

CSS3 supports an extensive and rich set of pseudo-classes, including nth-child(), along with some of its semantically related "variants," such as nth-of-type(), nth-first-of-type(), nth-last-of-type(), and nth-last-child().

CSS3 also supports Boolean selectors (which are also pseudo-classes) such as empty, enabled, disabled, and checked, which are very useful for Form-related HTML elements. One other pseudo class is not(), which returns a set of elements that do not match the selection criteria.

Although this section focuses on the :nth-child() pseudo-class, you will become familiar with various other CSS3 pseudo-classes. In the event that you need to use those pseudo-classes, a link is provided at the end of this section, which contains more information and

CSS3 Pseudo Classes

The CSS3 :nth-child() pseudo-class is both powerful and useful, and it has the following form:

```
:nth-child(insert-a-keyword-or-linear-expression-here)
```

The following list provides various examples of using the nth-child() pseudo-class in order to match various subsets of child elements of an HTML element (which can be substituted by other HTML elements as well):

div:nth-child(1): matches the first child element

div:nth-child(2): matches the second child element

div:nth-child(:even): matches the even child elements

div:nth-child(:odd): matches the odd child elements

The interesting and powerful aspect of the nth-child() pseudo-class is its support for linear expressions of the form an+b, where a is a positive integer and b is a non-negative integer, as shown here (using an HTML5 element):

div:nth-child(3n): matches every third child, starting from position 0

div:nth-child(3n+1): matches every third child, starting from position 1

div:nth-child(3n+2): matches every third child, starting from position 2

Another very useful pseudo-class involves :hover, which can easily create nice visual effects. The following example of the CSS3 :hover pseudo-class changes the font size of an element whose id attribute has value text1, whenever users hover over the associated element with their mouse:

```
#text1:hover {
  font-size: 12pt;
}
```

CSS3 Attribute Selection

You can specify CSS3 selectors that select HTML elements, as well as HTML elements based on the value of an attribute of an HTML element using various regular expressions. For example, the following selector selects img elements whose src attribute starts with the text string Laurie, and then sets the width attribute and the height attribute of the selected img elements to 100px:

```
img[src^=“Laurie”] {
  width: 100px; height: 100px;
}
```

The preceding CSS3 selector is useful when you want to set different dimensions to images based on the name of the images (Laurie, Shelly, Steve, and so forth).

The following HTML <img> elements do not match the preceding selector:

```
<img src="3Laurrie" width="200" height="200" />
```

CSS3 uses the meta-characters ^, $, and * (followed by the = symbol) in order to match an initial, terminal, or arbitrary position for a text string. If you are familiar with the Unix utilities grep and sed, as well as the vi text editor, then these meta-characters are very familiar to you. However, CSS3 imposes a restriction for using meta-characters: they can only be used in the context of an attribute match (which uses square brackets).

The following selector selects HTML img elements whose src attribute ends with the text string jpeg, and then sets the width attribute and the height attribute of the selected img elements to 150px:

```
img[src$="jpeg"] {
  width: 150px; height: 150px;
}
```

The preceding CSS3 selector is useful when you want to set different dimensions to images based on the type of the images (jpg, png, jpeg, and so forth).

The following selector selects HTML img elements whose src attribute contains any occurrence of the text string baby, and then sets the width attribute and the height attribute of the selected HTML img elements to 200px:

```
img[src*="baby"] {
  width: 200px; height: 200px;
}
```

The preceding CSS3 selector is useful when you want to set different dimensions to images based on the "classification" of the images (mybaby, yourbaby, babygirl, babyboy, and so forth).

If you want to learn more about patterns (and their descriptions) that you can use in CSS3 selectors, an extensive list is available here:

http://www.w3.org/TR/css3-selectors

This concludes part one of this chapter, and the next section delves into CSS3 graphics-oriented effects, such as rounded corners and shadow effects.

CSS3 Shadow Effects and Rounded Corners

CSS3 shadow effects are useful for creating vivid visual effects. You can use shadow effects for text as well as rectangular regions. CSS3 also enables you to easily render rectangles with rounded corners, so you do not need JPG files in order to create this effect.

Specifying Colors with RGB and HSL

Before we delve into the interesting features of CSS3, you need to know how to represent colors. One method is to use (R, G, B) triples, which represent the Red, Green, and Blue components of a color. For instance, the triples (255, 0, 0), (255, 255, 0), and (0, 0, 255) respectively represent the colors Red, Yellow, and Blue. Other ways of specifying the color include: the hexadecimal triples (F, 0, 0) and (FF, 0, 0); the decimal triple (100%, 0, 0); or the

which is a decimal number between 0 (invisible) to 1 (opaque) inclusive.

However, there is also the HSL (Hue, Saturation, and Luminosity) representation of colors, where the first component is an angle between 0 and 360 (0 degrees is north), and the other two components are percentages between 0 and 100. For instance, (0, 100%, 50%), (120, 100%, 50%), and (240, 100%, 50%) represent the colors Red, Green, and Blue, respectively.

The code samples in this book use (R, G, B) and (R, G, B, A) for representing colors, but you can perform an Internet search to obtain more information regarding HSL.

CSS3 and Text Shadow Effects

A shadow effect for text can make a Web page look more vivid and appealing. Many Websites look better with shadow effects that are not overpowering for users (unless you specifically need to do so).

Listing 2.1 displays the contents of the HTML5 page TextShadow1.html that illustrate how to render text with a shadow effect, and Listing 2.2 displays the contents of the CSS stylesheet TextShadow1.css that is referenced in Listing 2.1.

LISTING 2.1 TextShadow1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>CSS Text Shadow Example</title>
  <link href="TextShadow1.css"
       rel="stylesheet" type="text/css">
</head>

<body>
  <div id="text1">Line One Shadow Effect</div>
  <div id="text2">Line Two Shadow Effect</div>
  <div id="text3">Line Three Vivid Effect</div>
  <div id="text4">
    <span id="dd">13</span>
    <span id="mm">August</span>
    <span id="yy">2012</span>
  </div>
  <div id="text5">
    <span id="dd">13</span>
    <span id="mm">August</span>
    <span id="yy">2012</span>
```

```
  <div id="text6">
    <span id="dd">13</span>
    <span id="mm">August</span>
    <span id="yy">2012</span>
  </div>
</body>
</html>
```

The code in Listing 2.1 is straightforward: there is a reference to the CSS stylesheet TextShadow1.css that contains two CSS selectors. One selector specifies how to render the HTML element whose id attribute has value text1, and the other selector matches the HTML element whose id attribute is text2. Although the CSS3 rotate() function is included in this example, we'll defer a more detailed discussion of this function until later in this chapter.

LISTING 2.2 TextShadow1.css

```
#text1 {
  font-size: 24pt;
  text-shadow: 2px 4px 5px #00f;
}

#text2 {
  font-size: 32pt;
  text-shadow: 0px 1px 6px #000,
               4px 5px 6px #f00;
}
/* note the multiple parts in the text-shadow definition */
#text3 {
  font-size: 40pt;
  text-shadow: 0px 1px 6px  #fff,
               2px 4px 4px  #0ff,
               4px 5px 6px  #00f,
               0px 0px 10px #444,
               0px 0px 20px #844,
               0px 0px 30px #a44,
               0px 0px 40px #f44;
}

#text4 {
  position: absolute;
```

```
  right: 200px;
  font-size: 48pt;
  text-shadow: 0px 1px 6px  #fff,
               2px 4px 4px  #0ff,
               4px 5px 6px  #00f,
               0px 0px 10px #000,
               0px 0px 20px #448,
               0px 0px 30px #a4a,
               0px 0px 40px #fff;
  -webkit-transform: rotate(-90deg);
}

#text5 {
  position: absolute;
  left: 0px;
  font-size: 48pt;
  text-shadow: 2px 4px 5px #00f;
  -webkit-transform: rotate(-10deg);
}

#text6 {
  float: left;
  font-size: 48pt;
  text-shadow: 2px 4px 5px #f00;
  -webkit-transform: rotate(-170deg);
}

/* 'transform' is explained later */
#text1:hover, #text2:hover, #text3:hover,
#text4:hover, #text5:hover, #text6:hover {
-webkit-transform : scale(2) rotate(-45deg);
transform : scale(2) rotate(-45deg);
}
```

The first selector in Listing 2.2 specifies a font-size of 24 and a text-shadow that renders text with a blue background (represented by the hexadecimal value #00f). The attribute text-shadow specifies (from left to right) the x-coordinate, the y-coordinate, the blur radius, and the color of the shadow. The second selector specifies a font-size of 32 and a red shadow background (#f00). The third selector creates a richer visual effect by specifying multiple

that are possible with different values in the various components.

The final CSS3 selector creates an animation effect whenever users hover over any of the six text strings, and the details of the animation will be deferred until later in this chapter.

Figure 2.1 displays the result of matching the selectors in the CSS stylesheet TextShadow1.css with the HTML elements in the HTML page TextShadow1.html. The landscape-mode screenshot is taken from an Android application (based on the code in Listing 2.1 and Listing 2.2) running on a Nexus S 4G (Android ICS) smart phone.


1:27
TextShadow1
Line One Shadow Effect
Line Two
Three
August

Figure 2.1 CSS3 text shadow effects.

CSS3 and Box Shadow Effects

You can also apply a shadow effect to a box that encloses a text string, which can be effective in terms of drawing attention to specific parts of a Web page. However, the same caveat regarding over-use applies to box shadows.

The HTML page BoxShadow1.html and BoxShadow1.css are not shown here, but they are available on the CD. Together, they render a box shadow effect.

The key property is the box-shadow property, as shown here in bold for Mozilla, WebKit, and the non-prefixed property:

```
#box1 {
  position:relative;top:10px;
  width: 50%;
  height: 30px;
  font-size: 20px;
  -moz-box-shadow: 10px 10px 5px #800;
  -webkit-box-shadow: 10px 10px 5px #800;
  box-shadow: 10px 10px 5px #800;
```

Figure 2.2 displays a landscape-mode screenshot is taken from a Nexus S 4G with Android ICS (based on the code in BoxShadow1.html and BoxShadow1.css).

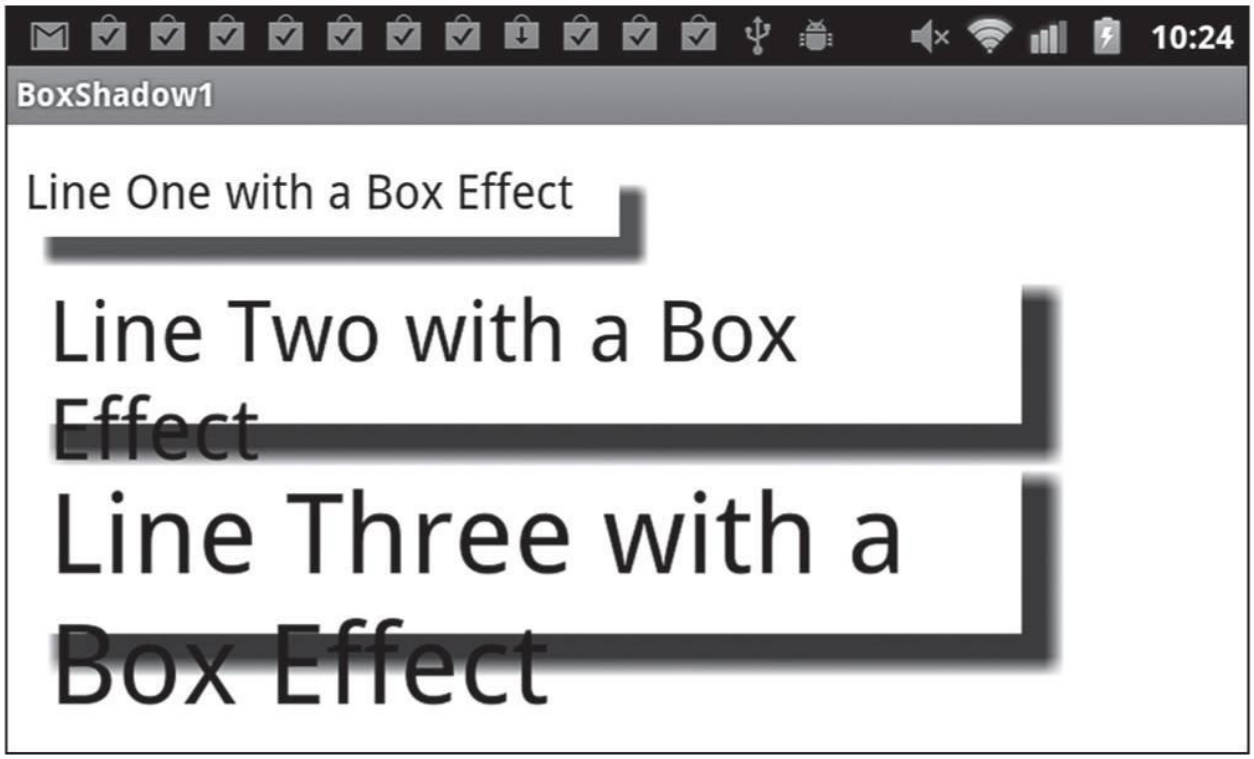


Figure 2.2 CSS3 box shadow effect on a Sprint Nexus with Android ICS.

CSS3 and Rounded Corners

Web developers have waited a long time for rounded corners in CSS, and CSS3 makes it very easy to render boxes with rounded corners. Listing 2.3 displays the contents of the HTML page RoundedCorners1.html that renders text strings in boxes with rounded corners, and Listing 2.4 displays the CSS file RoundedCorners1.css.

LISTING 2.3 RoundedCorners1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <link href="RoundedCorners1.css" rel="stylesheet"
                                    type="text/css">
</head>
<body>
  <div id="outer">
    <a href="#" class="anchor">Text Inside a Rounded Rectangle</a>
  </div>
  <div id="text1">Line One of Text with a Shadow Effect</div>
  <div id="text2">Line Two of Text with a Shadow Effect</div>
</body>
</html>
```

Listing 2.3 contains a reference to the CSS stylesheet RoundedCorners1.css that contains three CSS selectors that match the elements whose id attribute has value anchor, text1, and text2, respectively. The CSS selectors defined in RoundedCorners1.css create visual effects, and as you will see, the hover pseudo-selector enables you to create animation effects.

LISTING 2.4 RoundedCorners1.css

```
a.anchor:hover {
background: #00F;
}

a.anchor {
background: #FF0;
font-size: 24px;
font-weight: bold;
padding: 4px 4px;
color: rgba(255,0,0,0.8);
text-shadow: 0 1px 1px rgba(0,0,0,0.4);
-webkit-border-radius: 8px;
border-radius: 8px;
```

Listing 2.4 contains the selector a.anchor:hover that changes the text color from yellow (#FF0) to blue (#00F) during a two-second interval whenever users hover over any anchor element with their mouse.

The selector a.anchor contains various attributes that specify the dimensions of the box that encloses the text in the <a> element, along with two new pairs of attributes. The first pair specifies the border-radius attribute (and the WebKit-specific attribute) whose value is 8px, which determines the radius (in pixels) of the rounded corners of the box that encloses the text in the <a> element. The last two selectors are identical to the selectors in Listing 2.1.

Figure 2.3 displays the result of matching the selectors that are defined in the CSS stylesheet RoundedCorners1.css with elements in the HTML page RoundedCorners1.html in a landscape-mode screenshot taken from an Asus Prime tablet with Android ICS.

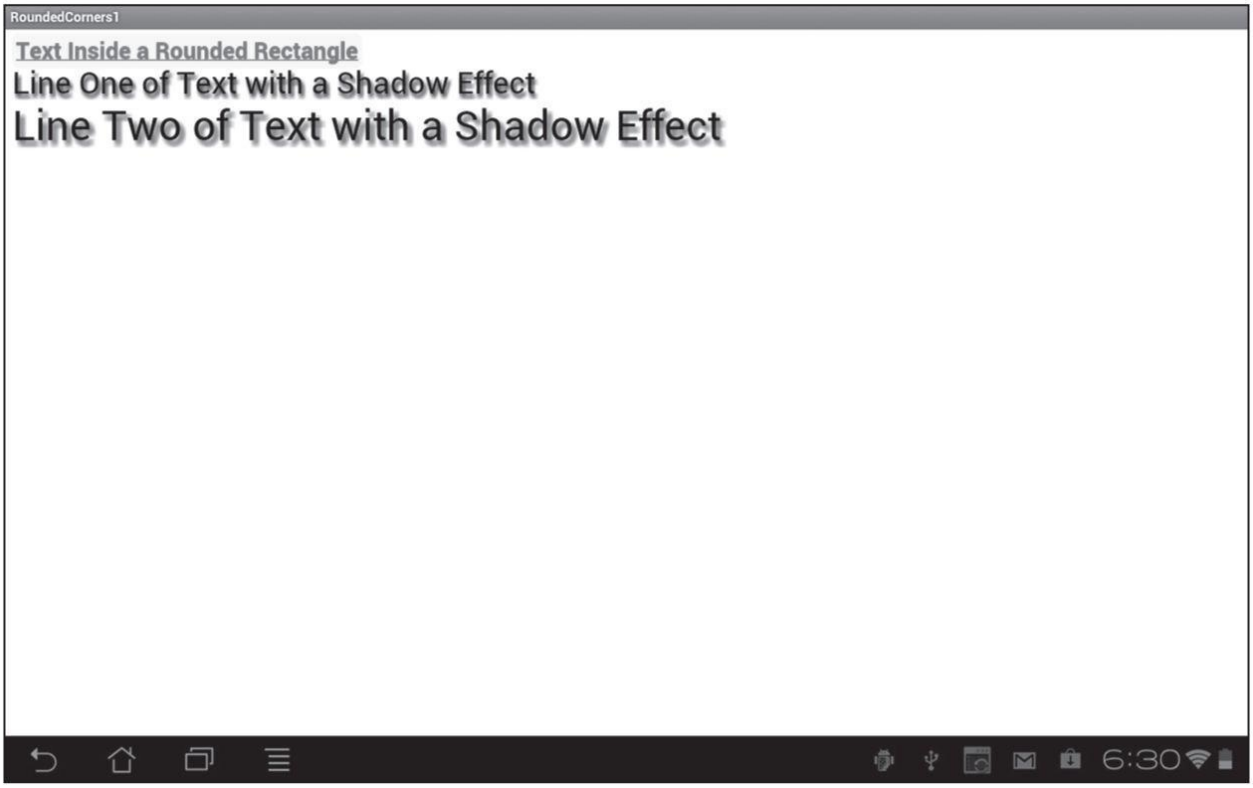


Figure 2.3 CSS3 rounded corners effect on an Asus Prime tablet with Android ICS.

CSS3 Gradients

CSS3 supports linear gradients and radial gradients, which enable you to create gradient effects that are as visually rich as gradients in other technologies such as SVG. The code samples in this section illustrate how to define linear gradients and radial gradients in CSS3 and then match them to HTML elements.

Linear Gradients

CSS3 linear gradients require you to specify one or more “color stops,” each of which specifies a start color, and end color, and a rendering pattern. Webkit-based browsers support the following syntax to define a linear gradient:

- An end point
- A start color using from()
- Zero or more stop-colors
- An end color using to()

A start point can be specified as an (x, y) pair of numbers or percentages. For example, the pair (100, 25%) specifies the point that is 100 pixels to the right of the origin and 25% of the way down from the top of the pattern. Recall that the origin is located in the upper-left corner of the screen.

Listing 2.5 displays the contents of LinearGradient1.html and Listing 2.6 displays the contents of LinearGradient1.css, which illustrate how to use linear gradients with text strings that are enclosed in <p> elements and an <h3> element.

LISTING 2.5 LinearGradient1.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>CSS Linear Gradient Example</title>
  <link href="LinearGradient1.css"
      rel="stylesheet" type="text/css">
</head>

<body>
  <div id="outer">
    <p id="line1">line 1 with a linear gradient</p>
    <p id="line2">line 2 with a linear gradient</p>
    <p id="line3">line 3 with a linear gradient</p>
    <p id="line4">line 4 with a linear gradient</p>
    <p id="outline">line 5 with Shadow Outline</p>
    <h3><a href="#">A Line of Gradient Text</a></h3>
  </div>
</body>
</html>
```

Listing 2.5 is a simple Web page containing five <p> elements and one <h3> element. Listing 2.5 also references the CSS stylesheet LinearGradient1.css, which contains CSS selectors that match the four <p> elements and the <h3> element in Listing 2.5.

LISTING 2.6 LinearGradient1.css

```
#line1 {
width: 50%;
```

```
background-image: -webkit-gradient(linear, 0% 0%, 0% 100%, from(#fff), to(#f00));
background-image: -gradient(linear, 0% 0%, 0% 100%, from(#fff), to(#f00));
-webkit-border-radius: 4px;
border-radius: 4px;
}

#line2 {
width: 50%;
font-size: 32px;
background-image: -webkit-gradient(linear, 100% 0%, 0% 100%, from(#fff), to(#ff0));
background-image: -gradient(linear, 100% 0%, 0% 100%, from(#fff), to(#ff0));
-webkit-border-radius: 4px;
border-radius: 4px;
}

#line3 {
width: 50%;
font-size: 32px;
background-image: -webkit-gradient(linear, 0% 0%, 0% 100%, from(#f00), to(#00f));
background-image: -gradient(linear, 0% 0%, 0% 100%, from(#f00), to(#00f));
-webkit-border-radius: 4px;
border-radius: 4px;
}

#line4 {
width: 50%;
font-size: 32px;
background-image: -webkit-gradient(linear, 100% 0%, 0% 100%, from(#f00), to(#00f));
background-image: -gradient(linear, 100% 0%, 0% 100%, from(#f00), to(#00f));
-webkit-border-radius: 4px;
border-radius: 4px;
}

#outline {
font-size: 2.0em;
font-weight: bold;
color: #fff;
```

```
}

h3 {
width: 50%;
position: relative;
margin-top: 0;
font-size: 32px;
font-family: helvetica, ariel;
}

h3 a {
position: relative;
color: red;
text-decoration: none;
-webkit-mask-image: -webkit-gradient(linear, left top, left bottom, from(rgba(0,0,0,1)), color-stop(50%, rgba(0,0,0,0.5)), to(rgba(0,0,0,0)));
}

h3:after {
content:“This is a Line of Gradient Text”;
color: blue;
}
```

The first selector in Listing 2.6 specifies a font-size of 32 for text, a border-radius of 4 (which renders rounded corners), and a linear gradient that varies from white to blue, as shown here:

```
#line1 {
width: 50%;
font-size: 32px;
background-image: -webkit-gradient(linear, 0% 0%, 0% 100%,
                    from(#fff), to(#f00));
background-image: -gradient(linear, 0% 0%, 0% 100%,
                    from(#fff), to(#f00));
-webkit-border-radius: 4px;
border-radius: 4px;
}
```

As you can see, the first selector contains two attributes with a -webkit- prefix and two standard attributes without this prefix. Since the next three selectors in Listing 2.6 are similar to the first selector, we will not discuss their content.

text in white with a thin black shadow, as shown here:

```
color: #fff;
text-shadow: 1px 1px 1px rgba(0,0,0,0.5);
```

The final portion of Listing 2.6 contains three selectors that affect the rendering of the <h3> element and its embedded <a> element. The h3 selector specifies the width and font size; the h3 selector specifies a linear gradient; and the h3:after selector specifies the text string "This is a Line of Gradient Text" to display *after* the HTML5 <h3> element. (*Note*: you can use h3:before to specify a text string to display *before* an HTML5 <h3> element.) Other attributes are specified, but these are the main attributes for these selectors.

Figure 2.4 displays the result of matching the selectors in the CSS stylesheet LinearGradient1.css to the HTML page LinearGradient1.html in a landscape-mode screenshot taken from an Android application running on an Asus Prime tablet with Android ICS.

Radial Gradients

CSS3 radial gradients are more complex than CSS3 linear gradients, but you can use them to create more complex gradient effects. Webkit-based browsers support the following syntax to define a radial gradient:

- A start point
- A start radius
- An end point
- An end radius
- A start color using from()
- Zero or more color-stops
- An end color using to()

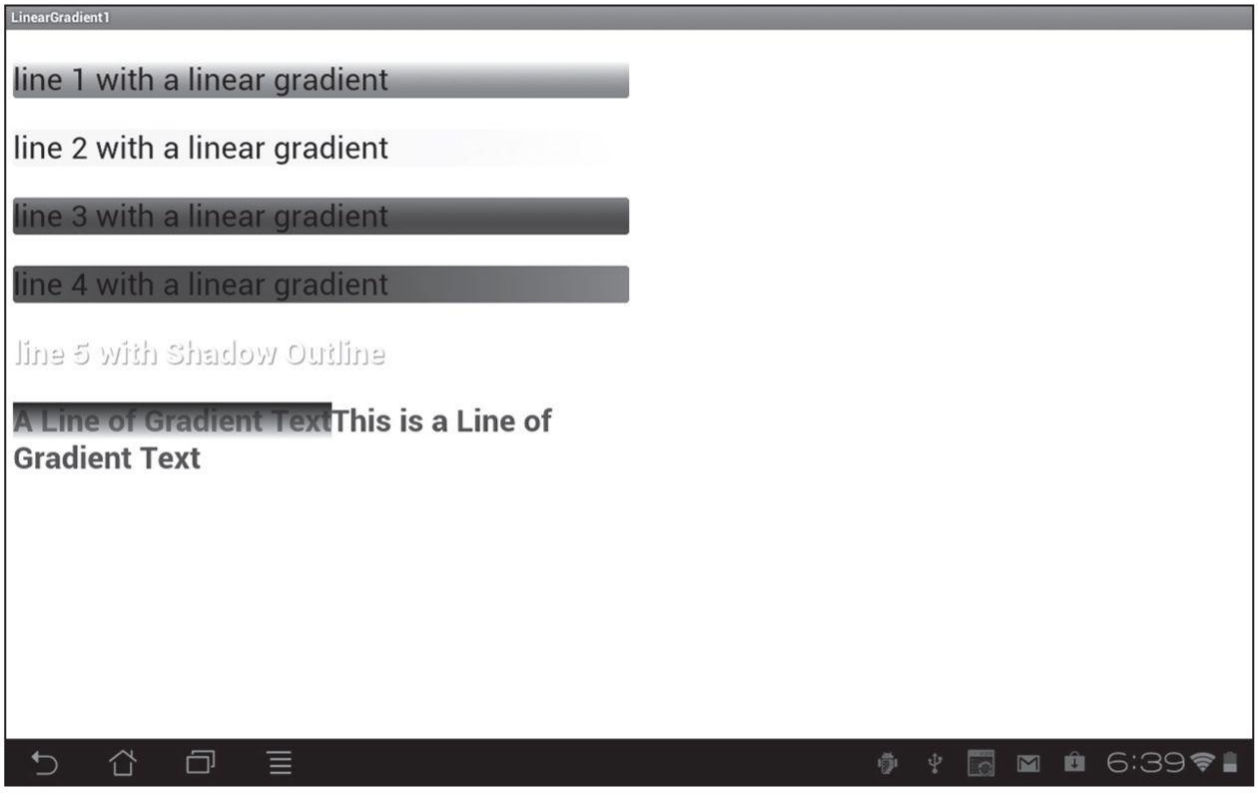


Figure 2.4 CSS3 Linear gradient effect on an Asus Prime 10” tablet with Android ICS.

Notice that the syntax for a radial gradient is similar to the syntax for a linear gradient, except that you also specify a start radius and an end radius.

The HTML5 Web page RadialGradient1.html and the CSS stylesheet RadialGradient1.css are not shown here, but the full listing is available on the CD. The essence of the code in the HTML5 code involves this code block:

```
<div id=“outer”>
  <div id=“radial3”>Text3</div>
  <div id=“radial2”>Text2</div>
  <div id=“radial4”>Text4</div>
  <div id=“radial1”>Text1</div>
 </div>
```

The CSS stylesheet RadialGradient1.css contains five CSS selectors that match the five HTML elements, and one of the selectors is shown here:

```
#radial1 {
background: -webkit-gradient(
  radial, 500 40%, 0, 301 25%, 360, from(red),
  color-stop(0.05, orange), color-stop(0.4, yellow),
  color-stop(0.6, green), color-stop(0.8, blue),
  to(#fff)
 );
}
```

The #radial1 selector contains a background attribute that defines a radial gradient using the –webkit- prefix, and it specifies the following:

- A start point of (500, 40%)
- A start radius of 0
- An end point of (301, 25%)
- An end radius of 360
- A start color of red
- An end color of white (#fff)

The other selectors have the same syntax as the first selector, but the rendered radial gradients are significantly different. You can create these (and other) effects by specifying different start points and end points, and by specifying a start radius that is larger than the end radius.

Figure 2.5 displays the result of matching the selectors in the CSS stylesheet

from an Android application running on an Asus Prime tablet with Android ICS.

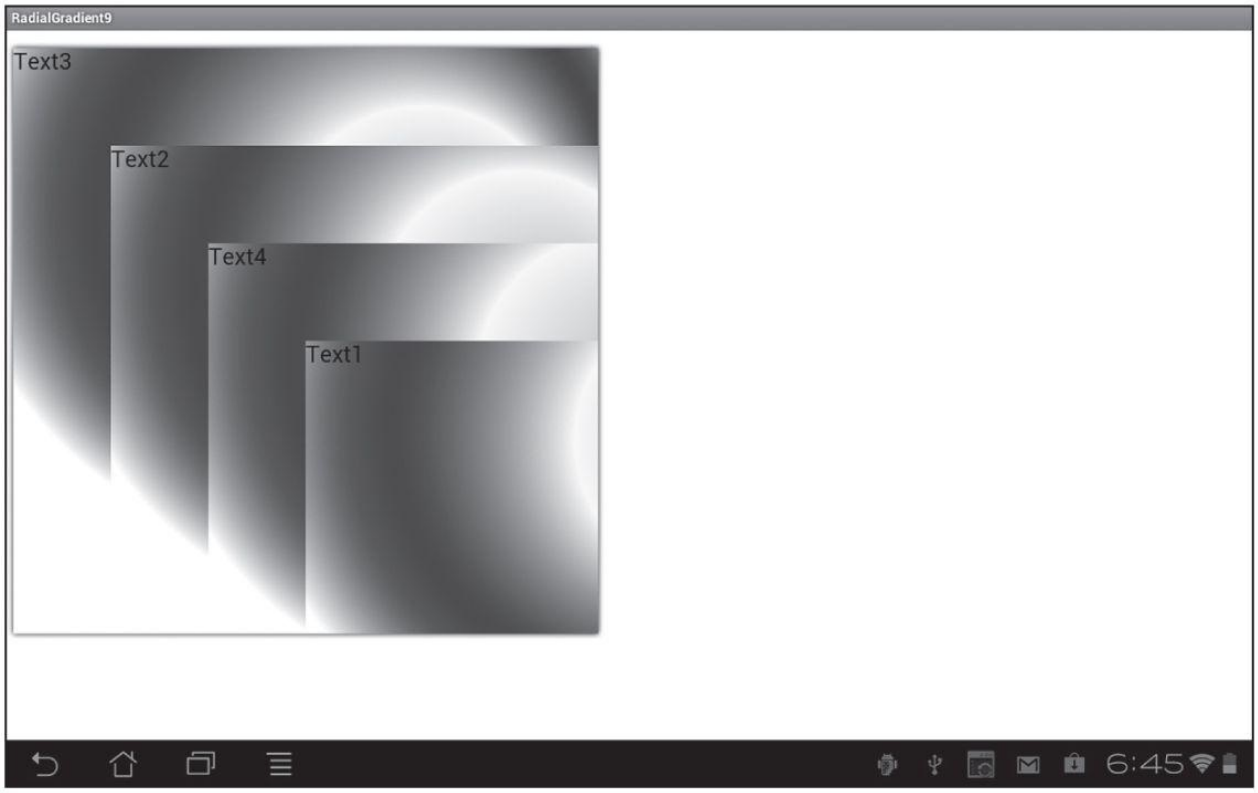


Figure 2.5 CSS3 Radial gradient effect on an Asus Prime tablet with Android ICS.

CSS3 2D Transforms

In addition to transitions, CSS3 supports four common transforms that you can apply to 2D shapes and also to JPG files. The four CSS3 transforms are scale, rotate, skew, and translate. The following sections contain code samples that illustrate how to apply each of these CSS3 transforms to a set of JPG files. The animation effects occur when users hover over any of the JPG files; moreover, you can create "partial" animation effects by moving your mouse quickly between adjacent JPG files.

Listing 2.7 displays the contents of Scale1.html and Listing 2.8 displays the contents of Scale1.css, which illustrate how to scale JPG files to create a "hover box" image gallery.

LISTING 2.7 Scale1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>CSS Scale Transform Example</title>
  <link href="Scale1.css"
        rel="stylesheet" type="text/css">
</head>
```

```
  <header><h1>Hover Over any of the Images:</h1></header>

  <div id="outer">
   <img src="sample1.png" class="scaled" width="150" height="150"/>
    <img src="sample2.png" class="scaled" width="150" height="150"/>
    <img src="sample1.png" class="scaled" width="150" height="150"/>
    <img src="sample2.png" class="scaled" width="150" height="150"/>
  </div>
</body>
</html>
```

Listing 2.7 references the CSS stylesheet Scale1.css (which contains selectors for creating scaled effects) and four HTML <img> elements that references the JPG files sample1.png and sample2.png. The remainder of Listing 2.7 is straightforward, with simple boilerplate text and HTML elements.

LISTING 2.8 Scale1.css

```
#outer {
float: left;
position: relative; top: 50px; left: 50px;
}

img {
-webkit-transition: -webkit-transform 1.0s ease;
transition: transform 1.0s ease;
}

img.scaled {
  -webkit-box-shadow: 10px 10px 5px #800;
  box-shadow: 10px 10px 5px #800;
}

img.scaled:hover {
-webkit-transform : scale(2);
transform : scale(2);
}
```

The img selector in Listing 2.8 specifies a transition property that applies a transform effect occurring during a one-second interval using the ease function, as shown here:

```
transition: transform 1.0s ease;
```

effect (which you saw earlier in this chapter), as shown here:

```
img.scaled {
  -webkit-box-shadow: 10px 10px 5px #800;
  box-shadow: 10px 10px 5px #800;
}
```

Finally, the selector img.scaled:hover specifies a transform attribute that uses the scale() function in order to double the size of the associated JPG file whenever users hover over any of the <img> elements with their mouse, as shown here:

```
transform : scale(2);
```

Since the img selector specifies a one-second interval using an ease function, the scaling effect will last for one second. Experiment with different values for the CSS3 scale() function and also different value for the time interval to create the animation effects that suit your needs.

Another point to remember is that you can scale both horizontally and vertically:

```
img {
-webkit-transition: -webkit-transform 1.0s ease;
transition: transform 1.0s ease;
}


img.mystyle:hover {
-webkit-transform : scaleX(1.5) scaleY(0.5);
transform : scaleX(1.5) scaleY(0.5);
}
```

Figure 2.6 displays the result of matching the selectors in the CSS stylesheet Scale1.css to the HTML page Scale1.html. The landscape-mode screenshot is taken from an Android application (based on the code in Listing 2.7 and Listing 2.8) running on a Nexus S 4G smart phone with Android ICS.

Rotate Transforms

The CSS3 transform attribute allows you to specify the rotate() function in order to create scaling effects, and its syntax looks like this:

```
rotate(someValue);
```

You can replace someValue with any number. When someValue is positive, the rotation is clockwise; when someValue is negative, the rotation is counter clockwise; and when someValue is zero, there is no rotation effect. In all cases, the initial position for the rotation effect is the positive horizontal axis.

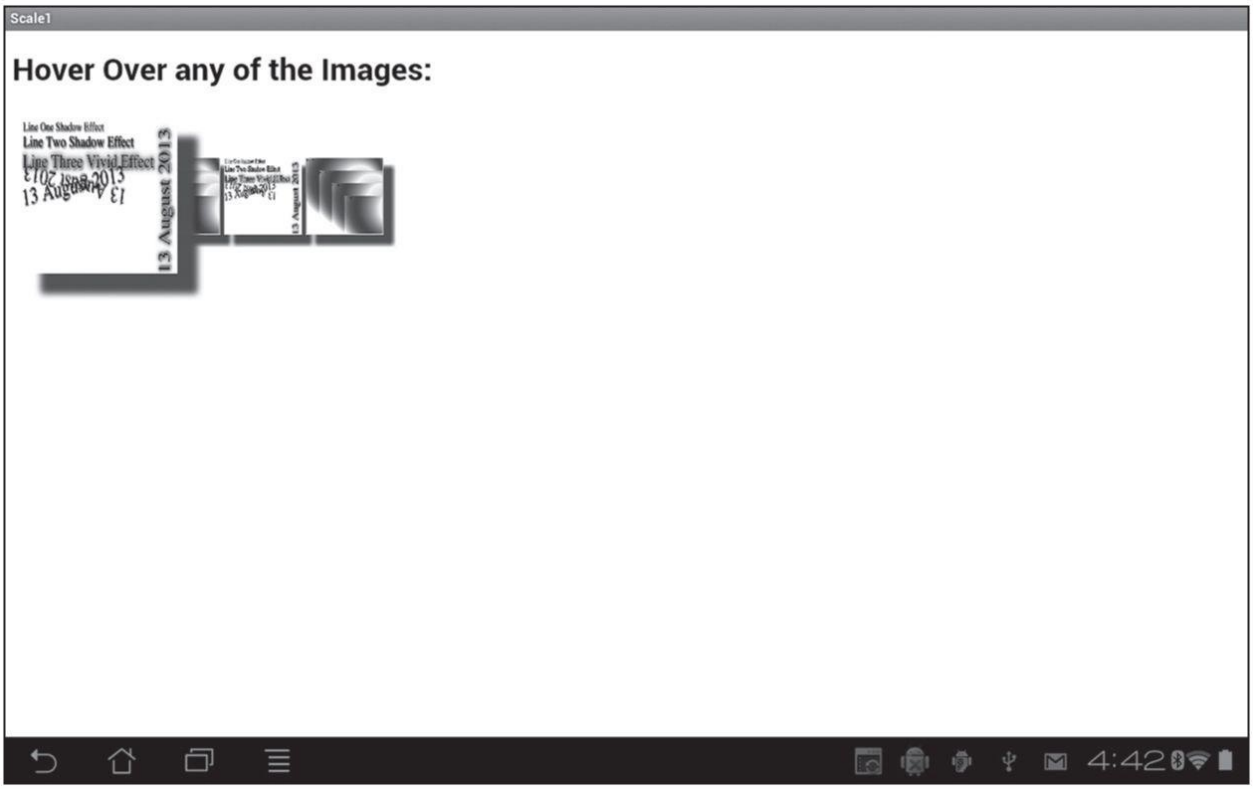


Figure 2.6 CSS3-based scaling effect on JPG files.

The HTML5 Web page Rotate1.html and the CSS stylesheet Rotate1.css on the CD illustrate how to create rotation effects, a sample of which is shown here:

```
img.imageL:hover {
-webkit-transform : scale(2) rotate(-45deg);
transform : scale(2) rotate(-45deg);
}
```

The img selector that specifies a transition attribute that creates an animation effect during a one-second interval using the ease timing function, as shown here:

```
transition: transform 1.0s ease;
```

The CSS3 transform attribute allows you to specify the skew() function in order to create skewing effects, and its syntax looks like this:

```
skew(xAngle, yAngle);
```

You can replace xAngle and yAngle with any number. When xAngle and yAngle are positive, the skew effect is clockwise; when xAngle and yAngle are negative, the skew effect is counter clockwise; and when xAngle and yAngle are zero, there is no skew effect. In all cases, the initial position for the skew effect is the positive horizontal axis.

The HTML5 Web page Skew1.html and the CSS stylesheet Skew1.css are on the CD, and they illustrate how to create skew effects. The CSS stylesheet contains the img selector that specifies a transition attribute, which in turn creates an animation effect during a

```
transition: transform 1.0s ease;
```

There are also the four selectors img.skewed1, img.skewed2, img.skewed3, and img.skewed4 create background shadow effects with darker shades of red, yellow, green, and blue, respectively (all of which you have seen in earlier code samples).

The selector img.skewed1:hover specifies a transform attribute that performs a skew effect whenever users hover over the first <img> element with their mouse, as shown here:

```
transform : scale(2) skew(-10deg, -30deg);
```

The other three CSS3 selectors also use a combination of the CSS functions skew() and scale()to create distinct visual effects. Notice that the fourth hover selector also sets the opacity property to 0.5, which takes place in parallel with the other effects in this selector.

Figure 2.7 displays the result of matching the selectors in the CSS stylesheet Skew1.css to the elements in the HTML page Skew1.html. The landscape-mode screenshot is taken from an Android application running on a Nexus S 4G smart phone with Android ICS.

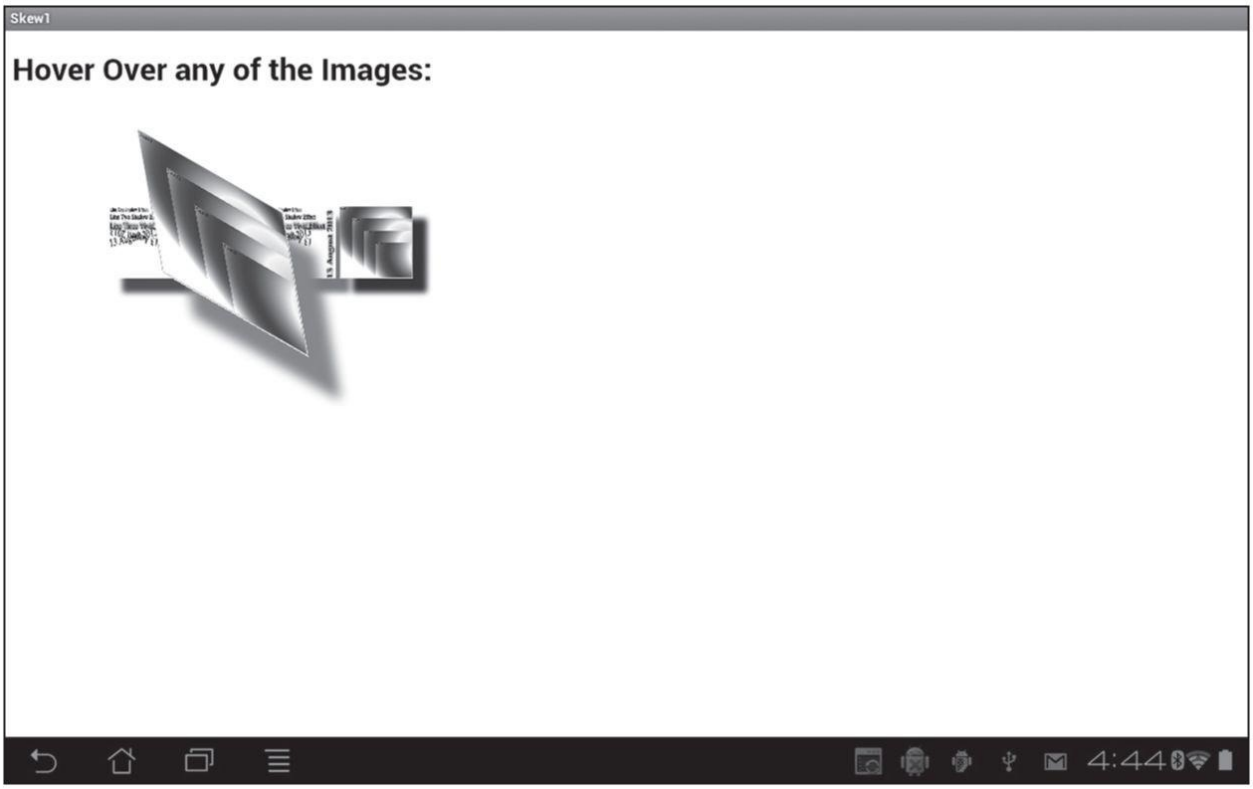


Figure 2.7 CSS3-based skew effects on JPG files.

The CSS3 transform attribute allows you to specify the translate() function in order to create an effect that involves a horizontal and/or vertical “shift” of an element, and its syntax looks like this:

```
translate(xDirection, yDirection);
```

The translation is in relation to the origin, which is the upper-left corner of the screen. Thus, positive values for xDirection and yDirection produce a shift to the right and downward, respectively, whereas negative values for xDirection and yDirection produce a shift to the left and upward. Zero values for xDirection and yDirection do not cause any translation effect.

The Web page Translate1.html and the CSS stylesheet Translate1.css on the CD illustrate how to apply a translation effect to a JPG file.

```
img.trans2:hover {
-webkit-transform : scale(0.5) translate(-50px, -50px);
transform : scale(0.5) translate(-50px, -50px);
}
```

The CSS stylesheet contains the img selector specifies a transform effect during a one-second interval using the ease timing function, as shown here:

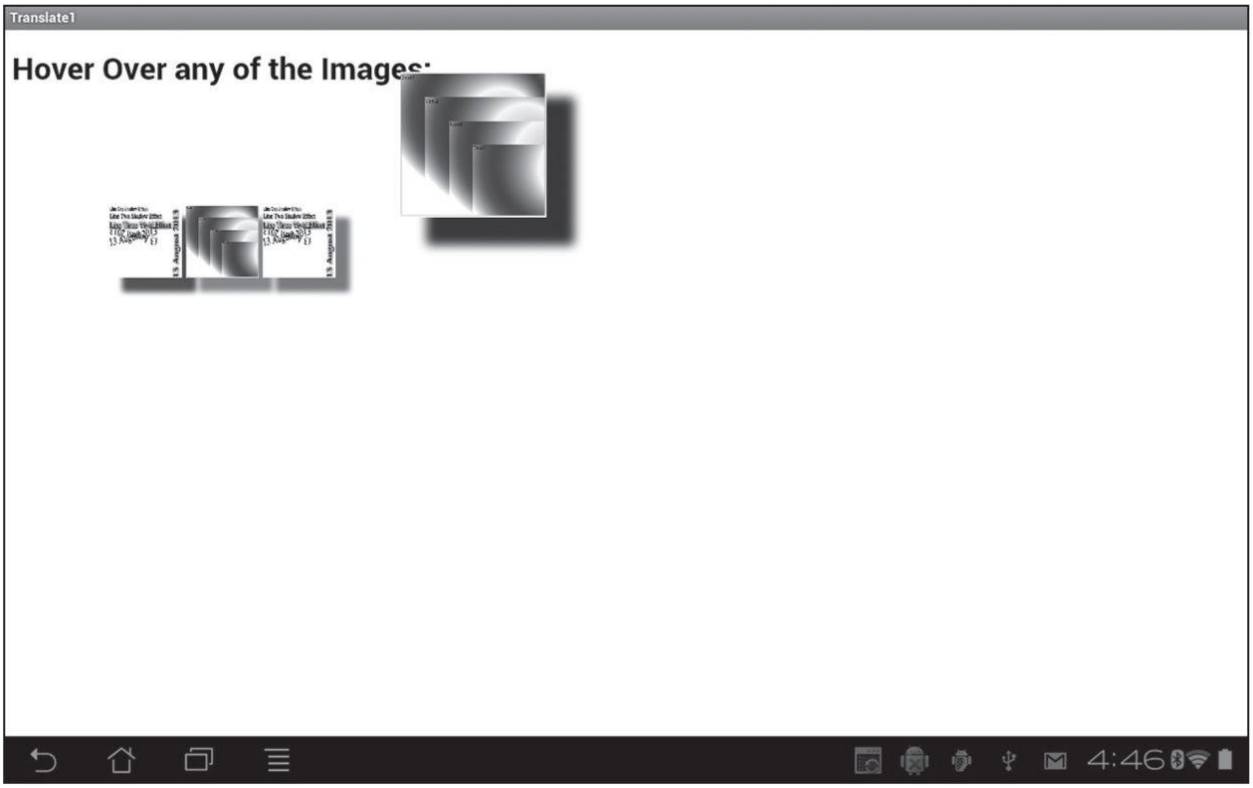
```
transition: transform 1.0s ease;
```

The four selectors img.trans1, img.trans2, img.trans3, and img.trans4 create background shadow effects with darker shades of red, yellow, green, and blue, respectively, just as you saw in the previous section.

The selector img.trans1:hover specifies a transform attribute that performs a scale effect and a translation effect whenever users hover over the first <img> element with their mouse, as shown here:

```
-webkit-transform : scale(2) translate(100px, 50px);
transform : scale(2) translate(100px, 50px);
```

Figure 2.8 displays the result of matching the selectors defined in the CSS3 stylesheet Translate1.css to the elements in the HTML page Translate1.html. The landscape-mode screenshot is taken from an Android application running on a Nexus S 4G smart phone with Android ICS.



CSS3 Media Queries

CSS3 media queries are very useful logical expressions that enable you to detect mobile applications on devices with differing physical attributes and orientation. For example, with CSS3 media queries you can change the dimensions and layout of your applications so that they render appropriately on smart phones as well as tablets.

Specifically, you can use CSS3 media queries in order to determine the following characteristics of a device:

- Browser window width and height
- Device width and height
- Orientation (landscape or portrait)
- Aspect ratio
- Device aspect ratio
- Resolution

CSS3 media queries are Boolean expressions that contain one or more "simple terms" (connected with and or or) that evaluate to true or false. Thus, CSS3 media queries represent conditional logic that evaluates to either true or false.

As an example, the following link element loads the CSS stylesheet mystuff.css only if the device is a screen and the maximum width of the device is 480px:

```
<link rel="stylesheet" type="text/css"
    media="screen and (max-device-width: 480px)"
            href="mystuff.css"/>
```

The preceding link contains a media attribute that specifies two components: a media type of screen and a query that specifies a max-device-width whose value is 480px. The supported values for media in CSS3 media queries are braille, embossed, handheld, print, projection, screen, speech, tty, and tv.

The next CSS3 media query checks the media type, the maximum device width, and the resolution of a device:

```
@media screen and (max-device-width: 480px) and (resolution: 160dpi) {
 #innerDiv {
  float: none;
 }
}
```

If the CSS3 media query in the preceding code snippet evaluates to true, then the nested CSS selector will match the HTML element whose id attribute has the value innerDiv, and its float property will be set to none on any device whose maximum screen width is 480px. As you can see, it's possible to create compact CSS3 media queries that contain non-trivial logic, which is obviously very useful because CSS3 does not have any if/then/else construct that is available in other programming languages.

Additional Code Samples on the CD

The CSS stylesheet CSS3MediaQuery1.css and the HTML5 Web page CSS3MediaQuery1.html illustrate how to use media queries in order to change the size of two images when users rotate their mobile device.

You can detect a change of orientation of a mobile device using simple JavaScript code, so you are not "forced" to use CSS3 media queries. The HTML5 Web page CSS3OrientationJS1.html on the CD illustrates how to use standard JavaScript in order to change the size of two images when users rotate their mobile device.

In essence, the code uses the value of the variable window.orientation in order to detect four different orientations of your mobile device. In each of those four cases, the dimensions of the JPG files are updated with the following type of code:

```
document.getElementById("img1").style.width  = "120px";
document.getElementById("img1").style.height = "300px";
```

Although this is a very simple example, hopefully this code gives you an appreciation for the capabilities of CSS3 Media Queries.

Summary

This chapter showed you how to create graphics effects, shadow effects, and how to use transforms in CSS3. You learned how to create animation effects that you can apply to HTML elements, and you saw how to define CSS3 selectors to do the following:

- Render rounded rectangles
- Create shadow effects for text and 2D shapes
- Create linear and radial gradients
- Use the methods translate(), rotate(), skew(), and scale()
- Create CSS3-based animation effects

ANIMATION EFFECTS WITH JQUERY AND CSS3

This chapter shows you how to create HTML5 Web pages that create animation effects and also provide interactivity for users. You'll see an assortment of jQuery functions that create various animation effects, which you can easily incorporate in your HTML5 Web pages. This eclectic chapter is intended to provide you with many animation effects, along with an assortment of code samples and code fragments that you can incorporate into your other HTML5 Web pages.

The first part of this chapter shows you how to use jQuery in order to manipulate the attributes of an element by setting the values of properties in CSS3 selectors, along with examples of creating animation effects using animate and effect. You'll see code examples that create slide-based and fade-related (fadeIn, fadeOut, fadeTo) animation effects. This section also illustrates how to create 2D animation effects using jQuery together with CSS3 keyframes.

The second part of this chapter illustrates how to create a "follow the mouse" HTML5 Web page that uses CSS3 for the visual effect and jQuery for updating the location of the gradient-filled rectangle. Remember that the CSS3 examples in the chapters of this book are specifically for WebKit-based browsers, but you can modify the code samples to include vendor-specific prefixes so that the code samples will run in other browsers. The last example in this chapter illustrates how to render SVG with jQuery using a jQuery plugin.

Note that this chapter covers jQuery animation effects and chapter seven contains some corresponding animation effects using jQuery Mobile, and that both chapters contain CSS3-based animation effects as well. Due to space constraints, this chapter covers only a portion of the animation-related functionality that is available in jQuery. You can learn more about jQuery animation by reading the online jQuery documentation or by performing an Internet search for additional tutorials and articles.

Working with CSS3 Selectors in jQuery

This section contains code samples that illustrate how to use jQuery to programmatically create HTML elements and to style them with CSS3 to create various effects, such as rounded corners, gradients, and shadow effects. By manually creating the required CSS3 selectors and the HTML elements, you can leverage the power of jQuery to create even more sophisticated visual effects.

Basic Animation Effects in jQuery

This section contains code fragments rather than complete code listings, and these

hide() and show(), and also how to set the animation speed. You'll also learn how to use the jQuery toggle() function to toggle CSS properties.

Keep in mind that this section covers only a portion of the rich set of functionality that is available with jQuery functions that create animation effects. Be sure to read the jQuery online documentation to learn about many other features that are supported.

The jQuery hide() and show() functions enable you to change the visibility of elements in an HTML page. For example, the following code block hides the second button when users click the first button; when users double click on the first button, the second button becomes visible:

```
$("#myButton1").click(function() {
  $(#myButton2").hide();
});
$("#myButton1").dblclick(function() {
  $(#myButton2").show();
});
```

The jQuery toggle function can handle two or more occurrences of the same event on an element. For example, the following code fragment handles one, two, or three click events on an element with the specified id value:

```
$("#myDiv1").toggle(
    function() {
      $("myText").text("First click");
    },
    function() {
      $("#myText").text("Second click");
    },
    function() {
      $("#myText").text("Third click");
    }
});
```

Two other useful jQuery methods are removeClass() and addClass(), which remove or add CSS classes, respectively, and an example is here:

```
$("#myDiv1").toggle(
    function() {
      $("myText").addClass("shiny");
    },
    function() {
      $("#myText").addClass("dark");
    }
```

The jQuery addClass() method adds a CSS class, whereas the toggleClass() method "toggles" the CSS class; i.e., the class is added if it's not present, and it's removed if it is already included.

Other related jQuery methods include fadeClass(), which uses a fading effect, and slideClass(), which creates a sliding effect.

Using Callback Functions

Many jQuery functions (including hide(), show(), toggle(), and slide-related functions) enable you to specify a callback function that is executed after the specified action is completed. For example, you can define the following code block:

```
$("#myButton1").click(function() {
  $(#myButton2").hide('slow',
    function callback() {
     // do something else
    });
});
```

When users click on an element whose id is myButton1, the preceding code block slowly hides the element whose id is myButton2, and after the animation is completed, the code in the callback function is executed.

This book contains WebKit-based code samples, but if you decide to use the preceding code block with IE, keep in mind that there is a bug involving the use of named functions in callbacks in IE. The following article provides useful information regarding named functions:

http://kangax.github.com/nfe

jQuery Fade and Slide Animation Effects

The jQuery fade-related and slide-related effects are easy to create, and if you define them appropriately, they can create very nice visual effects in your Web pages.

The following example combines the jQuery, .fadeIn(), and .fadeOut() functions and also shows you how to chain these functions so that you can create multiple animation effects.

The **fadeIn(), fadeOut(),** and **fadeToggle()** Functions

The three jQuery functions .fadeIn(), fadeOut (), and fadeToggle() methods can specify three parameters, and their syntax is shown here:

```
jQuery(list-of-elements).fadeIn(speed);
jQuery(list-of-elements).fadeOut(speed);
jquery(list-of-elements).fadeTo(speed);
```

The following code block illustrates how to use these three jQuery functions:

```
$("#something").click(function() {
  $(this).fadeIn('slow');
```

```
$(“#something”).click(function() {
  $(this).fadeOut(‘slow’);
});
$(“#something”).click(function() {
  $(this).fadeTo(‘slow’, .65);
});
```

Listing 3.1 displays the contents of FadeInOut.html, and illustrates how to perform simple and chained fading effects in jQuery.

LISTING 3.1 FadeInOut.html

```
<!DOCTYPE html>
<html lang=“en”>
 <head>
  <meta charset=“utf-8” />
  <title>jQuery Fade-Related Effects</title>

  <script src=“http://code.jquery.com/jquery-2.0.0b1.js”>
  </script>
  <script
   src=“http://code.jquery.com/jquery-migrate-1.1.0.js”>
  </script>
 </head>

 <body>
  <div>
   <div> <p>Hello World from jQuery (hover on me)</p> </div>
    <div> <p>Goodbye World from jQuery (hover on me)</p> </div>
   <div>
     <input type=“button” id=“button1” value=“Hide Me” />
   </div>
  </div>

  <script>
   $(document).ready(function(){
    var para = $(“div > p”);

     para.each(function(){
       var p = $(this);
```

```
            '>This Text Will Fade on Hover</span>');
        });

        para.hover(function() {
          // you can use 'slow' or 'fast'
         //$(this).find("span").fadeIn("slow");
          $(this).find("span").fadeIn(3000).fadeOut("fast")
                        .fadeIn("slow");
        }, function() {
         //$(this).find("span").hide();
          $(this).find("span").fadeOut(2000).fadeIn("slow")
                        .fadeOut("fast").fadeIn(2000)
                        .fadeOut("slow");
        });

        $("#button1").click(function() {
          $(this).fadeOut(500, function() {
         //$(this).remove();
          });
        });
      });
    </script>
  </body>
</html>
```

Listing 3.1 starts with two HTML elements and an HTML <button> elements, followed by jQuery code for applying fade-related effects to the HTML <p> elements. The first point to notice is the JavaScript variable para that stores a reference to the HTML <p> elements that are direct child elements of HTML elements, as shown here:

```
$(document).ready(function(){
  var para = $("div > p");
  // code omitted
}
```

The next code block dynamically adds an HTML element to the HTML <p> elements that are referenced in the para variable:

```
para.each(function(){
  var p = $(this);
  p.append('<span style="color:red;font-size:18px"'+
```

```
        '>This Text Will Fade on Hover</span>');
});
```

When users hover over any of the HTML <p> elements, the jQuery code creates multiple fade-related effects for the HTML <p> elements using jQuery method chaining. An example is shown below:

```
$(this).find("span").fadeOut(2000).fadeIn("slow")
                .fadeOut("fast").fadeIn(2000)
                .fadeOut("slow");
```

You can use jQuery methods to apply effects to elements other than the element that has the current focus. For example, if you want to hide a sibling element during a hover event, you can do something like this:

```
$(this).next().fade();
```

The next section shows you how to use jQuery slide-related functions in order to create slide-related animation effects.

jQuery Slide-Related Functions

The jQuery slideUp(), slideDown(), and slideToggle() methods can specify three parameters, and they have the following syntax:

```
jQuery(elements).slideUp([milliseconds], [easing-function],
                [callback-function]);


jQuery(elements).slideDown([milliseconds],
                [easing-function],
                [callback-function]);


jQuery(elements).slideToggle([milliseconds],
                  [easing-function],
                  [callback-function]);
```

The value milliseconds specifies the duration of the animation effect, and the callback-function is an optional JavaScript function that is executed after the animation is completed.

Listing 3.2 displays the contents of JQSlideUpDown.html, and illustrates how to perform simple and chained sliding effects in jQuery.

LISTING 3.2 JQSlideUpDown.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
   <meta charset="utf-8" />
   <title>jQuery Slide-Related Effects</title>
```

```
 </script>
 <script
  src="http://code.jquery.com/jquery-migrate-1.1.0.js">
 </script>
</head>

<body>
 <div>
  <div> <p>Hello World from jQuery (hover on me)</p> </div>
  <div> <p>Goodbye World from jQuery (hover on me)</p> </div>
  <div>
   <input type="button" id="button1"
        value="Click to Slide Me Up and Hide Me" />
  </div>
 </div>

 <script>
  $(document).ready(function(){
   var para = $("div > p");

   para.each(function(){
     var p = $(this);
     p.append('<span style="color:red;font-size:18px">'+
          'This Text Will Slide on Hover</span>');
   });

   para.hover(function() {
     // you can use 'slow' or 'fast'
    //$(this).find("span").slideDown("slow");
     $(this).find("span").slideDown(3000).slideUp("fast")
                .slideDown("slow");
   }, function() {
    //$(this).find("span").hide();
     $(this).find("span").slideUp(2000).slideDown("slow")
                .slideUp("fast").slideDown(2000)
                .slideUp("slow");
   });
```

```
        $("#button1").click(function() {
          $(this).slideUp(2000, function() {
          //$(this).remove();
          });
        });
      });
    </script>
  </body>
</html>
```

Listing 3.2 is similar to Listing 3.1, except that slide-related jQuery methods are used instead of fade-related jQuery methods, and the description of this code sample is analogous to the description of Listing 3.1.

One point to keep in mind is that sliding effects do not always work as expected (some jerkiness may occur) for elements that have CSS padding or margin properties or a width property that is not set to a fixed width. Experiment with these scenarios to see if the resultant behavior is what you expect, or if it is acceptable for your Web pages.

Easing Functions in jQuery

jQuery supports a set of so-called "easing" functions that provide different types of animation effects. In general terms, an easing function uses some type of equation as the path for an animation effect. For example, you can use a linear equation to create animation with constant speed. You can also use a quadratic equation (a polynomial of degree two, whose general form is a*x*x+b*x+c) to create animation effects with acceleration or for quadratic Bezier curves, and cubic equations for cubic Bezier curves.

jQuery also provides easing functions for animation whose speed is more complex (slow, fast, slow) at different positions of an easing function. Before you search for jQuery plugins, it's well worth your time to explore the existing jQuery easing functions, some of which are: linear, easeInQuad, easeOutQuad, easeInCubic, easeOutCubic, easeInOutCubic, easeInSine, easeOutSine, easeInOutSine, easeInElastic, easeOutElastic, and easeInOutElastic.

You can find numerous links that provide an extensive set of demonstration of jQuery easing functions, including the one shown here:

http://jqueryui.com/demos/effect/easing.html

In addition, there are many jQuery plugins available for custom animation-related easing functions, or you can create your own jQuery plugin if you cannot find one that fits your needs.

The jQuery .animate() Method

The jQuery animate() method can take four parameters, and they look like this:

```
jQuery(elements).animate([properties],
                         [milliseconds],
```

[complete-function]);

The properties parameter contains the list of properties to animate, and the milliseconds parameter specifies the duration of the animation effect. The easing-function parameter specifies one of the easing functions discussed in the previous section, and the complete-function specifies the JavaScript callback function to execute when the animation effect has completed.

The HTML5 Web page JQAnimate1.html on the CD illustrates how to use the jQuery animate() function in order to create animation effects on two PNG files.

This HTML5 Web page contains two PNG images, along with jQuery click handlers for two HTML <button> elements. Whenever users click on the left button, the jQuery code decreases the PNG opacity from 1.0 to 0.25, shifts the image file 50 units to the right, and increases its height by 100 units during a five-second interval, as shown here:

```
$('#text1').animate({
   opacity: 0.25,
   left: '+=50',
   height: '+=100'
 }, 5000, function() {
   // Animation complete (do something else)
});
```

Whenever users click on the second button, the jQuery code performs similar animation effects on the right-side image file. Launch the code in Listing 3.2 and click the buttons to see the animation effects.

You can see a variation of the animation effects in Listing 3.2 in the HTML5 Web page JQAnimate2.html, as well as sequential and parallel animation effects in the HTML5 Web page JQAnimate2.html, both of which are available on the CD.

You can use the jQuery animate() function to create other interesting visual effects by changing different CSS properties of HTML elements. For example, the following code block create a "wobbling" effect with list items that are part of an HTML <ul> element:

```
$('#mylist li').hover(function() {
  $(this).animate({paddingLeft: '+=15px'}, 200);
}, function() {
  $(this).animate({paddingLeft: '-=15px'}, 200);
});
```

You can also create linear and swing animation effects, as shown here:

```
$('p:first').toggle(function() {
```

```
}, function() {
  $(this).animate({'height':'-=150px'}, 1000, 'swing');
});
```

Custom CSS Animation Using the .animate() Function

You can use jQuery to animate many CSS properties, including border-width, bottom, font-size, height, margin, opacity, padding, right, top, width, and word-spacing. In addition, you can specify the duration with slow, fast, or an integer value that represents milliseconds.

The following code animates the width and height attributes of an HTML element so that their final values will be 500 and 300, respectively:

```
$("#myDiv").click(function() {
  $(this).animate({
    width:'500px', height: '300px'
  });
});
```

In the preceding jQuery code snippet, the width will increase to 500px if its initial value is less than 500px; otherwise it will decrease the width to 500px. (The same holds true for the height attribute).

CSS3-Based Animation Effects

This section illustrates a variety of animation effects that you can create with CSS3, where the code samples use CSS3 @keyframes rules and 2D/3D transforms.

Animation Effects with CSS3 Keyframes and 2D Transforms

Listing 3.3 displays the contents of JQButtonAnimation1.html, and illustrates how to create button-related animation effects that are triggered by the hover pseudo-class.

LISTING 3.3 JQButtonAnimation1.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>jQuery Button Animation Effect</title>

    <script src="http://code.jquery.com/jquery-2.0.0b1.js">
    </script>
    <script
      src="http://code.jquery.com/jquery-migrate-1.1.0.js">
    </script>

    <style>
```

```
  0% {
    font-size: 18px;
    background-color: #0f0;
    -webkit-transform: translate(0px,0px) rotate(-60deg)
                skew(-15deg,0);
  }
  25% {
    font-size: 24px;
    background-color: #0ff0
    -webkit-transform: translate(100px,100px)
                rotate(-180deg) skew(-15deg,0);
  }
  50% {
    font-size: 32x;
    -webkit-transform: translate(50px,50px) rotate(-120deg)
                skew(-25deg,0);
    background-color: #00f;
  }
  75% {
    font-size: 24px;
    background-color: #0ff;
    -webkit-transform: translate(100px,100px)
                rotate(-180deg)
                skew(-15deg,0);
  }
  100% {
    font-size: 18px;
    -webkit-transform: translate(0px,0px) rotate(0)
                skew(0,0);
    background-color: #f00;
  }
}

#button1 {
 font-size: 12px;
 background-color: #f00;
}
```

```
  #button1:hover {
   font-size: 36px;
   background-color: #00f;
  -webkit-animation-name: AnimButton;
  -webkit-animation-duration: 4s;
  }

  #button2 {
   font-size: 12px;
   background-color: #00f;
  }

  #button2:hover {
   font-size: 24px;
   background-color: #00f;
  -webkit-animation-name: AnimButton;
  -webkit-animation-duration: 2s;
  }
 </style>
</head>

<body>
  <div>
     <input type=“button” id=“button1”
          value=“Click Me or Hover Over Me” />
  </div>
  <div>
     <input type=“button” id=“button2”
          value=“Click Me or Hover Over Me” />
    </div>
  </div>

  <script>
   $(document).ready(function(){
     $(“#button1”).click(function() {
       $(this).fadeOut(500, function() {
       //$(this).remove();
```

```
      });

      $("#button2").click(function() {
        $(this).fadeOut(500, function() {
        //$(this).remove();
        });
      });
    });
  </script>

 </body>
</html>
```

Listing 3.3 contains an HTML <style> element with a CSS3 keyframes definition (which you could also move to a separate CSS stylesheet), followed by two HTML <input> elements of type button, both of which have click event handlers defined in the <script> element in Listing 3.3.

Whenever users click on either button, the CSS3 keyframes definition is applied to the button. This in turn creates animation effects using combinations of the functions translate(), rotate(), and skew() for the time periods (either 2 seconds or 4 seconds) specified in the associated selectors. In addition, the click handlers create a fading effect that lasts for 500 milliseconds.

When you launch the HTML Web page JQButtonAnimation1.html in a browser, you will see the animation effect whenever you hover over either of the buttons.

2D Transforms with CSS3 and jQuery

The code sample in this section shows you how to apply CSS transforms directly to elements (based on user-initiated events) using the jQuery css() function.

Listing 3.4 displays the contents of JQTransforms2D1.css, which contains CSS3 selectors that are applied to the HTML5 Web page JQTransforms2D1.html, shown in Listing 3.4.

LISTING 3.4 JQTransforms2D1.css

```
#outer {
  position: absolute;
  left: 50px;
  top: 150px;
}

#inner1 {
  float: left;
  background-color:#F00;
```

```
  height:150px;
}

#inner2 {
  float: left;
  background-color:#FF0;
  width: 200px;
  height:150px;
}

#inner3 {
  float: left;
  background-color:#00F;
  width: 200px;
  height:150px;
}
```

Listing 3.4 is very straightforward: several properties, such as the width and height, are specified for three HTML elements that are defined in Listing 3.5.

LISTING 3.5 JQTransforms2D1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>jQuery Transform Effects</title>
  <link href="Transforms1.css" rel="stylesheet" type="text/css">

   <script src="http://code.jquery.com/jquery-2.0.0b1.js">
   </script>
   <script
     src="http://code.jquery.com/jquery-migrate-1.1.0.js">
   </script>
</head>

<body>
  <h1>Click Inside Any of the Rectangles:</h1>

  <div id="outer">
```

```
  <div id="inner2"></div>
  <div id="inner3"></div>
  </div>
</div>

<script>
 $(document).ready(function() {
   $("#inner1").click(function() {
     $("div").css({height: '300px',
       'webkit-transform': 'scale(0.5, 0.5) skew(-10deg,
                                    20deg)'
     });
   });

   $("#inner2").click(function() {
     $("div").css({height: '200px',
                width: '250px',
      'webkit-transform': 'scale(0.5, 0.8) rotate(-45deg)'
     });
   });

   $("#inner3").click(function() {
     $("div").css({height: '100px',
                width: '250px',
      'webkit-transform': 'skew(-10deg, 10deg) rotate(-45deg)'
     });
   });
 });
</script>

</body>
</html>
```

Listing 3.5 defines event handlers for the click event for three HTML elements, all of which invoke the jQuery css() function in order to update properties of the element that received a click event.

For example, the first HTML element is updated as follows whenever users click on this element:

```
        'webkit-transform': 'scale(0.5, 0.5) skew(-10deg, 20deg)'
    });
```

As you can see, the height property is set to 300px, and transforms are applied to the elements when users click on them.

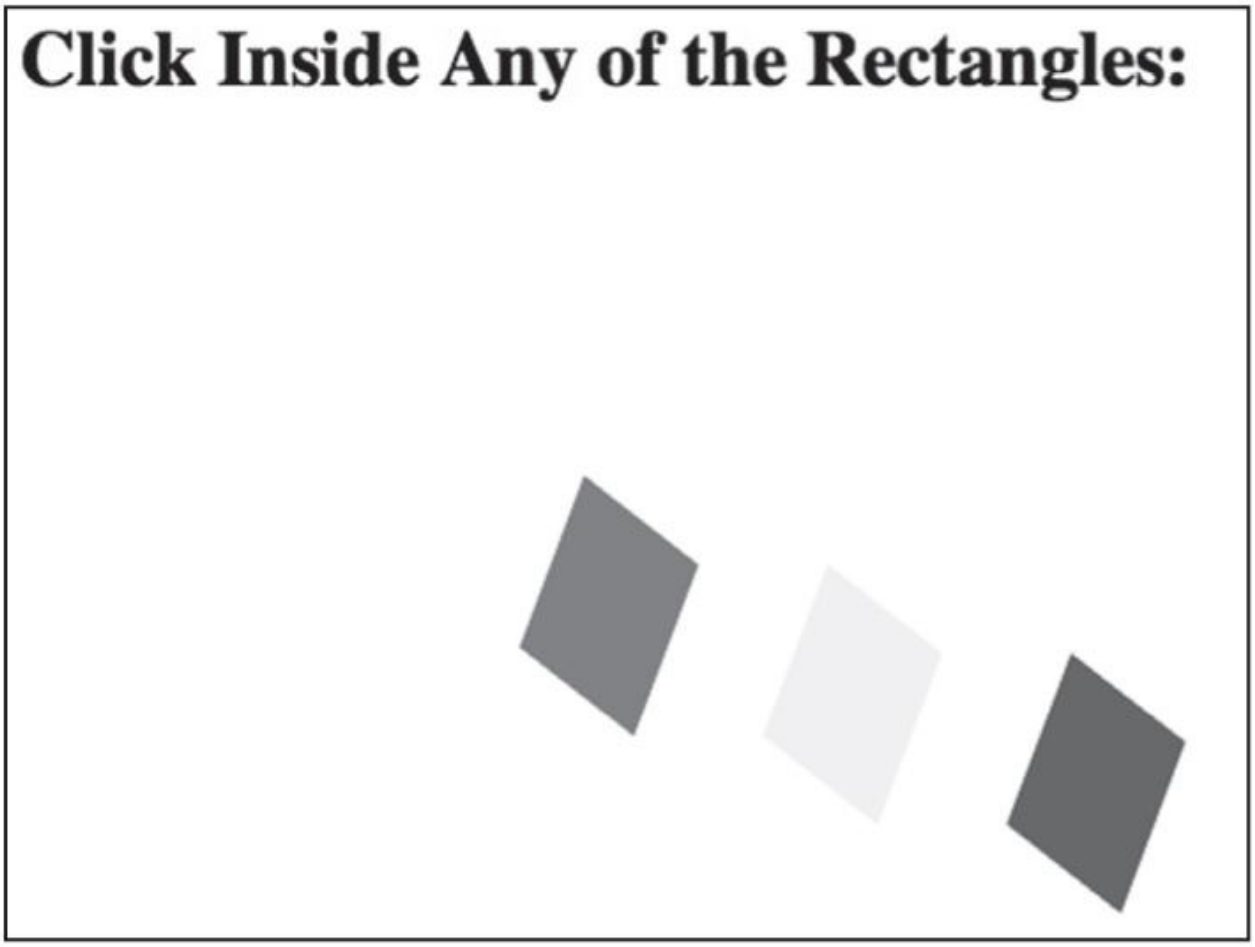
Keep in mind that you can move the CSS3 code, referenced in the click handlers in Listing 3.5, to a separate CSS stylesheet. This makes it easier to maintain the CSS3 code in a single file, and you can also reference the same CSS stylesheet in multiple HTML Web pages.

In addition to CSS3 2D animation effects, you can obviously create CSS3 3D animation effects. Experiment with the code in Listing 3.5 by adding some of the 3D effects that are available in code chapters in Chapter 2, or from the following open source project:

http://code.google.com/p/css3-graphics

Figure 3.1 displays JQTransforms2D1.html in the Chrome browser on a MacBook.

The remainder of this chapter contains examples that are different from the previous code samples. Listing 3.4 and Listing 3.5 involve mouse-related functionality and how to handle mouse events programmatically. Listing 3.6 in the next section shows you a very rudimentary game-oriented code sample that combines JavaScript, jQuery, and CSS3 more extensively than earlier examples.



A Follow-the-Mouse Example with jQuery

The code sample in this section extends the functionality introduced in the previous section by showing you how to programmatically create HTML elements and append them to the DOM.

Listing 3.6 displays the contents of JQSketchFollowMouse1.html that illustrates how to render a element under the current location of a user's mouse using jQuery and CSS3 in an HTML5 Web page.

LISTING 3.6 JQSketchFollowMouse1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>jQuery Follow the Mouse Example</title>

   <script src="http://code.jquery.com/jquery-2.0.0b1.js">
   </script>
   <script
     src="http://code.jquery.com/jquery-migrate-1.1.0.js">
   </script>
</head>

<body>
  <div></div>

  <script>
   // see commentary about the preceding empty <div> element
   $(document).ready(function() {
    var rectWidth  = 20;
    var rectHeight = 20;
    var moveCount  = 0;
    var insertNode = true;
    var currColor  = "";
    var rectColors = new Array('#ff0000', '#ffff00',
                               '#00ff00', '#0000ff');
    var newNode;

    $(document).mousemove(function(e) {
```

```
        // are users are moving their mouse?
        if(insertNode == true) {
          // create a rectangle at the current position
          newNode = $('<div id=newNode>').css({'position':'absolute',
                                'background-color':'#ff0000',
                                'width':rectWidth+'px',
                                'height':rectHeight+'px',
                                 top: e.pageY,
                                 left: e.pageX
                               });

          //append the rectangle to body with this code:
          $("div").append(newNode);

          //do not append the rectangle to body with this code:
          //$(document.body).append(newNode);

          insertNode = false;
        } else {
         currColor = rectColors[moveCount % rectColors.length];

          $('div').each(function() {
            $(this).css({top: e.pageY,
                         left: e.pageX,
                         'background-color': currColor
             });
          });
        }
      });
    });
  </script>
 </body>
</html>
```

Listing 3.6 initializes some JavaScript variables and then uses the jQuery `css()` method to dynamically create an HTML `<div>` element whose upper-left vertex has the same coordinates of the point where a `mousemove` event occurs, thereby creating a follow-the-

gives us access to the coordinates of the current move position. We can use the attributes e.pageX and e.pageY to set the CSS properties left and top, respectively.

Notice also the inclusion of an empty HTML element immediately after the HTML <body> element in Listing 3.6. This element is required because UI elements added after a script are known to disappear in some older browsers. The solution is to include the empty element and then append content to that element.

Handling Other Events with jQuery

You have seen code samples that illustrate how to use jQuery to handle various events. jQuery provides extensive support for mouse-related events and also support for keyboard events, as described in the next two sections.

Handling Mouse Events

jQuery supports the following common mouse events that are probably familiar to you: mousedown, mouseenter, mouseleave, mousemove, mouseout, mouseover, and mouseup. You can detect each of these mouse events in jQuery using the following jQuery code constructs:

```
$(“#myInput”).mousedown(function() {
  // do something
});
$(“#myInput”).mouseenter(function() {
  // do something
});
$(“#myInput”).mouseleave(function() {
  // do something
});
$(“#myInput”).mousemove(function() {
  // do something
});
$(“#myInput”).mouseout(function() {
  // do something
});
$(“#myInput”).mouseover(function() {
  // do something
});
$(“#myInput”).mouseup(function() {
  // do something
});
```

You can include any of the preceding code snippets that you need in your HTML Web pages and then add the processing logic to provide the intended functionality.

You can also detect keypress, keyup, and keydown events in jQuery. For example, this code displays an alert when users click on the uppercase "Z" key:

```
$("#myInput").keypress(function(e) {
  if (e.which == 90) alert ('Z was typed.')
});
```

You can check for other key events, keeping in mind that uppercase A is decimal 65 (hexadecimal 41), lowercase a is decimal 97 (hexadecimal 61), and lowercase z is decimal 122 (hexadecimal 7A).

You can also add many other effects, including the animation effects that are available in previous code samples in this chapter.

Additional Code Samples on the CD

The HTML5 Web page JQSlideShow1.html illustrates how to create a slideshow with JPG images using jQuery, which uses the following code block to determine the next HTML <img> element (which wraps around to the first image in the list when we reach the right-most image) and adds the show attribute, as shown here:

```
next = curr.next().length ? curr.next() :
        curr.parent().children(':first');
next.addClass("show");
```

The HTML5 Web page JQSketchSolid1.html on the CD illustrates how to create a rudimentary sketching program with jQuery and CSS3 in an HTML5 Web page.

Figure 3.2 displays JQSketchSolid1.html in the Chrome browser on a MacBook.

Figure 3.2 Sketching with jQuery in the Chrome browser on a MacBook.

along with other visual effects, and its homepage is here:

http://www.zachstronaut.com/projects/rotate3di/

You can also combine jQuery with SVG instead of dynamically creating elements that are styled with CSS3 selectors. The accompanying CD contains the HTML5 Web page JQArchDoubleEllipse1Rotate1.html that illustrates how to combine jQuery with SVG. However, this code is similar to earlier code samples, so its contents are omitted from this chapter.

The HTML5 Web page JQBouncingBalls1.html on the CD illustrates how to render a set of vertically bouncing balls in a jQuery HTML5 Web page. Figure 3.3 displays the result of rendering this HTML5 Web page on an Asus Prime tablet with Android ICS.

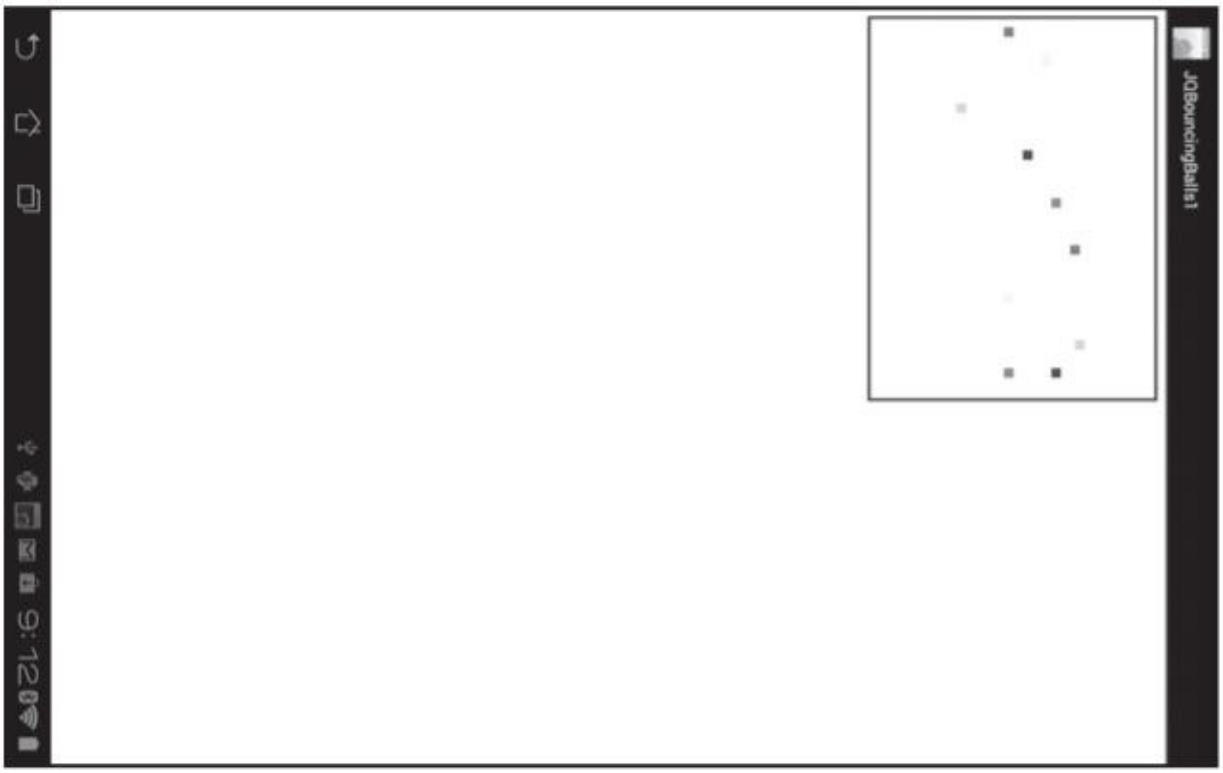


Figure 3.3 Bouncing balls on an Asus Prime tablet with Android ICS.

The jQuery plugin PFold provides a nice paper-folding effect, and it's downloadable here:

http://tympanus.net/Development/PFold/index.html

Animation: Comparing CSS3 with jQuery

The performance differences between CSS3 and jQuery are an interesting and obviously important topic, especially if you plan to develop an HTML Web page or application that contains many animation effects. This section provides three links with various tips and some performance benchmarks.

performance-related tips:

http://christianheilmann.com/2013/01/25/five-things-you-can-do-to-make-html5-perform-better/

The second link provides performance benchmarks (and it differs slightly from one of Christian's recommendations):

http://blog.tumult.com/2013/02/28/transform-translate-vs-top-left/?utm_source=html5weekly&utm_medium=email

The third link is in an article on Opera's Website, comparing the performance of CSS3 and jQuery animation:

"CSS3 wins the race by lengths. The huge difference in performance is because the browser's CSS processor is written in C++ and native code executes very fast whereas jQuery (JavaScript) is an interpreted language and the browser can't predict JavaScript ahead in time, in terms of what event will occur next."

The preceding quote is from the following Website:

http://dev.opera.com/articles/view/css3-vs-jquery-animations/#comments

The preceding Website also states the following:

"Although the results above indicate that you should use CSS3 for animations, you should bear in mind the advantages and disadvantages we discussed earlier on in the article. You need to keep in mind that a fair amount of people still use Internet Explorer 7 and 8, so you should use jQuery if your animations absolutely need to work the same in those older browsers."

The preceding links provide guidelines for improving the performance of your HTML5 Web pages. However, you might need to perform a more detailed analysis of the contents of your HTML5 Web page in order to fine-tune its performance. Perform an Internet search for articles that discuss performance issues in Web pages with similar animation effects.

Summary

This chapter introduced you to jQuery graphics and animation effects, along with code samples that illustrated how to use jQuery functions to create simple animation effects. In particular, you learned how to do the following:

- Basic animation effects in jQuery
- The Effect action in jQuery
- Scrolling effects in jQuery
- Working with CSS3 selectors in jQuery
- Setting properties with the css() function
- Toggling CSS properties
- Creating rounded corners
- Creating shadow effects
- Setting linear and radial gradients

- The hide() and show() functions
- Using callback functions
- The fadeIn(), fadeOut(), and fadeTo() methods
- Setting the speed
- Toggling hide() and show()
- jQuery Fade and Slide animation effects
- The fadeIn() and fadeOut() functions
- jQuery slideUp() and slideDown() functions
- Easing functions in jQuery
- Custom CSS animation using animate()
- Creating a slideshow with images
- CSS3-Based Animation Effects
- Simple Sliding Effects with CSS3
- Updating Multiple Attributes with CSS3
- Sliding Effects with CSS
- Animation Effects with CSS3 keyFrames and 2D Transforms
- 2D Transforms with CSS3 and jQuery
- 3D Transforms with CSS3 and jQuery
- A Follow-the-Mouse Example with jQuery
- Your First Sketching Program

The next chapter introduces you to various jQuery UI Controls, along with code samples that show how to render some jQuery UI controls in HTML5 Web pages.

jQuery UI Controls

This chapter introduces you to various jQuery UI controls, along with code samples that show how to render some jQuery UI controls in HTML5 Web pages. The rationale for using these UI controls is simple: they do not require nearly as much effort as writing your own custom UI controls, nor do you need to maintain the code for these controls. Moreover, these UI controls work well on a range of browsers.

You will see examples of using jQuery to render accordions, buttons, combo boxes, date pickers, progress bars, sliders, and tabs. In addition, you will learn how to programmatically handle user-initiated events involving jQuery UI controls. There are more UI controls available that you can learn about by consulting the jQuery homepage.

The jQuery UI controls in this chapter are presented alphabetically, so feel free to skip around to read about the UI controls that are of interest to you. Although information about these UI controls is available in the jQuery documentation, this is a primer book, so it's more appropriate to include a list of UI controls in one convenient location instead of telling you to "go read the documentation." After you have read this chapter, you will also be in a better position to understand the lengthy code sample at the end of the chapter. The sample illustrates an HTML5 Web page with various jQuery UI controls in a manner that reflects a somewhat realistic scenario.

This chapter also contains code samples for handling user click events that trigger updates in other (sometimes graphical) elements that are defined elsewhere in the same HTML5 Web page. This approach makes it easy to understand how to implement event-related functionality, and hopefully you will be able to adapt the code samples in this chapter to your specific needs.

One point to keep in mind is that the HTML5 Web pages in this chapter contain both HTML markup and jQuery code. For longer Web pages, it makes more sense to put jQuery code in a separate file (just as we have done with CSS stylesheets). However, almost every Web page in this chapter is short (between one page and 1.5 pages), so it's a choice based on convenience to keep the code in a single file.

A second point involves how to write your own jQuery plugins, which is beyond the scope of this book. You can find online tutorials that show you how to write jQuery plugins if you cannot find any existing jQuery plugins that meet your needs.

Using jQuery 2.0 in this Chapter

The following code samples work correctly on jQuery 1.5, but they do not work correctly with jQuery 2.0.0:

- JQUISlider1.html (no slider displayed)
- JQUISliderColors1.html (no slider displayed)
- JQUIThemes1.html (use plugin available on github)

Keep in mind that future versions of jQuery will undoubtedly resolve these issues and other inconsistencies that you might encounter in your own HTML Web pages that use jQuery.

Accordion Effects

jQuery UI supports an accordion widget, which contains one or more “folders” whose contents are shown only when users click on a particular folder.

Listing 4.1 displays the contents of the HTML5 Web page JQUIAccordion1.html, which illustrates how to render an accordion widget.

LISTING 4.1 JQUIAccordion1.html

```
<!DOCTYPE html>
<html lang=“en”>
 <head>
  <meta charset=“utf-8” />
  <title>jQuery Accordion</title>

  <link href=“JQUIAccordion1.css” rel=“stylesheet”
                                type=“text/css”>
  <link type=“text/css”
      href=“css/themename/jquery-ui-1.8.14.custom.css” />
  <script src=“http://code.jquery.com/jquery-2.0.0b1.js”>
  </script>
  <script src=“http://code.jquery.com/jquery-migrate-1.1.0.js”>
  </script>
  <script src=“http://code.jquery.com/ui/1.10.1/jquery-ui.js”>
  </script>

  <link href=“JQUIAccordion1.css”
      rel=“stylesheet” type=“text/css”>
 </head>

 <body>
  <div id=“accordion”>
    <h3><a href=”#”>Section 1</a></h3>
    <div>
```

```
    </div>

    <h3><a href="#">Section 2</a></h3>
    <div>
      <p> This is the second section of the accordion. </p>
      <ul>
        <li>List item one</li>
        <li>List item two</li>
        <li>List item three</li>
      </ul>
    </div>

    <h3><a href="#">Section 3</a></h3>
    <div>
      <p> This is the section of the third accordion. </p>
    </div>
    <h3><a href="#">Section 4</a></h3>
    <div>
      <p> This is the section of the fourth accordion. </p>
      <div id="outer">
        <div id="inner1"></div>
        <div id="inner2"></div>
        <div id="inner3"></div>
      </div>
    </div>
  </div>

  <script>
    $(document).ready(function() {
      $("#accordion").accordion();
    });
  </script>
 </body>
</html>
```

Listing 4.1 is straightforward: after the usual file references, there is an HTML <script> element that references a HTML element (whose id attribute has value accordion). This HTML element also contains 4 HTML elements that refer to four different

The <script> element contains jQuery code that renders the contents of this HTML element as an accordion in literally one line of code, shown here:

```
<script>
  $(document).ready(function() {
    $("#accordion").accordion();
  });
</script>
```

In fact, the code in the preceding <script> element is the typical manner in which jQuery renders the contents of an HTML element as a jQuery widget:

```
$("#theDivID").widgetType();
```

In this example, the widgetType function is the jQuery accordion() function. You can create different accordion effects by overriding some of the default CSS definitions for a jQuery accordion. For example, insert the following section of code before the first section of the accordion in Listing 4.1, and see how this changes the effect of selecting each section in the accordion:

```
<h3 class="ui-accordion-header ui-helper-reset ui-state-active ui-corner-top">
  <span class="ui-icon ui-icon-triangle-1-s"/>
  <a href="#">Section 1</a>
</h3>
```

LISTING 4.2 JQUIAccordion1.css

```
#inner1 {
  float: left;
  background-col33r:#F00;
  width: 200px;
  height:200px;
}

#inner2 {
  float: left;
  background-color:#FF0;
  width: 200px;
  height:200px;
}

#inner3 {
  float: left;
  background-color:#00F;
```

```
  height:200px;
}
```

The three selectors in Listing 4.2 match their corresponding HTML elements in Listing 4.1, which renders three rectangular shapes with red, yellow, and blue, respectively. This effect is visible when users click on the lowest folder (labeled “Section 4”) in the accordion. In a sense, an accordion can be viewed as a set of vertical tabs, where each tab contains whatever HTML content you want to render, including graphics-like effects.

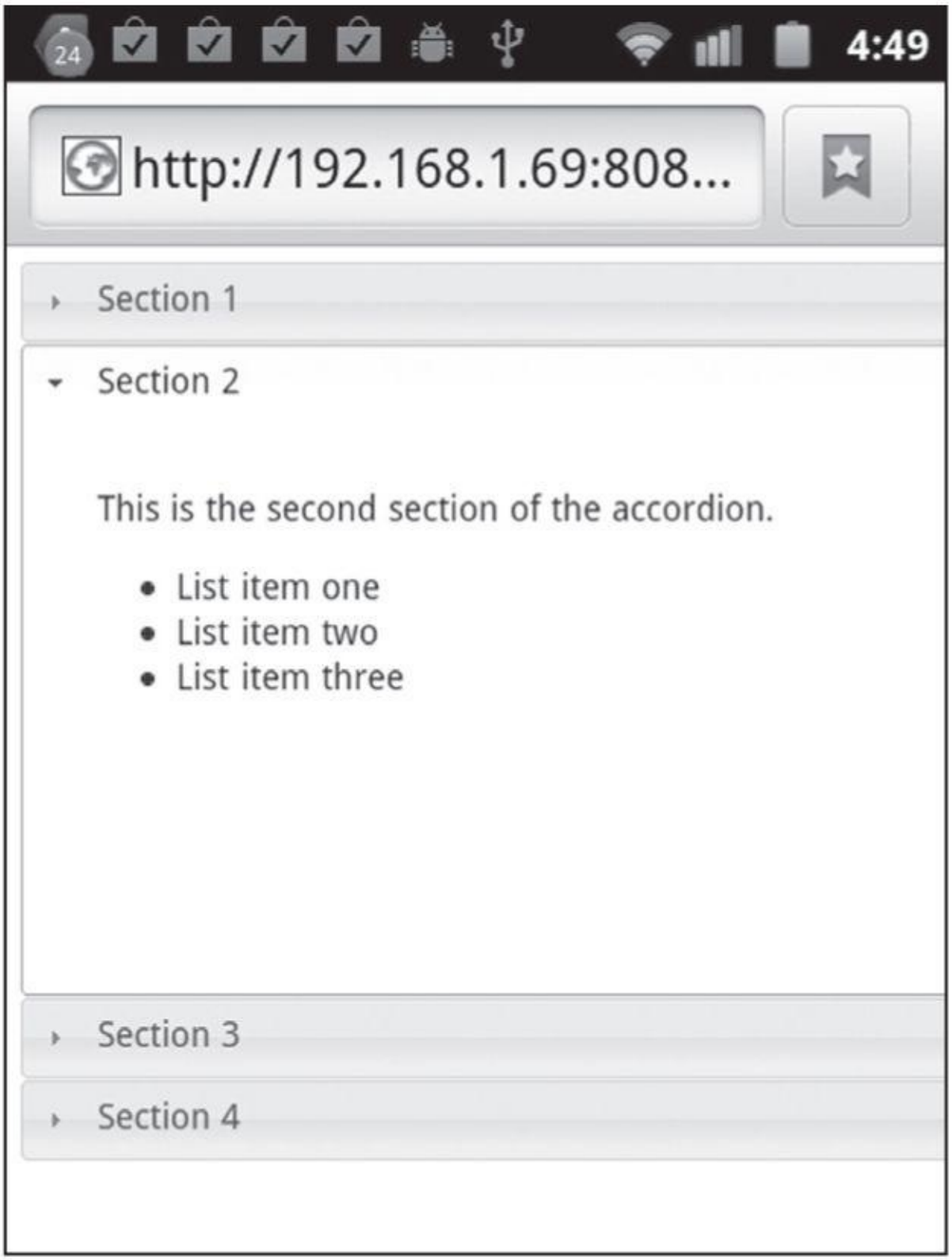


Figure 4.1 A jQuery accordion on a Nexus S 4G with Android ICS.

Launch the HTML5 Web page in Listing 4.1 and click on each "folder" in the rendered accordion, which will reveal the contents of the currently selected folder and also hide the contents of the other folders of this accordion.

Figure 4.1 displays the result of rendering JQUIAccordion1.html in Listing 4.1 in a landscape-mode screenshot taken from an Android application running on a Nexus S 4G with Android ICS.

Buttons

Buttons are obviously important in Web pages, especially for submitting form-based data, and jQuery provides significant support for button-related functionality. You can define CSS3 selectors to apply whatever styling effects you need to HTML buttons, including gradients and shadow effects.

Listing 4.3 displays the contents of the HTML5 Web page JQUIButtons1.html, which illustrates how to render buttons in an HTML Web page. The CSS stylesheet JQUIButtons1.css is omitted because its contents are the same as Listing 4.2.

LISTING 4.3 JQUIButtons1.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <title>jQuery Buttons</title>

  <link href="JQUIButtons1.css"
  rel="stylesheet" type="text/css">
  <link type="text/css" rel="Stylesheet"
       href="css/themename/jquery-ui-1.8.14.custom.css" />

  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script
    src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>

  <script src="http://code.jquery.com/ui/1.10.1/jquery-ui.js">
  </script>
 </head>
```

```
<div id="outer">
  <div class="buttons">
    <button>A regular button</button>
    <input type="submit" value="A submit button">
    <a href="#">An anchor button</a>
  </div>

  <div id="inner1"></div>
  <div id="inner2"></div>
  <div id="inner3"></div>
</div>

<script>
  $(function() {
    $( "input:submit, a, button", ".buttons" ).button();
  });
</script>

<script>
 $(document).ready(function() {
    var divColors = new Array('#000', '#F0F', '#F00',
                              '#0F0', '#00F', '#0FF');
    var clickCount = 0;
    var color1 = "";

    $("button").click(function() {
      ++clickCount;
      color1 = divColors[(clickCount) % divColors.length];
      $("#inner1").css({background: color1});
    });

    $("input").click(function() {
      ++clickCount;
      color1 = divColors[(clickCount) % divColors.length];
      $("#inner2").css({background: color1});
    });
```

```
        ++clickCount;
        color1 = divColors[(clickCount) % divColors.length];
        $("#inner3").css({background: color1});
      });
    });
  </script>

 </body>
</html>
```

Listing 4.3 contains an HTML <script> element that applies the jQuery button() method to three HTML elements and converts them into jQuery buttons with one line of code, as shown in this code snippet:

```
$("input:submit, a, button", ".buttons").button();
```

Notice that the class .buttons (which is used to style an HTML element that contains the HTML buttons and anchors) is also specified in the preceding code snippet. This extra class is redundant in this code snippet, but it shows you the flexibility of specifying a set of HTML elements that you want to convert into jQuery buttons.

The next block of code renders three HTML elements with colors that are specified in their corresponding selectors that are defined in the associated CSS stylesheet.

Launch the HTML Web page in Listing 4.3, and when you click on the top row of buttons (which includes the anchor link), the rectangles change colors. For example, the buttons change color whenever users click on them because of the following code:

```
$("button").click(function() {
   ++clickCount;
   color1 = divColors[(clickCount) % divColors.length];
   $("#inner1").css({background: color1});
});
```

In the preceding event handler, the colors are selected from a JavaScript array called divColors that contains a set of colors.

Figure 4.2 displays the result of rendering JQUIButtons1.html in Listing 4.3, in a landscape-mode screenshot taken from an iPad3.

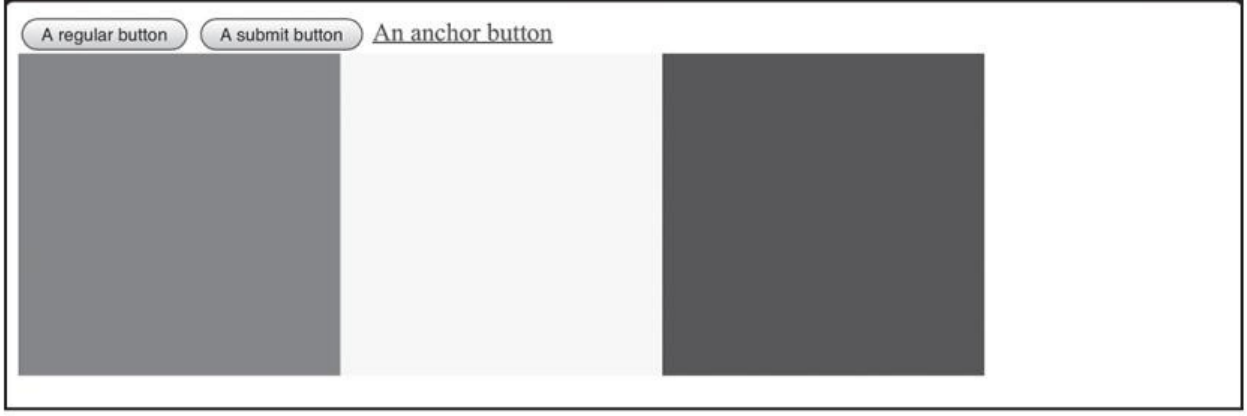


Figure 4.2 Buttons on an iPad3.

Check Boxes and Radio Buttons

jQuery enables you to add event handlers to HTML checkboxes and radio buttons so that you can determine which ones that users have selected.

Listing 4.4 displays the contents of JQUICheckBoxRadio1.html that illustrates how to render a checkbox and a set of radio buttons in an HTML Web page. The CSS stylesheet JQUICheckBoxRadio11.css is omitted because its contents are the same as Listing 4.2.

LISTING 4.4 JQUICheckBoxRadio1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>jQuery Checkbox and Radio ButtonsEffect</title>

  <link href="JQUICheckBoxRadio1.css"
      rel="stylesheet" type="text/css">
  <link type="text/css"
      href="css/themename/jquery-ui-1.8.14.custom.css" />

   <script src="http://code.jquery.com/jquery-2.0.0b1.js">
   </script>
   <script
    src="http://code.jquery.com/jquery-migrate-1.1.0.js">
   </script>

  <script src="http://code.jquery.com/ui/1.10.1/jquery-ui.js">
  </script>
ox
```

```
  #CheckBoxRadioInfo, #CheckBoxRadioInfo2 {
    font-size: 20px;
    top: 10px;
    width: 50%;
    height: 50px;
  }
 </style>
</head>

<body>
 <div id="outer">
  <div>
   <fieldset id="CheckBoxRadioInfo">
    <label for="checkbox1"><strong>Check Something:</strong>
                                    </label>
    <input type="checkbox" name="checkbox1" id="checkbox1">
                                    </input>

    <input type="radio" name="radio"
          value="radio1" checked="checked"></input>
    <input type="radio" name="radio" value="radio2"></input>
    <input type="radio" name="radio" value="radio3"></input>
   </fieldset>
  </div>

  <div>
   <fieldset id="CheckBoxRadioInfo2">
    <label for="input1"><strong>You Clicked On:</strong>
                                  </label>
    <input type="input" name="input1" id="input1"></input>
   </fieldset>
  </div>
  </div>
 </div>

 <script>
  $(document).ready(function() {
```

```
    $("#input1").val("checkbox1");
  });

  $("input[name='radio']").click(function() {
    $("#input1").val($(this).val());
  });
 });
</script>
</body>
</html>
```

Listing 4.4 contains two HTML elements that contain an HTML <fieldset> element, which specifies an HTML checkbox followed by HTML radio buttons.

You can find the state of the checkbox whose name attribute has the value checkbox1 with this event handler:

```
$("input[name='checkbox1']").click(function() {
  $("#input1").val("checkbox1");
});
```

Similarly, you can determine which radio button is checked with this code:

```
$("input[name='radio']").click(function() {
  $("#input1").val($(this).val());
});
```

In addition, you can also check which radio button is selected with the following code snippet:

```
var value = $("input[@name= fieldname] :checked" ).val();
```

In the preceding snippet, you need to replace fieldname with the corresponding name in the form field.

```
$('input:radio[ name="postage" ]').change( function(){
 if ($(this).is( ':checked' ) && $(this).val( ) == 'Yes') {
 // append goes here
 }
});
```

Figure 4.3 displays the result of rendering JQUICheckBoxRadio1.html in Listing 4.4 in a Chrome browser on a MacBook.

Figure 4.3 Checkboxes/button in the Chrome browser on a MacBook.

Combo Boxes

jQuery provides support for HTML combo boxes, and you can attach event handlers to detect events that are associated with those HTML elements.

Listing 4.5 displays the contents of JQUIComboBox1.html that illustrates how to render a combo box in an HTML Web page, and to execute a block of code whenever users select a different value in the combo box.

LISTING 4.5 JQUIComboBox1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>jQuery ComboBox</title>

  <link type="text/css"
       href="css/themename/jquery-ui-1.8.14.custom.css" />
  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>

  <script src="http://code.jquery.com/ui/1.10.1/jquery-ui.js">
  </script>

  <style>
   #ComboBoxInfo {
     position: relative;
     font-size: 20px;
     top: 10px;
```

```
    height: 50px;
   }
  </style>
</head>

<body>
  <div id="outer">
   <div>
    <fieldset id="ComboBoxInfo">
     <label for="ComboBox"><strong>My ComboBox:</strong></label>
     <select id="ComboBox" >
       <option value="1">Value 1</option>
       <option value="2">Value 2</option>
       <option value="3">Value 3</option>
       <optgroup label="Group1">
         <option value="4">Value 4</option>
         <option value="5">Value 5</option>
         <option value="6">Value 6</option>
       </optgroup>
     </select>

     <label for="selected1"><strong>You selected:</strong>
                                        </label>
    <input id="selected1" type="text" value="">
    </fieldset>
   </div>
  </div>

  <script>
    $(document).ready(function() {
      $("#ComboBox").change(function() {
        // display the selected value
        $("#selected1").val(
           $("#ComboBox option:selected").text());
      });
    });
  </script>
```

```
</html>
```

Listing 4.5 is straightforward: the first section references the required jQuery files, followed by an HTML <style> element, and then the definition of the items in a combo box.

Whenever users change the selected item in the combo box, this block of code is executed:

```
$(“#ComboBox”).change(function() {
   // display the selected value
   $(“#selected1”).val(
      $(“#ComboBox option:selected”).text());
});
```

Figure 4.4 displays the result of rendering JQUIComboBox1.html in Listing 4.5 in the Chrome browser on a MacBook.

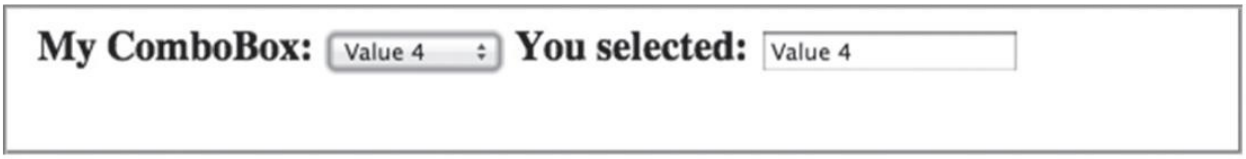


Figure 4.4 Combo box in the Chrome browser on a MacBook.

Date Pickers

jQuery supports “date picker” functionality that enables you to set the past, current, and future dates, as well as the ability to modify those dates in a contextually relevant manner.

Listing 4.6 displays the contents of the HTML5 Web page JQUIDatePicker1.html that illustrates how to render a jQuery datepicker widget in an HTML Web page. The CSS stylesheet JQUIDatePicker1.css is omitted because its contents are the same as Listing 4.2.

LISTING 4.6 JQUIDatePicker1.html

```
<!DOCTYPE html>
<html lang=“en”>
<head>
  <meta charset=“utf-8” />
  <title>jQuery Date Picker</title>

  <script
   src=“http://code.jquery.com/jquery-ui-1.8.14.custom.min.js”>
  </script>

  <script src=“http://code.jquery.com/jquery-2.0.0b1.js”>
```

```
  <script
    src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>

  <script src="http://code.jquery.com/ui/1.10.1/jquery-ui.js">
  </script>

  <style>
   #input1, #input2, #input3 {
     width: 150px;
     height: 50px;
     float: left;
   }
  </style>
</head>

<body>
  <div id="outer">
    <div id="input1">
      Last Week:
      <input type="text" name="date" id="date1" />
    </div>

    <div id="input2">
      Today's Date:
      <input type="text" name="date" id="date2" />
    </div>

    <div id="input3">
      A Future Date:
      <input type="text" name="date" id="date3" />
    </div>
  </div>

  <script>
    $(document).ready(function() {
      // set date to last week
```

```
                                    “defaultDate” );
    $(“#date1”).datepicker( “option”, “defaultDate”, -7 );

    // set date to today
    var defaultDate2 = $(“#date1”).datepicker(“option”,
                                    “defaultDate”);
    $(“#date2”).datepicker(“option”, “defaultDate”, +0 );

    // set date to the future
    var futureDate = “07/07/2017”;
    $(“#date3”).datepicker(“setDate”,futureDate);
  });
 </script>
</body>
</html>
```

Listing 4.6 references the usual jQuery files, followed by a <style> element that applies styling to the three HTML elements. Next, the HTML <body> element specifies the container of the three HTML elements, followed by a block of code that converts three HTML <input> fields into jQuery datepicker widgets, shown here:

```
$(“#date1”).datepicker();
$(“#date2”).datepicker();
$(“#date3”).datepicker();
```

You can specify a future date and when you want to change that future date, jQuery will display the previous and next months that are relative to that date, as shown here:

```
// set date to the future
var futureDate = “07/07/2017”;
$(“#date3”).datepicker(“setDate”,futureDate);
```

Figure 4.5 displays the result of rendering JQUIDatePicker1.html in Listing 4.6 in the Chrome browser on a MacBook.

Figure 4.5 A date picker in the Chrome browser on a MacBook.

Progress Bars

You can create and update progress bars very easily using jQuery. Listing 4.7 displays the contents of JQUIProgressBar1.html and illustrates how to render a progress bar in an HTML Web page.

LISTING 4.7 JQUIProgressBar1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>jQuery Progress Bar</title>

  <link rel="stylesheet"
type="text/css"  href="http://ajax.googleapis.com/ajax/libs/jqueryui/1.8/themes/base/jquery-ui.css"/>

  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script
   src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>

  <script src="http://code.jquery.com/ui/1.10.1/jquery-ui.js">
  </script>

  <style>
```

```
    width: 30%;
   }
  </style>
 </head>

 <body>
  <div id=“outer”>
   <div id=“progressBar1”>
    Progress Bar:
    <div id=“progressBarDiv1”> </div>
   </div>

   <div id=“value1”>
    Progress Bar Value:
    <input id=“text1” type=“text” value=“0” />
   </div>

   <div id=“newValue”>
    Set New Value:
    <input id=“newVal” type=“text” value=“0” />
   </div>

  <script>
   $(document).ready(function() {
   $(“#progressBarDiv1”).progressbar({ value: 40 });
     $(“#text1”).val($(“#progressBarDiv1”).progressbar(“value”));

     $(“#newValue”).bind(“change”, function() {
      var newVal = $(“#newVal”).val();
      $(“#progressBarDiv1”).progressbar(“option”, “value”,
                                parseInt(newVal));
     });
   });
  </script>
 </body>
</html>
```

Listing 4.7 references the usual jQuery files, followed by an HTML element

of the progress bar (which is initialized to 40), and an input field that allows users to change the value of the status bar.

The creation, display, and update of the value of the progress bar is handled in the following code block:

```
$(document).ready(function() {
  $(“#progressBarDiv1”).progressbar({ value: 40 });

  $(“#text1”).val($(“#progressBarDiv1”).progressbar(“value”));

  $(“#newValue”).bind(“change”, function() {
    var newVal = $(“#newVal”).val();

  $(“#progressBarDiv1”).progressbar(“option”, “value”,
                          parseInt(newVal));
  });
});
```

As you can see, the preceding code block binds a change event to the second HTML <input> field. When users enter a new value, the progress bar is updated with that value.

Figure 4.6 displays the result of rendering JQUIProgressBar1.html in Listing 4.7 in the Chrome browser on a MacBook.

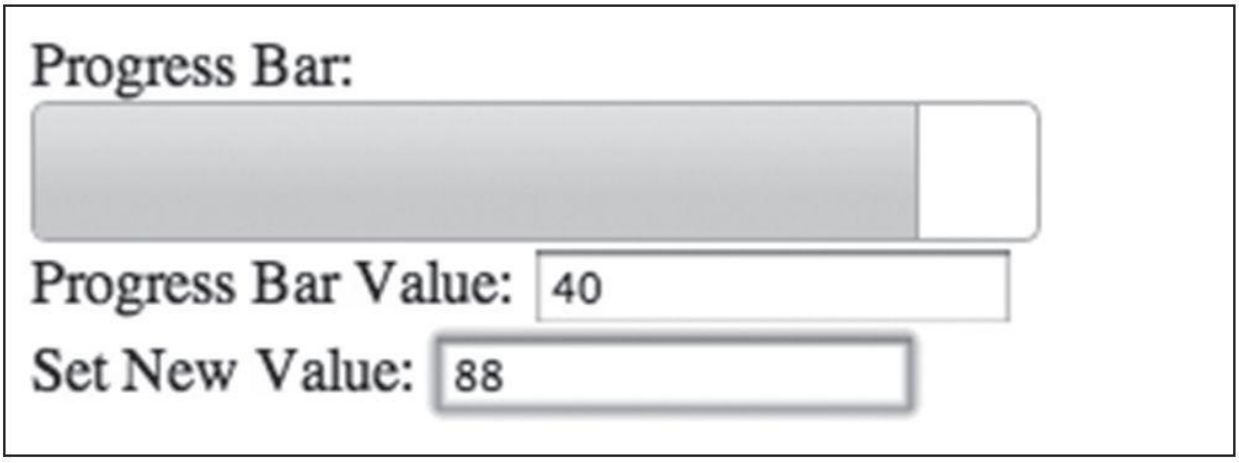


Figure 4.6 Progress bar in the Chrome browser on a MacBook.

Additional Code Samples on the CD

The HTML5 Web page JQUISlider1.html illustrates how to render horizontal and vertical sliders in an HTML Web page, and a key block of code is shown here:

```
$(document).ready(function() {
   $(“#sliderDiv1”).slider({
```

```
            min: 0,
            max: 200,
            step: 10,
            value: 100
        });
    }
```

One simple use of jQuery sliders is to use them to control the speed of animation effects (which can also be done with a spinner).

The HTML5 Web page JQUIAnimSlider1.html illustrates how to use a slider widget to control an animation effect.

Currently, jQuery does not provide scroll pane widgets, but you still have some options available. One option is to use HTML5 progress bars, which are simple to use and they provide some "reasonable" functionality. If HTML5 progress bars are insufficient for your needs, you may use a jQuery plugin that supports scroll pane widgets, such as the jScrollPane plugin whose homepage is here:

http://jscrollpane.kelvinluck.com/

The jScrollPane plugin provides the jScrollPane-<version>.min.js JavaScript file, as well as a CSS stylesheet (jScrollPane.css) that you can override with your own customizations.

This plugin provides over 20 demos that show you how to create scroll bars (horizontal, vertical, or both), the use of arrow buttons, its scrollTo and scrollBy methods, how to style scrollbars in an IFRAME, and various other effects.

The jQuery tab widget supports various options, methods, and events, and the HTML5 Web page JQUITabs.html illustrates how to render tabs in an HTML Web page.

jQuery UI provides very nice support for switching between different themes, which consist of different look-and-feel effects that are applied to widgets and text. You can find very good information about jQuery themes on these Websites:

http://jqueryui.com/docs/Theming

http://jqueryui.com/docs/Theming/API

http://jqueryui.com/docs/Theming#Using_ThemeRoller_and_Themes

The HTML5 Web page JQUIThemes1.html illustrates how to dynamically apply different themes to an HTML Web page with a set of widgets:

http://midnightprogrammer.net/post/Change-Page-Themes-Dynamically-Using-JQuery-Theme-Roller.aspx

NOTE *The theme switcher is no longer supported as of jQuery UI 1.9, but you can use the "Super-Theme-Switcher" that is available on* github*:*
https://github.com/harborhoffer/Super-Theme-Switcher

Create “Exploding” Effects

The following example is purely for fun, and you might even find a good use for the jQuery effect() function, which enables you to create some nice visual effects.

The HTML5 Web page JQUIEffects1.html illustrates how to horizontally shake and then “explode” an HTML element and its HTML <p> child elements.

The <script> element defines a click handler for the container HTML element by means of method chaining that uses the jQuery effect() method, as shown here:

```
$(“#outer”).effect(‘shake’, {times:3}, 300)
        .effect(‘highlight’, {}, 3000)
        .hide(‘explode’, {}, 1000);
```

When users click on this element, it will first shake horizontally three times (each time for 300 milliseconds), then change the color to a highlight effect. After 3 seconds, it will create an “exploding” effect that lasts for one second.

The HTML5 Web page JQUITravel1.html illustrates how to use some of the UI components that you saw earlier in this chapter. The validation details have been omitted in order to illustrate how to manipulate the UI components in a code sample of reasonable length, using techniques that you can use in your own Web pages.

Useful Links

The jQuery UI Website provides extensive documentation and details regarding jQuery UI components. Its homepage is here:

http://jqueryui.com/

Another useful jQuery UI link is a Website that provides useful demos of jQuery UI components:

http://jqueryui.com/demos/

You can use the jFormer jQuery plugin with HTML5 forms, and its homepage is here:

https://www.jformer.com/

A collection of jQuery plugins for playing audio and video files is here:

http://superdit.com/2011/04/27/12-jquery-plugins-for-playing-audio-video-files/

The jQuery plugin jTweetsAnywhere for displaying tweets is here:

http://thomasbillenstein.com/jTweetsAnywhere/

Summary

This chapter introduced you to jQuery UI controls, along with code samples that illustrated how to create HTML5 Web pages with jQuery UI controls. You learned how to use the following jQuery UI controls:

- Accordion effects

- Check boxes and radio buttons
- Combo boxes
- Date pickers
- Dialog boxes
- Progress bars
- Scroll panes
- Sliders
- Tabs
- Theming jQuery UI
- ThemeRoller and themes

OTHER HTML5 TECHNOLOGIES

This chapter provides an overview of several HTML5-related technologies and code samples for many of these technologies. Although you can use the HTML5 technologies in this chapter without jQuery, it makes sense to discuss jQuery plugins for various HTML5 technologies. Since new jQuery plugins are continually being written, compare the latest jQuery plugins with the plugins that are covered in this chapter.

The HTML5 technologies in this chapter are presented according to their W3C status: W3C Recommendation (REC), Candidate Recommendation (CR), Last Call (LC), Working Draft (WD), and experimental APIs. The HTML5 technologies in each section are presented alphabetically. This approach enables you to quickly determine the status of each HTML5 technology, which is the main advantage over using a pure alphabetical listing of HTML5 technologies.

The first part of this chapter briefly describes the stages of the W3C process, after which you will discover that the HTML5 technologies that are currently in the REC status are SVG, MathML, CORS, and the Geolocation APIs.

The second part of this chapter contains the HTML5 technologies with a W3C CR status, which includes the Battery API and the Vibration API. You will also learn about XHR2 (currently in WD status), which includes an example of the new FormData object.

The third part of this chapter contains a code sample that illustrates drag and drop for JPG files via jQuery (which is simpler than "pure" HTML5 Drag and Drop APIs), an example of invoking some of the HTML5 file APIs in jQuery, and finally a jQuery plugin for obtaining Geolocation information about users. You will also learn about the jQuery plugins jStorage and jStore that provide a layer of abstraction on top of the HTML5 Storage APIs. The final part of this chapter provides an overview of the History APIs, which is an HTML5 technology whose W3C status is WD.

If you have the choice (and perhaps the "luxury"), you probably prefer ease of coding instead of getting "bogged down" in long and tedious code samples when there are simpler alternatives available. For example, the code sample that uses jQuery for HTML5 Drag and Drop (DnD) is much more straightforward than a code sample that directly invokes the existing HTML5 DnD APIs.

The other advantage of simpler code samples is that they will quickly introduce you to a number of HTML5 technologies, after which you will be in a better position to explore the nuances of those HTML5 technologies. Another scenario is to combine jQuery Mobile with other toolkits, such as PhoneGap (discussed in Chapter 10) or appMobi (not discussed in this book). Toolkits such as PhoneGap provide support for hardware functionality; you can also combine PhoneGap with jQuery Mobile. Your application

mobile Web applications.

In general, jQuery and jQuery Mobile will probably work with other toolkits, but you ought to check online forums for any issues that might affect your Web applications. Finally, if you prefer not to use jQuery plugins, you can perform an Internet search to find online tutorials that illustrate to create HTML5 Web pages using pure HTML5 APIs for all the HTML5 technologies that are covered in this book.

The Stages in the W3C Review Process

If you are unfamiliar with the various stages of the W3C process, the following brief description (from earliest stage to final stage) is provided below.

A WD ("Working Draft") document is the first form of a standard that is publicly available. Comments are widely accepted but not guaranteed to be incorporated, and this document could differ significantly from its final form.

A LC ("Last Call") status involves the creation of a public record of the responses of a working group to all comments about a specification. This stage also handles bug reports about a specification. Incidentally, HTML5 reached LC status in the W3C in May 2011.

A CR ("Candidate Recommendation") status is firmer than the WD document, where major features have been finalized. The goal is to elicit assistance from the development community regarding the extent to which the standard can be implemented.

A PR ("Proposed Recommendation") status means that the standard has passed two previous stages, and at this point the document is submitted to the W3C for final approval.

A REC ("Recommendation") status is the final stage of a ratification process, comparable to a published technical standard in many other industries. The criterion for the specification becoming a W3C Recommendation is "two 100% complete and fully interoperable implementations."

Recall that Chapter 1 describes some of the groups that create specifications, with detailed information about APIs for the technologies that are discussed in this chapter.

HTML5 APIs in W3C Recommendation Status (REC)

You might be surprised to discover that SVG and MathML have a REC status. Both of these technologies are mature and are included under the HTML5 umbrella. However, neither topic is covered in this book.

HTML5 Geolocation

Geolocation allows users to share their current location, which may be determined by the following methods:

- Cell tower
- GPS hardware on the device
- IP address
- Wireless network connection

browser then determines the location and passes it back to the Geolocation API. Note that the W3C Geolocation specification mentions that there is no guarantee that the Geolocation API returns the device's actual location.

The geolocation object is a child object of window.navigator, and you can check if your browser supports geolocation with the following type of code block:

```
if(window.navigator.geolocation) {
  // geolocation supported
} else {
  // geolocation not supported
}
```

The W3C Geolocation API enables you to obtain geolocation information in a browser session that is running on a device. The geolocation object is available in the global window.navigator object, accessed via window.navigator.geolocation.

Note that the Geolocation API requires users to allow a Web application to access location information:

- The geolocation object contains the following three methods:
- getCurrentPosition(successCallback, errorCallback, options)
- watchPosition(successCallback, errorCallback, options)
- clearWatch(watchId)

The method getCurrentPosition() tries to get geolocation information. It uses the successCallback method if it is successful; otherwise, it calls the errorCallback method in its argument list.

The method watchPosition() obtains the geolocation at regular intervals, and it also returns a watchId value. Success and failure are handled through the two JavaScript methods in its list of arguments.

Finally, the method clearWatch(watchId) stops the watch process based on the value of watchId.

The major difference between the first two methods is that the watchPosition() method will return a value immediately upon being called, and the returned value uniquely identifies that watch operation.

A table that displays support for Geolocation on desktop and mobile browsers is here:

http://caniuse.com/geolocation

Obtain a User's Position with **getCurrentPosition()**

The PositionOptions object is an optional parameter that can be passed to the getCurrentPosition() method and watchPosition() methods. All of the properties in the PositionOptions object are optional as well.

For example, you can define an instance of a PositionOptions object by means of the

```
var options = {
  enableHighAccuracy: true,
  maximumAge: 60000,
  timeout: 45000
};
```

Next, we can invoke the getCurrentPosition() method by specifying a JavaScript success function, a JavaScript error function, and the previously defined options variable, as shown here:

```
navigator.geolocation.getCurrentPosition(successCallback,
                                         errorCallback,
                                         options);
```

Track a User's Position with **watchPosition()**

This method is useful when an application requires an updated position each time that a device changes location. The watch operation is an asynchronous operation that is invoked as shown here:

```
var watchID = null;
var options = { enableHighAccuracy: true, timeout: 30000 };

if (window.navigator.geolocation) {
  watchID = navigator.geolocation.watchPosition(
          successCallback,errorCallback, options);
} else {
  alert('Your browser does not support geolocation.');
}

function successCallback(position) {
  console.log("Success obtaining the device location");
}

// Error obtaining the location
function errorCallback(error) {
  console.log("Error obtaining the device location");
}
```

If your browser supports Geolocation, the JavaScript variable watcher is initialized via an invocation of the watchPosition() method of the geolocation object. Notice the JavaScript functions successCallback() and errorCallback() for handling success or failure, respectively. (In our case, these functions simply display a message in the browser's console).

passing a watchId to the clearWatch() method, as shown here:

```
navigator.geolocation.clearWatch(watcher);
```

After creating a new watch operation, you can remove that watch after successfully retrieving the position of a device, as shown here:

```
function successCallback(position) {
  navigator.geolocation.clearWatch(watcher);
  // Do something with a location here
}
```

As you can see, the JavaScript successCallback() function does nothing more than "clear" the JavaScript variable watcher. The key point is that you will continue receiving information until you clear this variable.

You can perform a Google search to find various JavaScript plugins that support Geolocation. If you prefer to use a jQuery plugin for Geolocation, there are several available, including this one:

http://mobile.tutsplus.com/tutorials/mobile-web-apps/html5-geolocation/

HTML5 Cross-Origin Resource Sharing (CORS)

In brief, the "same origin policy" allows scripts that originate from the same site and protocol to execute, and they can access each other's methods and properties without restriction. This policy also restricts AJAX calls to the same origin as the page (but with a "loophole" that circumvents this restriction when dynamically loading <script> tags).

As a simple example, the following pair of URLs is from the same site but with different protocols, so they do not meet the criteria for "same origin policy":

http://foo.com

https://foo.com

On the other hand, *cross-origin resource sharing* (CORS) specifies the ways in which a Web server can allow its resources to be accessed by Web pages from different domains. Although CORS is more flexible than "same origin policy," it does not allow access to resources by any and all requests.

In simplified terms, the CORS specification provides support for cross-domain communication by means of a simple header exchange between a client and a server.

Some of the new HTTP headers for the CORS specification are Options, Origin, and Access-Control-Allow-Origin. When the appropriate CORS headers are provided, CORS makes it possible to make asynchronous HTTP requests to other domains.

Keep the following point in mind: there is no single CORS API, but any API that uses the cross-origin features may be referred to as a CORS API. In practice, a CORS API uses the XMLHttpRequest object as a "container" for sending and receiving the requisite headers for CORS, and also the withCredentials property that can be used for determining programmatically whether or not an XMLHttpRequest object supports CORS.

many free articles with such information.

HTML5 APIs in W3C Candidate Recommendation Status (CR)

This section of the chapter contains a set of HTML5 APIs that have Candidate Recommendation status. Although the HTML5 technologies Navigation Timing, RDFa, and Selectors have CR status, they are not discussed in this book.

The Battery API

The Battery API is maintained by the DAP (Device APIs) working group, which provides information about the battery status of the hosting device. The following simple code snippet from Battery1.html shows you how to write the battery level to the console each time the level changes:

```
navigator.battery.onlevelchange = function () {
  console.log(navigator.battery.level);
};
```

The next section contains an AJAX-based code sample shows you how to use the new FormData object with XHR2, described below.

XMLHttpRequest Level 2 (XHR2)

The XMLHttpRequest Level 2 specification supports the following new features:

- Handling byte streams such as File, Blob, and FormData objects for upload and download
- Showing progress events during upload and download
- Making cross-origin requests
- Making anonymous requests (not HTTP Referrer)
- Setting a timeout for the request

Before we look at an XHR2 code sample, we'll start with a code sample that shows you how to make a simple AJAX request, followed by an AJAX request that uses jQuery.

Making AJAX Calls without jQuery

The code sample in this section shows you how to make a "traditional" AJAX call. Later, we will look at how to accomplish the same task using jQuery. The purpose of this example is to illustrate the fact that jQuery (once again) enables you to write simpler code that is easier to maintain, debug, and enhance with additional functionality (which you already know from the code samples you have seen throughout this book).

NOTE

You must launch BasicAjax1.html *from an actual Web server (not a file navigator) because this Web page reads the contents of a local file called* sample.xml.

LISTING 5.1 BasicAjax1.html

```
<!DOCTYPE html>
```

```
<head>
 <meta charset=utf-8 />
 <title>Basic Ajax</title>

 <script>
  var xmlHTTP, myFile = “http://localhost:8080/sample.xml”;
     var url = “http://localhost:8080/sample.xml”;

  function loadXML(url, callback) {
   if (window.XMLHttpRequest) {
     // Chrome, Firefox, IE7+, Opera, and Safari
     xmlHTTP = new XMLHttpRequest();
   } else {
     // IE5 and IE6
     xmlHTTP = new ActiveXObject(“Microsoft.XMLHTTP”);
   }

   xmlHTTP.onreadystatechange = callback;
   xmlHTTP.open(“GET”, url, true);
   xmlHTTP.send();
  }

  function init() {
   loadXML(myFile, function() {
    if(xmlHTTP.readyState==4 && xmlHTTP.status==200) {
     document.getElementById(“myDiv”).innerHTML =
                                xmlHTTP.responseText;
    }
   });
  }
 </script>
</head>

<body onload=“init()”>
  <div id=“myDiv”></div>
</body>
</html>
```

loaded into a browser. The init() function invokes the loadXML() function with the name of an XML document, along with a JavaScript function that is executed when the AJAX request is completed.

The loadXML() function contains conditional logic that determines how to initialize the JavaScript variable xmlhttp, followed by a code block that sets the name of the callback function, specifies a GET method and a URL in the url variable (not shown in this code sample), and then makes the actual AJAX request, as shown here:

```
xmlHTTP.onreadystatechange = callback;
xmlHTTP.open("GET", url, true);
xmlHTTP.send();
```

When the AJAX request is completed, the HTML element in Listing 5.1 is updated with the data that is returned by the AJAX request. In this code sample, the XML document sample.xml is an SVG document containing 3 rectangles and is reproduced here:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    width="100%" height="100%">
 <rect x="50"  y="10" width="100" height="200"
      stroke="blue" fill="red" />
 <rect x="200" y="10" width="100" height="200"
      stroke="blue" fill="green" />
 <rect x="350" y="10" width="100" height="200"
      stroke="blue" fill="blue" />
</svg>
```

If you are new to AJAX, then the code in Listing 5.1 might seem convoluted, and perhaps even confusing. Fortunately, jQuery simplifies the process of making AJAX requests by shielding you from the lower-level details, as you will see in the next section.

Making AJAX Calls with jQuery

This example is the modified version of Listing 5.1, which adds jQuery functionality to the code. There are several jQuery methods that provide AJAX-based functionality, including jQuery.load(), jQuery.get(),jQuery.post(), and jQuery.ajax().

NOTE *You must launch* JQueryAjax1.html *from an actual Web server (not a file navigator) because this Web page reads the contents of a local file called* sample.xml.

Listing 5.2 displays the contents of JQueryAjax1.html that illustrates how to use the first of these jQuery AJAX methods in an HTML Web page in order to produce the same result as Listing 5.1.

```
<!DOCTYPE html>
<html>
<head>
 <meta charset=utf-8 />
 <title>JQuery Ajax</title>

  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>
</head>

<body>
  <div id="myDiv"></div>

  <script>
    var url = "http://localhost:8080/sample.xml";

    $(document).ready(function() {
      $("#myDiv").load(url, function() {});
    });
  </script>
</body>
</html>
```

Listing 5.2 contains only one line of code that performs an AJAX request via the jQuery load() method, as shown here:

```
$("#myDiv").load(url, function() {});
```

The result executing the code in Listing 5.2 is the same as the result of executing the Listing 5.1. The contents of the HTML element whose id attribute is myDiv are replaced with the contents of sample.xml, and three SVG-based rectangles are rendered.

Alternatively, you can use the jQuery.ajax() method as shown here:

```
  $.ajax({
    url:  url,
    type: "get",
    success: GotData,
    dataType: 'xml'
  });
```

GotData() where you would process the result of the Ajax invocation.

As you would expect, jQuery provides much more AJAX functionality. Specifically, jQuery provides support for the following callback hooks:

beforeSend()

fail()

dataFilter()

done()

always()

The functions beforeSend(), error(), and done() are intuitively named and they behave as expected. The dataFilter() callback is the first callback to receive data; after it performs its processing, the done() callback is invoked. Finally, the always() callback is invoked, regardless of whether the result of the AJAX invocation was successful or in error.

A concise example of jQuery code that makes an AJAX request using method chaining with several of the preceding APIs is here:

```
// Assign handlers immediately after making the request,
// and get a reference to the jqxhr object for this request
var jqxhr = $.ajax( “example.php” )
   .done(function() { alert(“success”); })
   .fail(function() { alert(“error”); })
   .always(function() { alert(“complete”); });

// perform other work here …

// Set another completion function for the request above
jqxhr.always(function() { alert(“second complete”); });
```

If you want to explore more AJAX-related features in jQuery, a complete list of jQuery AJAX methods is here:

http://api.jquery.com/category/ajax/

http://api.jquery.com/ajaxComplete/

The next section contains a code sample that uses the new FormData object.

AJAX Requests using XMLHttpRequest Level 2 (XHR2)

Listing 5.3 displays AjaxForm.html, which illustrates how to create an HTML5 Web page using the new FormData object and XHR2.

LISTING 5.3 AjaxForm.html

```
<!doctype html>
<html lang=“en”>
```

```
 <meta charset=utf-8 />
 <title>Ajax Form</title>

 <script>
  function sendForm(form) {
    var formData = new FormData(form);

    var xhr = new XMLHttpRequest();
    xhr.open('POST', form.action, true);
    xhr.onload = function(e) {
      // do something here
    };

    xhr.send(formData);

    // Prevent page submission
    return false;
  }
 </script>
</head>

<body>
<form id="myform" name="myform" action="xhr2.php">
  <input type="text"   name="uname" value="asmith">
  <input type="number" name="id"    value="33333">
  <input type="submit" onclick="return sendForm(this.form);">
</form>
</body>
</html>
```

Listing 5.3 is straightforward. The <body> element contains a HTML <form> element with several input fields. Next, the form data is submitted via the JavaScript function sendForm(), which creates a FormData object and then submits the user-provided data via XHR2, as shown in this code block:

```
var xhr = new XMLHttpRequest();
xhr.open('POST', form.action, true);
xhr.onload = function(e) {
   // do something here
```

```
xhr.send(formData);
```

NOTE

The CD does not provide a PHP file xhr2.php*, so the HTML Web page* AjaxForm1.html *does not submit form data to a server.*

A final point to know: the XMLHttpRequest Level 2 specification supports the transfer of binary data and also tracks the upload progress through the XMLHttpRequestUpload object. Consequently, XHR2 can be used for binary file transfers via the File APIs and the FormData object.

If you need to use XHR2 in your HTML Web pages, an XHR2 library is here:

https://github.com/p-m-p/xhr2-lib

More tutorials and information regarding XHR2 are here:

http://www.html5rocks.com/en/tutorials/file/xhr2/

http://www.matiasmancini.com.ar/jquery-plugin-ajax-form-validation-html5.html

HTML5 Drag and Drop (DnD)

HTML5 Drag and Drop (DnD) enables you to rearrange the layout of HTML elements in an HTML Web page. HTML4 does not have built-in support for DnD, and creating such support requires considerably more JavaScript code than a toolkit such as jQuery.

On the other hand, HTML5 provides APIs that support Drag and Drop in HTML5 Web pages. HTML5 Drag and Drop emits the following events:

```
drag
dragend
dragenter
dragleave
dragover
dragstart
drop
```

In addition, the HTML5 DnD provides a source element, the data content, and the target. Respectively, they represent the drag “start” element, the data being dragged, and the “target” element.

In your HTML5 Web page, you attach event listeners to elements (with an optional third parameter), as shown here:

```
myElement.addEventListener(‘dragenter’, handleDragEnter, false);
myElement.addEventListener(‘dragleave’, handleDragLeave, false);
```

```
myElement.addEventListener('dragstart', handleDragStart, false);
```

Next, you define custom code in each of the JavaScript event handlers that will be executed whenever the associated event occurs.

However, keep in mind that HTML5 Web pages with DnD functionality still require browser-specific code, which means that you need to maintain the code in multiple HTML5 Web pages if you want to support multiple browsers.

Eric Bidelman has written an extensive and detailed blog entry that shows you how to write an HTML5 Web page with Drag and Drop functionality:

http://www.html5rocks.com/en/tutorials/dnd/basics/

We will skip examples of "native" HTML5 DnD and proceed to an example of using jQuery with HTML5 DnD, which is covered in the next section.

jQuery and HTML5 Drag and Drop

Drag and Drop is exceptionally simple in jQuery: only one line of code is required for an HTML element.

Listing 5.4 displays the contents of the HTML5 Web page JQDragAndDrop1.html, which illustrates how easy it is to create an HTML5 Web page with Drag and Drop using jQuery.

LISTING 5.4 JQDragAndDrop1.html

```
<!doctype html>
<html lang="en">
<head>
 <meta charset=utf-8 />
 <title>JQuery DnD</title>

<style>
 div[id^="draggable"] {
   position:relative; width: 100px; height: 100px;
 }

 #draggable1 { background: red; }
 #draggable2 { background: yellow; }
 #draggable3 { background: blue; }
</style>

  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>
```

```
<script src="http://ajax.googleapis.com/ajax/libs/jqueryui
                        /1.8.9/jquery-ui.min.js">
</script>
—>
<script src="http://ajax.googleapis.com/ajax/libs/jqueryui
                        /1.10.1/jquery-ui.min.js">
</script>

 <script
  src="http://code.jquery.com/jquery-ui-1.8.14.custom.min.js">
 </script>
</head>

<body>
 <div id="content" style="height: 400px;">
  <div id="draggable1">
      <img src="sample1.png" width="100" height="100" />
  </div>
  <div id="draggable2">
      <img src="sample2.png" width="100" height="100" />
  </div>
  <div id="draggable3">
      <img src="sample3.png" width="100" height="100" />
  </div>
 </div>

 <script
  $(document).ready(function() {
    $('#draggable1').draggable();
    $('#draggable2').draggable();
    $('#draggable3').draggable();
  });
 </script>
</body>
</html>
```

Listing 5.4 contains a block of jQuery code that makes the three HTML elements (defined in the <body> element) draggable, using this type of code snippet:

The <body> element contains three elements, each of which contains a PNG image that you can drag around the screen.

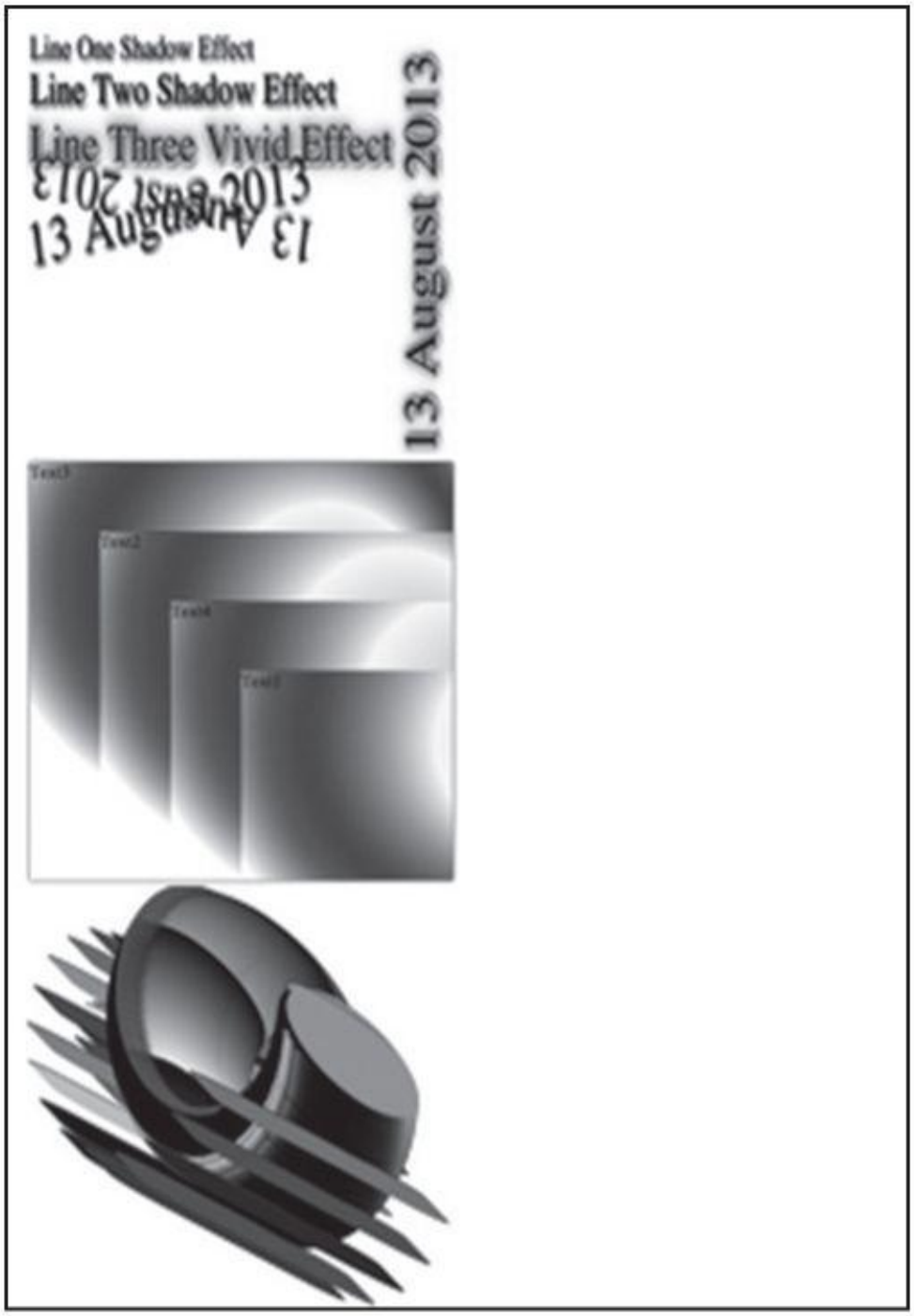


Figure 5.1 Three images in the Chrome browser on a MacBook.

Figure 5.1 displays the result of rendering Listing 5.4 in the Chrome browser on a MacBook.

Figure 5.2 shows an example of dragging the images in Listing 5.4 to different positions in the Chrome browser on a MacBook.

Figure 5.2 Three dragged images in the Chrome browser on a MacBook.

More information about jQuery Drag and Drop (including the list of available options) is available here:

http://jqueryui.com/demos/draggable/

There are several jQuery plug-ins for drag-and-drop functionality that are listed here:

http://plugins.jquery.com/projects/plugins?type=45

You can also use jQuery Mobile with HTML5 DnD, and although we will not discuss an example (due to space constraints), you can perform an Internet search to find tutorials and code examples, or you can start with the details in this link:

http://www.jsplugins.com/Scripts/Plugins/View/Jquery-Mobile-Drag-And-Drop/

jQuery and HTML5 Local Storage

HTML5 provides both local storage and session storage that enable Web pages to store data locally. Although they are not discussed in this book, you can perform an Internet search to learn about their functionality. This section introduces you to jQuery plugins that provide a layer of abstraction over local storage and session storage.

The jQuery plugin jStorage is a cross-browser plugin that enables you to use jQuery syntax in order to manage data in local storage, and its homepage is here:

http://www.jstorage.info/

The JStorage APIs enable you to get, set, and delete data in storage. You can also check

named JStorage APIs are listed here for your convenience:

```
$.jStorage.set(key, value)
$.jStorage.get(key)
$.jStorage.deleteKey(key)
$.jStorage.flush()
$.jStorage.index()
$.jStorage.storageSize()
$.jStorage.currentBackend()
$.jStorage.reInit()
$.jStorage.storageAvailable()
```

If local storage or session storage is unavailable (such as in IE6 or IE7), jStorage adds polyfills to support this functionality.

Listing 5.5 displays the contents of JQJStorage1.html and illustrates how to use JStorage in order to save data in the storage area of your browser.

LISTING 5.5 JQJStorage1.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
   <meta charset=utf-8 />
   <title>JStorage Example</title>

  <link rel="stylesheet"
   href="http://code.jquery.com/mobile/1.1.0/jquery.mobile-1.1.0.min.css" />

  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>

  <script src="http://code.jquery.com/ui/1.10.1/jquery-ui.js">
  </script>

 <script
  src="http://code.jquery.com/mobile/1.1.0/
                        jquery.mobile-1.1.0.min.js">
 </script>
```

```
<script>
  $(document).ready(function() {
    // Check if “key” exists in the storage
    var value = $.jStorage.get(“key”);

    if(!value){
      // load the data from the server
      value = load_data_from_server()

      // save the value
      $.jStorage.set(“key”,value);
    }
  });
</script>
</head>

<body>
  <div> </div>
</body>
</html>
```

The core functionality in Listing 5.5 takes place in a short JavaScript code block, with conditional logic that checks for the existence of a data item in local storage with the value key. If this value does not exist, then the load_data_from_server() JavaScript function (which is not implemented in this code sample) is invoked, and then the return value is stored in local storage.

You can also use the jQuery plugin jStore for HTML5 storage, which is a cross-browser plugin that enables you to use jQuery syntax in order to manage data in local storage. Its homepage is here:

http://code.google.com/p/jquery-jstore/source/browse/trunk/src/engines/jStore.Html5.js?r=6

Libraries for HTML5 Local Storage

A number of JavaScript toolkits are available for local storage, some of which use jQuery (and some do not use jQuery). This short section briefly describes some of the available toolkits.

lscache emulates memcache functions using HTML5 localStorage for caching data in a client browser, along with an expiration time for each data item.

The lscache homepage is here:

https://github.com/pamelafox/lscache

If the localStorage limit (approximately 5MB) is exceeded, items that are closest to their expiration date are removed. If localStorage is unavailable, lscache does not cache anything (and all cache requests return null).

The lscache methods are set(), get(), remove(), and flush(), and a jQuery lscache plugin is available here:

https://github.com/mckamey

YQL LocalCache is a wrapper for YQL to support local storage, and its homepage is here:

https://github.com/phunkei/autoStorage

A sample invocation with YQL LocalCache with self-explanatory parameters is here:

```
yqlcache.get({
  yql: 'select * from flickr.photos.search where text="warsaw"',
  id: 'myphotos',
  cacheage: ( 60*60*1000 ),
  callback: function(data) {
    console.log(data);
  }
});
```

The returned data in the callback is an object with two properties: data (the YQL data) and type ('cached' for cached data or 'freshcache'). You can get additional code samples here:

https://github.com/codepo8/yql-localcache

Savify is a jQuery plugin for automatically recording a user's progress in a form while it is being completed. Its homepage is here:

https://github.com/blackcoat/Savify

jQuery and HTML5 File APIs

The HTML5 File APIs enable you to create, read, and write files on the file system. The first step is to obtain access to the HTML5 FileSystem, after which you can perform file-related operations. You can read about these APIs and see some code examples here (and make sure you check whether or not your target browser is supported):

http://blueimp.github.com/jQuery-File-Upload/

http://www.htmlgoodies.com/html5/other/responding-to-html5-filereader-events.html

The CD contains the HTML Web page JQFileInfo1.html, which illustrates how to use jQuery in order to display the attributes of a file that is selected by users. A more interesting (and useful) example is JQFileUpload2.html in Listing 5.6, which illustrates how to

LISTING 5.6 JQFileUpload2.html

```
<!DOCTYPE HTML>
<html lang="en">
 <head>
  <meta charset=utf-8 />
  <title>File Upload with XHR2</title>
 <script src="http://code.jquery.com/jquery-2.0.0b1.js">
 </script>
 <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
 </script>
</head>

 <body>
 <script>
  $(document).ready(function () {
    var url = "http://localhost:8080/";

    $("body").on("change", '#fileUploader', function() {
      // Prepare post data
      var data = new FormData();
      data.append('uploadfile', this.files[0]);

      // invoke the jQuery .ajax method:
      $.ajax({
        url: url,
        type: 'POST',
        contentType: false,
        processData: false,
        data: data,
        success:  function(data) {
                  // do something here
                },
        dataType: 'text'
      });
    });
  });
 </script>
```

```
  <div>
    <input id="fileUploader" type="file" multiple />
  </div>
 </body>
</html>
```

Listing 5.6 contains an HTML <input> field that enables users to select a file from the file system. After a file is selected, the jQuery "change" event is triggered, and an XHR2 FormData object (discussed earlier in this chapter) is created and populated, as shown here:

```
var data = new FormData();
data.append('uploadfile', this.files[0]);
```

The selected file is uploaded via the jQuery .ajax() method, which contains a "success" function that is invoked after the AJAX request has been completed successfully.

Note that you must set the value of contentType and processData to false, or you will get the following error in the Web Inspector:

```
Uncaught TypeError: Illegal invocation
```

The following link provides an explanation regarding these two parameters:

http://stackoverflow.com/questions/12431760/html5-formdata-file-upload-with-rubyonrails

If you want to use a jQuery plugin for uploading files in HTML5 Web pages, there are several available, such as the cross-browser jQuery plugin jquery-filedrop. Its homepage is here:

https://github.com/weixiyen/jquery-filedrop

HTML5 History APIs

Prior to HTML5, the browser support for history-related APIs provided limited functionality. You could find the number of items in the browser history, move forward and backward, and several links backward, as shown here:

```
console.log(history.length);
console.log(history.forward());
console.log(history.back());
console.log(history.go(-2));
```

Several new APIs are available, as shown here:

```
history.pushState(data, title, url);
history.replaceState(data, title, url);
```

The parameters in the history.pushState() method are as follows:

- data is some type of structured data, assigned to the history item.
- title is the name of the item in the history drop-down that is displayed by the browser's

- url is the (optional) URL that is displayed in the address bar.

The parameters in the history.replaceState() method are the same as the history.pushState() method, except that the former updates information that is already in the browser's history.

In addition, there is the new popstate event, which occurs when uses view their browser history, and you can use it in the following manner:

```
window.addEventListener("popstate", function(event) {
  // do something here
});
```

Keep in mind that different browsers implement HTML5 browser history in different ways, so the following JavaScript toolkit is useful:

https://github.com/balupton/History.js/

The History.js toolkit supports jQuery, MooTools and Prototype. In HTML5 browsers, you can modify the URL directly without resorting to hashes (such as example.com/#/foo/…). In addition, changing the fragment does not trigger a page reload.

HTML5 Offline Web Applications

The purpose of offline Web applications is simple: users can work on an application even when they are disconnected from the Internet. When users do have access to the Internet, their data changes are synchronized so that everything is in a consistent state.

A Website that contains demos, additional links, and tutorial-like information is here:

http://appcachefacts.info/

The HTML5 specification requires a so-called "manifest file" (with "appcache" as the suggested suffix) that contains the following three sections:

CACHE (the list of files that are going to be cached)

NETWORK (the files outside the cache that can only be accessed online)

FALLBACK (specifies the resource to display when users try to access non-cached resources)

As a simple example, Listing 5.7 displays the contents of a sample manifest file called MyApp.appcache.

LISTING 5.7 MyApp.appcache

```
CACHE MANIFEST
# Verson 1.0.0
CACHE:
Index.html
Cachedstuff.html
Mystyle.css
Myimage.jpg
```

```
NETWORK:
*
FALLBACK:
/ noncached.html
```

You must ensure that the manifest file is served with the following MIME type:

```
text/cache-manifest
```

Second, Web applications that uses offline functionality must reference the manifest file at the top of one of its Web pages:

```
<html lang="en" manifest="mymanifest.manifest">
```

If you have a Web page that is hosted by a provider, you can verify that the Web page contains the correct MIME type by issuing the following type of command:

```
curl –I http://www.myprovider.com/mymanifest.manifest
```

Detecting Online and Offline Status

The simplest way to determine whether or not an application is offline in an HTML5 Web page is with the following code snippet:

```
if(navigator.online) {
  // application is online
} else {
  // application is offline
}
```

For mobile applications that use jQuery Mobile, you can use the following type of code block:

```
$(document).bind("offline", function() {
  // application is offline
}
```

Binding the offline event as shown in the preceding code block is useful for handling situations whereby an application goes offline while users are actively viewing an application.

In addition, you would send data to a server only when you are online, and store data locally via HTML5 LocalStorage when you are offline.

The jQuery plugin jquery-offline is a cross-browser plugin that enables you to use jQuery syntax for offline applications. Its homepage is here:

https://github.com/wycats/jquery-offline

Summary

This chapter provided an overview of various HTML5 technologies and grouped them according to their current W3C status. For your convenience, these HTML5 technologies

- CORS
- AJAX (XHR2)
- Drag and Drop
- File APIs
- Forms
- Geolocation

INTRODUCTION TO SINGLE-PAGE APPLICATIONS

This chapter provides an introduction to Single-Page Applications, commonly referred to as SPAs. The first part of this chapter briefly discusses the rationale for creating a single-page application in JavaScript. The XHR2 section contains AJAX-related examples, starting with a generic AJAX code sample, followed by a jQuery-based AJAX code sample. The second part of this Chapter provides an overview of BackboneJS and Twitter Bootstrap. The third part of this chapter contains an abbreviated introduction to Jade, MongoDB, Mongoose, and NodeJS. The final portion of this chapter provides code for a minimalistic SPA that you can enhance with your own custom code.

What is an SPA?

Let's start with a whirlwind "review" of Web application architecture. First there were servers that delivered static pages to Websites, which evolved into servers that could deliver dynamically generated Web pages (using languages such as PHP) to Websites. Next came AJAX, which provided partial-page refresh functionality that was more efficient and also created a better user experience. Then, a collection of Comet-based solutions arose in order to make the communication between browsers and servers more efficient. More recently, HTML5 WebSockets provide bidirectional full duplex asynchronous communication between browsers and servers via a persistent TCP connection using the WebSocket protocol. This is an improvement over the HTTP protocol that is the basis for all earlier Web applications.

During this time, various strategies were devised for the division of code between the browser and the server. One approach involves a "thick server" where most of the operations (such as generating dynamic Web pages and user authentication/validation) are performed, along with a "thin client" (or browser) that renders a Web page from a server but with limited additional functionality.

Modern Web Architecture

Recently there has been a proliferation of JavaScript toolkits that enable people to develop Web applications with a "thick client" and "thin server." Although there is a diversity of architectures (even among recent Websites), many modern Web applications have the following architecture:

- DOM (contains read-only data)
- Models (contain the state and data of the application)
- Modules (small independent subsystems)
- Views (they observe model changes via notifications)

have a model layer that handles data and view layers that read data from models, so they can redraw the UI without involving a server round-trip, which can give SPAs a more "native" feel to them. Fewer round-trips reduce the load on the server, which is obviously another benefit.

However, keep in mind the following points. First, there is no "specification" for SPA, and that SPA is evolving and improving over time as a result of the contributions from people in the community. Second, some toolkits that have an SPA also use AJAX, which arguably breaks the "ideal" design of SPA. Thus, different toolkits (and the applications that are based on those toolkits) have different implementations of SPA, which in turn increases the likelihood that you will find a toolkit that is well suited to your specific needs.

MVC and MV* Patterns

The MVC (Model-View-Controller) pattern is well documented (plenty of online information is available) and has been popular for years; however, the concept of controllers can be absent from SPAs and from some JavaScript toolkits. "Traditional" Web applications use a page refresh in order to navigate between independent views, whereas JavaScript Web applications based on SPA can retrieve (via AJAX) data from a server and dynamically rendered data in different views in the same Web page. Routers are used for navigation purposes because URLs are not updated while navigating to different views in a SPA Web application.

Although controllers usually update the view when the model changes (and vice versa), most JavaScript MVC frameworks tend to differ from the MVC pattern in terms of their use of controllers. As designs evolve in a community, variants emerge based on individual preferences and experiences. Since no individual or organization promotes the idea of MVC, these variants are free to co-exist.

Contrary to what you might read in online articles, BackboneJS does not have an MVC pattern: controllers are absent in current versions. (Also, note that the Controller object in early versions was renamed to the Router object in later versions.) Some controller-like logic is located in BackboneJS views and also in BackboneJS routers (the latter are employed to manage application states). BackboneJS views receive notifications when there are model changes, and the views are updated accordingly. BackboneJS can be best classified as having an MV* architecture.

In addition, there are other models, such as the MVP (Model-View-Presenter) pattern and MVVM (Model-View-ViewModel, which is used in .NET frameworks and in KnockoutJS). MVP appears to be better suited to Web applications in which there are numerous large views, and the role of the Presenter overlaps with that of a controller.

You can learn more about MVC and MVP in this article:

http://martinfowler.com/eaaDev/uiArchs.html

Generating Web Pages in SPAs

As you already know, many Web pages are generated dynamically on a server that sends those Web pages to browsers. In modern Web applications, one solution is to

the more traditional approach. The distinction between server-rendered and client-rendered Web pages helps to reinforce the fact that the *server* renders a given Web page, but the *client* renders subsequent updates.

Handling Model-Related Events in SPAs

The two major options for handling model data changes are observables and event emitters, and there is little difference between these two approaches. When a change occurs, the code that is "bound" to that change event is triggered. For example, events are registered on objects:

```
MyApp.on('change', function() { … });
```

whereas observers are attached through global names:

```
Framework.registerObserver(window.MyApp,'change',function(){…});
```

Observables usually have some type of name resolution system, where you use strings in order to refer to objects. A global name resolution system (where names are strings rather than directly accessing objects) is often added for observables in order to facilitate the use of observers, which can only be registered when the objects they refer to have been instantiated.

Client-Side Technologies for SPAs

In brief, client-side technologies for an SPA include toolkits such as jQuery, BackboneJS, and Jade. However, you can certainly use other toolkits, such as variants of BackBoneJS (Spine, Vertebrae, and so forth), or an entirely different toolkit (such as EmberJS). In addition to Jade, there are other templating engines available, including Mustache and Handlebars. Perform an Internet search for information as well as a feature comparison of these toolkits.

BackboneJS

Backbone.js is an open source JavaScript toolkit for developing structured Web applications, and its homepage is here:

http://Backbone.js.org/

Although Backbone.js is not based on an MVC (Model-View-Controller) pattern, there are some similarities. Backbone.js provides *models* with key-value binding and custom events, *collections* with an API of enumerable functions, and *views* with declarative event handling, all of which is connected to your existing API over a RESTful JSON interface. Backbone.js is an open-source component of DocumentCloud, and it's available on GitHub under the MIT software license, where you can find the source code, an online test suite, an example application, a list of tutorials, and real-world projects that use Backbone.js.

Download Backbone.js from its GitHub repository:

https://github.com/documentcloud/backbone/

A Brief Introduction to BackboneJS

The following subsections give you a simple overview of models, views, collections, and routers in BackboneJS. After you have read this section, you can search for online

What is a Model?

According to the authors of Backbone:

"Models are the heart of any JavaScript application, containing the interactive data as well as a large part of the logic surrounding it: conversions, validations, computed properties, and access control."

Whenever you create a new instance of a model, the initialize() method is invoked (which is also the case when you instantiate a new view in BackboneJS). A simple example is shown in the following code block:

```
Vehicle = Backbone.Model.extend({
   initialize: function(){
      alert("I am a vehicle");
   }
});
var vehicle = new Vehicle;
```

You can set parameters in a model in one of two ways. One method is to use a constructor, as shown here:

```
var vehicle = new Vehicle({model: "Ford", make: "Mustang"});
```

Another way is to use the set method, as shown here:

```
vehicle.set({make: "Ford", model: "Mustang"});
```

After setting property/value pairs, you can retrieve their values as follows:

```
var make  = vehicle.get("make");  // "Mustang"
var model = vehicle.get("model"); // "Ford"
```

You can also set default values in the following manner:

```
Vehicle = Backbone.Model.extend({
   defaults: {
     make:   "Ford",
     model:  "Mustang"
   },
   initialize: function(){
      alert("I am a vehicle");
   }
});
```

Model Changes

You can listen for changes to the model and execute code whenever a change is detected.

For example, here the new contents of the initialize method:

```
    alert("I am a vehicle");


    this.bind("change:model", function(){
        var model = this.get("model");
        alert("Changed my model to " + model);
    });
}
```

There are various other features available for manipulating models that you can read in the online documentation.

What is a View?

In simplified terms, Backbone views provide a visual representation of the data models in an application. Moreover, Backbone views listen to events and respond to changes in data models in order to ensure that data in a given view is synchronized with its associated model.

```
SearchView = Backbone.View.extend({
    initialize: function(){
        alert("A view of a vehicle.");
    }
});


// The initialize function is like a constructor in that
// it is always called when you instantiate a Backbone View
var search_view = new SearchView();
```

What is a Collection?

A Collection in BackboneJS is an ordered set of models. Collections are useful because you can include methods inside of them to fetch data from a server, prepare that data before returning the collection, and set sample collections for debugging/testing.

Moreover, you can add event listeners and attach views to collections, which is not the case for simple JavaScript arrays.

As a simple example, we can use the Model Vehicle that we created in the previous section in order to create a Collection in Backbone as follows:

```
var Cars = Backbone.Collection.extend({
  model: Vehicle
});


var vehicle1 = new Vehicle({ make: "Ford",  model: "Mustang"  });
var vehicle2 = new Vehicle({ make: "GM",    model: "Camaro"   });
```

```
var myCars = new Cars([ vehicle1, vehicle2, vehicle3]);
console.log( myCars.models ); // [vehicle1, vehicle2, vehicle3]
```

What is a Router?

BackboneJS routers are used for mapping URL fragments in your application (e.g., mapping example.com/*/foo to shareFoo). Routing the URLs in your applications via URL fragments makes them useful for applications that need URL routing and history capabilities. Defined routers should contain at least one route and a function to map to that particular route. Keep in mind that routes interpret anything after "#" tag in the URL, and all links in your application should target either "#/action" or "#action. "

As a simple example, the following code block defines a Backbone router:

```
var MyAppRouter = Backbone.Router.extend({
   routes: {
       // matches http://example.com/#you-can-put-anything-here
      "*actions": "defaultRoute"
   },
   defaultRoute: function( actions ){
      // The variable passed in matches the variable
      // that is in the route definition "actions"
      alert( actions );
   }
});

// Initiate the router
var appRouter = new MyAppRouter();

// Start Backbone history (required for bookmarkable URLs)
Backbone.history.start();
```

This concludes a bare-bones introduction to BackboneJS. The next section provides some useful links with more detailed information.

Useful Links

A collection of BackboneJS tutorials is here:

http://backbonetutorials.com/

An example of using BackboneJS with jQuery Mobile:

http://coenraets.org/blog/2012/03/using-backbone-js-with-jquery-mobile/

http://weblog.bocoup.com/organizing-your-backbone-js-application-with-modules/

Backbone Boilerplate

Backbone-boilerplate is an open source JavaScript toolkit that provides "boilerplate" functionality analogous to toolkits such as HTML5 Boilerplate. Its homepate is here:

https://github.com/tbranyen/backbone-boilerplate

The set-up instructions for Backbone-boilerplate are available in the readme.md file that is included in this distribution:

https://github.com/backbone-boilerplate/grunt-bbb

The set-up is somewhat lengthy (and beyond the constraints of this chapter), but it's worth exploring this toolkit if you plan to use BackboneJS extensively in your code.

Variations of BackboneJS

There are many variations of BackboneJS (which are also extensions of BackboneJS), some of which are listed here:

pine.js

joint.js

ligament.js

vertebrae.js

bones.js (provides server-side functionality)

hambone.js

shinbone.js

Space constraints in this chapter preclude a thorough discussion of these toolkits, but you can perform an Internet search to find information about these toolkits.

One other variant of BackboneJS is SpineJS, which uses controllers (even though it is based on BackboneJS). Controllers in SpineJS provide the "glue" in Web applications, and they provide the functionality of traditional controllers.

The following SpineJS code block defines a Spine controller with three functions. The controller synchronizes changes in models to update views (and vice versa). Compare this code with BackboneJS code blocks that you saw earlier in this chapter:

```
// Controllers inherit from Spine.Controller:
var RecipesController = Spine.Controller.sub({
  init: function(){
    this.item.bind('update',  this.proxy(this.render));
    this.item.bind('destroy', this.proxy(this.remove));
  },

  render: function(){
    // Handle templating
    this.replace($('#recipeTemplate').tmpl(this.item));
```

```
  },

  remove: function(){
    this.$el.remove();
    this.release();
  }
});
```

EmberJS

In addition to BackboneJS, there are various toolkits available, and one of the most popular is EmberJS, whose homepage is here:

http://emberjs.com/

EmberJS applications download everything that is required to run during the initial page load, which is consistent with SPA. One point to keep in mind is that EmberJS adopts an MVC pattern as well as a heavy emphasis on routers, and also a templating engine. EmberJS focuses on the "heavy lifting" to handle some of the more burdensome JavaScript tasks. On the other hand, BackboneJS has a much more minimalistic approach (but it can be enhanced with Backbone Boilerplate).

As a simple illustration, the following code block shows you how to define JavaScript objects in EmberJS.

```
Person = Ember.Object.extend({
  greeting: function(msg) {
    alert(msg);
  }
});

var person = Person.create();
// alert message: My Name is Dave
person.greeting("My name is Dave");

var person = Person.create({
  name: "Jeanine Smith",
  greeting: function() {
    this.greeting("My name is " + this.get('name'));
  }
});

// alert message: My Name is Jeanine
```

You can extend objects in EmberJS, as shown here:

```
var HappyPerson = Person.extend({
  greeting: function(msg) {
    this._super(greeting.toUpperCase());
  }
});
```

You can also add event listeners in EmberJS , as shown here:

```
person.addObserver('name', function() {
  // do something after a name change
});
person.set('name", 'Oswald');
```

An extensive comparison between BackboneJSand EmberJS is here:

http://www.i-programmer.info/programming/htmlcss/4966-creating-web-apps-the-camera-api.html

If BackboneJSand EmberJS are not a good fit for your needs, you can perform an Internet search for other toolkits. You can also find various online articles that discuss the merits of various toolkits, and sometimes you can find articles that provide a direct comparison of various toolkits. One point to keep in mind is that there is an abundance of toolkits available. If time is a constraint, you might need to limit your analysis to a small number of toolkits so that you can properly assess them.

Twitter Bootstrap

Twitter Bootstrap is an open source project for creating Websites and Web applications. This toolkit contains HTML- and CSS-based design templates for UI Controls (such as forms, buttons, charts, navigation, and so forth), as well as user interactions.

Currently, Twitter Bootstrapis the most watched and forked repository in GitHub, and you can download it here:

http://twitter.github.com/Bootstrap/

http://blog.getBootstrap.com/

Twitter Bootstrap was designed primarily as a style guide to document best practices, and also to assist people of diverse skill levels. Bootstrap supports new HTML5 elements and syntax, and you can use Bootstrap as a complete kit or to start something more complex.

Twitter Bootstrap was initially created for modern browsers, but it has expanded its support to include all major browsers (including IE7). In addition, Twitter Bootstrap2.0 supports tablets and smartphones, and its responsive design means that its components are scaled according to a range of resolutions and devices, thereby providing a consistent experience. Moreover, Twitter Bootstrapprovides custom-built jQuery plugins, and it's built on top of the LESS toolkit. Some features ofTwitter Bootstrap 2.0 are tooltips, styled stateful buttons, more table and form styling, and an improved structure for its CSS source code (multiple files

form controls, navigation lists, labels, and others. If you prefer, you can create your own custom download if you do not want to download the full toolkit.

Bootstrap requires HTML5 doctype, which is specified with this snippet:

```
<!DOCTYPE html>
```

Twitter Bootstrap sets basic global display, typography, and link styles (located in scaffolding.less of the distribution), including the following examples:

- Remove margin on the body
- Set background-color: white; on the body
- Use the @baseFontFamily, @baseFontSize, and @baseLineHeight attributes as the typographic base
- Set the global link color via @linkColor and apply link underlines only on :hover

Keep in mind that Twitter Bootstrap 2.0 uses much of Normalize.css (which also powers HTML5 Boilerplate) instead of the 'reset' block in version 1 of Bootstrap.

Additional details are available here:

http://twitter.github.com/bootstrap/scaffolding.html

One interesting Bootstrap feature is its support for progress bars using the CSS classes .bar and .progress that are available in Bootstrap. As an illustration, the following code block shows how to render bars with different colors based on custom attributes that start with the string progress, as shown here:

```
<div class="progress progress-info" style="margin-bottom: 9px;">
  <div class="bar" style="width: 20%"></div>
</div>
<div class="progress progress-success" style="margin-bottom:
                                    9px;">
  <div class="bar" style="width: 40%"></div>
</div>
<div class="progress progress-warning" style="margin-bottom:
                                    9px;">
  <div class="bar" style="width: 60%"></div>
</div>
<div class="progress progress-danger" style="margin-bottom:
                                    9px;">
  <div class="bar" style="width: 80%"></div>
</div>
```

Useful Links

The following links provide a sample application using Twitter Bootstrap, as well as a set of videos and tutorials:

http://webdesign.tutsplus.com/tutorials/workflow-tutorials/twitter-Bootstrap-101-tabs-and-pills/

http://twitter.github.com/Bootstrap/examples.html

Here is an example of BackboneJS with Twitter Bootstrap:

http://coenraets.org/blog/2012/02/sample-app-with-backbone-js-and-twitter-Bootstrap/

The following open source project uses Bootstrap with SaSS instead of LESS:

https://github.com/jlong/sass-twitter-Bootstrap

A Minimalistic SPA

The remainder of this chapter shows you how to set up several server-side toolkits for creating a very simple SPA. You will see "bare bones" code that provides a starting point for an SPA that enables users to see the books that their friends have read, along with other information, such as reviews, interesting book quotes, and so forth. This application uses the following technologies:

Jade

MongoDB

Mongoose

NodeJS

The next section of this chapter provides a high-level description of the technologies in the preceding list that have not been discussed already. The final section of this chapter contains code blocks that will help you understand the logic of the SPA application. As a suggestion, include additional toolkits (such as BackboneJS and jQuery) in this SPA so that you can expand your skillset.

Jade

Jade is a popular templating language for NodeJS, and its homepage is here:

http://jade-lang.com/

Download Jade here:

https://github.com/visionmedia/jade#readme

After you uncompress the Jade distribution, you can create a single JadeJavaScript file with the following command:

```
make jade.js
```

Note that you can also install Jade via npm (the package manager for NodeJS) as follows:

```
npm install jade
```

Jade Code Samples

Jade uses HTML tags to generate HTML elements, the "#" syntax to generate id attributes, and the "." syntax to generate class attributes. Indentation specifies nesting of

```
html
  head
  body
    div
    div#myid
    div.myclass
    p my paragraph
```

generates the following HTML:

```
<html>
 <head></head>
 <body>
   <div></div>
   <div id="myid"></div>
   <div class="myclass"></div>
   <p>my paragraph</p>
 </body>
</html>
```

Jade supports interpolation, so you can pass data to a Jade document. For example, suppose you have this data:

```
{fname: john, lname: smith}
```

and that you have this Jade element:

```
div #user #{fname} #{lname}
```

then the result is this HTML element:

```
<div id="user">john smith</div>
```

Jade also supports comments (single and multi-line), conditional logic, a "case" statement, filters, and iteration.

As a more concrete example, Listing 6.1 displays the contents of layout.jade , which generates a very basic HTML5 Web page.

LISTING 6.1 layout.jade

```
doctype 5
html
  head
    title= title
    link(rel='stylesheet', href='/css/style.css')
    script(src='/js/jquery-1.8.2.min.js')
    script(src='/js/client.js')
```

```
body
  block content
```

As you can see in Listing 6.1, Jade uses an easy syntax for defining links to CSS stylesheets and for defining HTML <script> elements.

In addition, Jade allows you to extend the contents of a template and also to use conditional logic. Listing 6.2 extends the contents with layout.jade based on whether or not user.id (which is defined elsewhere) is defined.

LISTING 6.2 index.jade

```
extends layout


block content
  h1= title
  - if (user.id)
    p Hello #{user.name}, and welcome to #{title}
    p
      a(href='/managebooks') My Book List
    p
      a(href='/logout') Logout
    #bookhistory
  - else
    div
      a(href='/login') Login with password
    div
      a(href='/register') Register
```

As you can see, the first portion of Listing 6.2 (starting with "if") displays a welcome message and a link where the currently logged-in user can see a list of books. The second portion of Listing 6.2 (starting with "else") displays a element with a link for the login page, as well as a second element for a registration page.

A Minimal NodeJS Code Sample with Jade

This section shows you how to create a tiny NodeJS application that uses Jade with the following three Jade files (which are placed in a views subdirectory) and one JavaScript file:

views/index.jade

views/layout.jade

views/navigation.jade

server.js

Here are the contents of index.jade :

```
h1
```

Here are the contents of layout.jade:

```
!!! 5
 html(lang='en')
  head
 title= title
 body!= body
  div#navigation!= partial('navigation.jade')
```

Here are the contents of navigation.jade:

```
div#navigation
 a(href='/') home
```

Listing 6.3 displays the contents of server.js that references the three Jade files in this section.

LISTING 6.3 server.js

```
var express = require('express');
var app = express.createServer();


app.configure(function () {
  app.set('views', __dirname + '/views');
  app.set('view engine', 'jade');
});


app.get('/', function(req, res) {
   res.render('index.jade',
    { pageTitle: 'Jade Example', layout: false });
});


app.listen(9000);
```

Although we have not covered Node-related code, you can see that the code creates an application, sets some configuration values, and then renders the HTML Web page that is defined by the Jade-based templates in the view subdirectory.

In slightly more detail, Listing 6.3 starts by creating an express-based application called app. Next, the code specifies the views subdirectory of the current directory (indicated by the built-in variable __dirname), and also configures Jade as the "view engine." The next portion of code specifies what to do with a GET request, as shown here:

```
app.get('/', function(req, res) {
   res.render('index.jade',
    { pageTitle: 'Jade Example', layout: false });
```

The preceding code renders a response by returning the contents of index.jade (in the views subdirectory), which is an HTML Web page with a link to the Google homepage.

The final line of code listens on port 9000 of localhost.

NOTE

You must launch this Node application by performing a 'cd' into the simple_app *directory that is on the CD.*

Other Templating Solutions

There are many other templating solutions available, and some of the more popular ones are listed here:

Mustache.js

Handlebars.js

Dust.js

jQuery.tmpl plugin

Underscore Micro-templating

PURE

You can perform an Internet search to find links and tutorials with examples that illustrate how to use the preceding templating engines.

MongoDB

MongoDB is a popular NoSQL database, and its homepage is here:

http://www.mongodb.org/

A good place to start working with MongoDB is here:

https://mongolab.com/home

Download and install MongoDB from this Website:

http://www.mongodb.org/downloads

After you install MongoDB, you can start MongoDB with the following command:

mongodb

You can also launch an interactive session by typing mongo in another command shell, which enables you to manage schemas and perform CRUD-like operations from the command line. (CRUD stands for *create, read, update,* and *delete*, the four key database operations.)

For example, the following sequence of commands shows you how to insert a new user in the users schema (which is defined in the Mongoose section later in this chapter).

```
>
> use db
switched to db db
> users = db.users
db.users
> users.find();
>
> users.insert({firstName:‘a’, lastName:‘b’});
> users.find()
{
    “_id” : ObjectId(“509425f82685d84f9c41c858”),
    “firstName” : “a”,
    “lastName” : “b”
}
```

Consult the documentation for additional details about commands that you can execute in the MongoDB command shell.

NodeJS

NodeJS is a popular server technology that executes JavaScript. You can download and install NodeJS from its homepage:

http://nodejs.org/

This section will describe only the basic information and the modules that you need in order to work with a NodeJS server used in the sample application on the CD.

After you have installed NodeJS on your machine, install express and mongoose with these commands:

```
npm install express
npm install mongoose
```

Just to show you that you can use NodeJS without a templating engine (such as Jade), Listing 6.4 displays the contents of HelloWorld.js that displays the text string “Hello World” when users navigate to localhost:5000 in a browser.

LISTING 6.4 HelloWorld.js

```
//create an app server
var express = require(“express”);
var app = express();


//send ‘Hello World’ to the client
app.get(‘/’, function(req, res) {
```

```
});


app.listen(5000);
```

Launch the application from the command line as follows:

```
node HelloWorld.js
```

Launch a browser and navigate to the URL http://localhost:5000. You will see the text string "Hello World."

Now that you have seen how to launch simple NodeJS applications (with and without templates), you are ready to create a NodeJS application that uses Mongoose and MongoDB, which is the topic of the next section of this chapter.

Mongoose

Mongoose is an ORM (written in NodeJS) for MongoDB that enables you to define schemas in a MongoDB database, and its homepage is here:

http://mongoosejs.com/index.html

Install Mongoose with the npm package manager, as shown here:

```
npm install mongoose
```

The following sections show you how to connect to a MongoDB instance via Mongoose, followed by an example of creating schemas, creating instances of the schemas, and then modifying and saving those instances to a MongoDB database.

Connecting to MongoDB via Mongoose

You can connect to MongoDB with the mongoose.connect() method. The following command is the minimum needed to connect the myapp database running locally on the default port (27017):

```
mongoose.connect('mongodb://localhost/myapp');
```

You can specify additional parameters in the uri depending on your environment:

```
mongoose.connect('mongodb://username:password@host:port/
                                database');
```

Consult the documentation for additional options for connecting to a MongoDB database.

Creating Schemas in Mongoose

The permitted SchemaTypes in Mongoose are: String, Number, Date, Buffer, Boolean, Mixed, ObjectId, and Array.

First we need to define two variables that reference Mongoose and a MongoDB database as follows:

```
var mongoose = require('mongoose'),
   db = mongoose.createConnection('localhost', 'test');
```

Now let's create a simple Person schema in Mongoose:

```
  id:       Schema.Types.ObjectId,
  firstName: String,
  lastName:  String
})
```

We can add methods to our Person schema, as shown here:

```
personSchema.methods.pname = function () {
 var fullname = this.firstname+" "+this.lastname;
 console.log(fullname);
}
```

Next we compile our Person schema into a Model as follows:

```
var Person = db.model('Person', personSchema);
```

Note that a model is just a class that enables us to construct documents. In this example, a document is a Person object.

Let's create two Person objects as follows:

```
var p1 = new Person({ firstName: 'John', lastname: 'Smith' })
var p2 = new Person({ firstName: 'Jane', lastname: 'Jones' })
```

Finally, we can save our objects in MongoDB as follows:

```
p1.save(function (error) {
 if (error) {
   console.log("error saving p1");
 }
});


p2.save(function (error) {
 if (error) {
   console.log("error saving p2");
 }
});
```

You can specify other properties in a Mongoose schema, such as indexes, setters, and getters. For example, the following code block shows you how to expand the definition of the User schema with several additional properties:

```
mongoose.model('User', {
 properties: ['first, 'last'],
 cast:      {age: Number},
 indexes:    ['first', 'last'],
 setters:    {first: function(){} },
 getters:    {full_name: function(){} },
```

```
};
```

An SPA Code Sample

The SPA code sample creates three Mongoose schemas to represent users, books, and user comments about books.

The userSchema contains information about users and the list of books that each user had read. There is a one-to-many relationship (each user can read multiple books), as shown here:

```
var userSchema = new mongoose.Schema({
  userid:    Schema.Types.ObjectId,
  firstName: String,
  lastName:  String,
  date:      {type: Date, default: Date.now},
  books:     [{type: Schema.Types.ObjectId, ref: 'Book'}]
})
```

The bookSchema contains information about books and the list of users that have read each book. Similar to userSchema, there is a one-to-many relationship (multiple users can read each book), as shown here:

```
var bookSchema = new mongoose.Schema({
  bookid:      Schema.Types.ObjectId,
  bookAuthor:  String,
  bookTitle:   String,
  bookPubDate: {type: Date},
  bookEdition: Number,
  date:        {type: Date, default: Date.now},
  users:       [{type: Schema.Types.ObjectId, ref: 'User'}]
})
```

The commentSchema contains information about the comments that users make about books that they have read. It references a user, a book, and a comment about a book, as shown here:

```
var commentSchema = new mongoose.Schema({
  commentid: Schema.Types.ObjectId,
  userid:    {type: Schema.Types.ObjectId, ref: 'User'},
  bookid:    {type: Schema.Types.ObjectId, ref: 'Book'},
  comment:   String,
  date:      {type: Date, default: Date.now}
})
```

Now we can compile our three schemas into Models as follows:

```
var User    = db.model('User', userSchema);
var Comment = db.model('Comment', commentSchema);
```

Finally, we can create objects for the three schemas. The following code block creates a new book, a new user, and a comment about the new book by the new user:

```
var b1 = new Book({bookAuthor:  'Oswald Campesato',
                   bookTitle:  'HTML5 Canvas and CSS3 Graphics',
                  bookPubDate: '07/30/2012'
                 });

var u1 = new User({firstName:  'John', lastname: 'Smith'});
var c1 = new Comment({userid: u1, bookid: b1, comment:
                                  "Great book!"});
```

Now we need to update `b1` to add user `u1` to the list of users who have read the book:

```
b1.users.push(u1);
```

Next, update `u1` to add the book `b1` to the list of books that user `u1` has read:

```
u1.books.push(b1);
```

Save the modified `b1`, `u1`, and `c1` with the following code snippet:

```
b1.save();
u1.save();
c1.save();
```

You also need code that enables users to find and update existing books and users. The following method enables us to find and update a book by its `id` value:

```
Book.findByIdAndUpdate(b1.bookid, { $set: { books: appendB1}},
  function (err, book) {
    if (err) return handleError(err);
    res.send(book);
});
```

The following method enables us to find and update a user:

```
Book.findByIdAndUpdate(b1.userid, { $set: { books: appendU1}},
  function (err, book) {
    if (err) return handleError(err);
    res.send(book);
});
```

Now you know how to do the following:

- Connect to a MongoDB database in Mongoose
- Create MongoDB schemas using Mongoose

The code in Listing 6.5 contains the code (the schema definitions are shown above and are not displayed in the code sample) for connecting to a MongoDB instance, creating schemas, populating some instances of the schemas, and then saving the instances to the database.

LISTING 6.5 book.js

```
/* server */
var express = require('express')
  , app = express.createServer()
  , mongoose = require('mongoose')
  , db = mongoose.createConnection('localhost', 'test');

/* models */
mongoose.connect('mongodb://127.0.0.1/sampledb');

var Schema = mongoose.Schema
  , ObjectId = Schema.ObjectID;

// schemas declared earlier in Mongoose section
var userSchema = …
var bookSchema = …
var commentSchema = …

var Book    = db.model('Book', bookSchema);
var User    = db.model('User', userSchema);
var Comment = db.model('Comment', commentSchema);

//A new book, a new user, and a comment
//about the new book by the new user:

var b1 = new Book({bookAuthor:  'Oswald Campesato',
                   bookTitle:   'HTML5 Canvas and CSS3 Graphics',
                   bookPubDate: '07/30/2012'
                  });

var u1 = new User({firstName:  'John', lastname: 'Smith'});
var c1 = new Comment(
       {userid: u1, bookid: b1, comment: "Great book!"});
```

```
// Add user u1 to the list of users (in b1) who have read the book:
b1.users.push(u1);

// Add book b1 to the list of books (in u1) that user u1 has read:
u1.books.push(b1);

//show books
app.get('/showbooks', function(req,res){
console.log("finding books");
    Book.find({}, function(error, data){
        res.json(data);
    });
});

//show users
app.get('/showusers', function(req,res){
console.log("finding users");
    User.find({}, function(error, data){
        res.json(data);
    });
});

// sample URL for adding a user:
//http://localhost:3003/adduser/tom/smith

app.get('/adduser/:first/:last/:username', function(req, res){
console.log("adding user");
    var user_data = {
        first_name: req.params.first
      , last_name: req.params.last
      , username: req.params.username
    };

    var user = new User(user_data);

    user.save( function(error, data){
        if(error){
```

```
        }
        else{
            res.json(data);
        }
    });
});


app.listen(3003);
console.log("listening on port %d", app.address().port);
```

Listing 6.5 consolidates many of the code fragments that you saw earlier in this chapter. There is also a code block that enables you to add new users by extracting the relevant pieces of information from a URL and then constructing the following structure that is saved to the database:

```
//sample URL for adding a user:
//http://localhost:3003/adduser/tom/smith


app.get('/adduser/:first/:last/:username', function(req, res){
  var user_data = {
        first_name: req.params.first
      , last_name: req.params.last
      , username: req.params.username
    };

    var user = new User(user_data);

    user.save( function(error, data){
        if(error){
            res.json(error);
        }
        else{
            res.json(data);
        }
    });
});
```

Now open a command shell and launch MongoDB with the following command:

```
mongod
```

Open a second shell and launch the code in Listing 6.5 as follows:

Finally, launch a browser session and navigate to `localhost:3003` to view the application.

Summary

In this chapter, you learned about the rationale for creating a single-page application (SPA). You also learned about technologies such as `Jade`, `MongoDB`, `Mongoose`, and `NodeJS` that you can use for supporting the different parts of an SPA.

INTRODUCTION TO JQUERY MOBILE

This chapter contains an introduction to jQuery Mobile, and shows you how to create HTML5-based applications using jQuery Mobile. Web pages for desktop browsers are rendered differently on mobile devices, which is immediately apparent whenever you see the tiny font size of the rendered text on a mobile device. Fortunately, jQuery Mobile is highly "page aware," and it provides many useful before-and-after page-related events that you can override with customizations that are tailored to your needs.

In fact, jQuery Mobile is designed around the notion of (mostly) single-page applications with multiple "page views" (discussed later in this chapter), whereas jQuery was designed when multi-page sites and applications were predominant. Thus, jQuery Mobile has a view-oriented model, whereas jQuery has a Web page-oriented model. This important distinction will help you understand the rationale for the features that are available in jQuery Mobile.

In this chapter, you will become well acquainted with the `data-` prefix in custom attributes, which are part of the HTML5 specification (Section 3.2.3.8). jQuery Mobile makes extensive use of custom attributes with a `data-` prefix, whereas custom attributes are not used in jQuery. In fact, jQuery Mobile uses this custom attribute for specifying behavior, functionality, and layout. As you will see, two frequently used custom attributes are `data-role` and `data-transition`.

If you want to write Web pages that display correctly on different devices, then at a minimum you need to take into account the dimensions (width and height) of those devices. Other considerations include, but are not limited to: DPI (Dots Per Inch), which varies between mobile devices, and whether or not you want to allow users to pinch or zoom into the Web page. One of the strengths of jQuery Mobile is that device differences are handled automatically for you, in addition to its rich feature set and its support for many mobile platforms.

The first part of this chapter provides an overview of some features of jQuery Mobile, as well as some important differences from jQuery. You will see how jQuery Mobile programmatically enhances your Web pages with extra functionality. This reduces the coding effort on your part in order to create mobile-enabled Web pages, and also ensures that your Web pages will render correctly on different mobile devices. In addition, you will learn about page-related events that jQuery Mobile exposes (there are many such events), and some of the default behavior that you can override programmatically.

The second part of this chapter discusses multi-page views in jQuery Mobile, and various ways for positioning headers and footers. The third part of this chapter discusses

you will see complete code samples and useful code snippets.

The fourth part of this chapter contains code samples that illustrate how to work with various widgets (including list views, navigation bars, and menus) in jQuery Mobile. The intent of these code samples is to show you not just how to use these widgets, but also how to incorporate CSS3-based graphics effects in the code samples. Some of these effects, such as shadow and gradients (which you learned how to do in Chapter 3), show you how to create a richer visual effect that goes beyond the "out of the box" functionality of jQuery Mobile. The final portion of this chapter contains code samples that show you how to use AJAX and Geolocation in HTML Web pages with jQuery Mobile.

Using jQuery 2.0 in This Chapter

The following code samples work correctly in jQuery2.0.0, but they report "spurious" warnings that you can see when you launch them in a WebKit -based browser and then open the Web Inspector:

- JQMAjax1.html
- JQMButtons1.html
- JQMFixed1.html
- JQMForm1.html
- JQMHelloWorld1.html
- JQMMenu1.html
- JQMNavigationBar1.html
- JQMNestedListViews1.html
- JQMPageEvents1.html
- JQMSimpleListView1.html
- JQMMultiPageViews1.html
- JQMGeolocation1.html

Overview of jQuery Mobile

jQuery Mobile is essentially a collection of jQuery plugins and widgets that enable you to write mobile Web applications that run on multiple platforms. You already know that jQuery focuses on desktop Web applications; by contrast, jQuery Mobile (which includes a CSS stylesheet and a JavaScript library) is intended for mobile devices. However, jQuery Mobile does rely on the "base" jQuery library that you must reference prior to referencing the jQuery Mobile library in a Web page. In addition, jQuery Mobile uses features of HTML5 and CSS3 (such as transitions and animation), and small icons for navigation.

jQuery Mobile relies on custom attributes with a data- prefix. In case you do not know, custom attributes are new in HTML5, and they always have such a prefix. This support for custom data attributes provides HTML5 markup with some of the functionality that is available in XML. This enables code to process custom tags and their values and also pass validation at the same time.

A simple jQuery Mobile page has the following structure:

- A mandatory element with a data-role="content" attribute
- An optional element with a data-role="footer" attribute

During initialization, jQuery Mobile pre-processes a Web page and inserts additional markup, CSS classes, and event handlers. You will see an example of how jQuery Mobile modifies a Web page in the "Hello World" code sample later in this chapter.

There are several important details that you need to be aware of when writing jQuery Mobile Web pages. First, jQuery Mobile provides the following page-related events that you can invoke programmatically during the lifecycle of a jQuery Mobile page: pageInit(), pageCreate(), pageShow(), and pageHide(). Second, jQuery Mobile supports custom events for handling user gestures such as swipe, tap, tap-and-hold, and orientation changes of a device. Third, jQuery Mobile uses themes to customize the look and feel of mobile applications, along with progressive enhancement (discussed briefly in Chapter 1) to enable your mobile application to run on a diverse set of Web-enabled devices.

Another key point to keep in mind is that jQuery uses this construct:

```
$(document).ready() {
  // do something here
}
```

On the other hand, jQuery Mobile uses this construct:

```
$(selector).live('pageinit', (function(event){
  // do something here
}));
```

Notice the different focus: jQuery sends an event when a Web page has been loaded, whereas jQuery Mobile sends an event when a page (or page view) has been initialized.

Key Features and Components in jQuery Mobile

If you have read the previous chapters that cover jQuery, you have already acquired substantial knowledge of jQuery features. This knowledge is useful for another reason: jQuery Mobile uses jQuery as its foundation, and the jQuery library must always precede the jQuery Mobile library in your HTML5 Web pages.

jQuery Mobile provides many useful features that will simplify the process of creating mobile applications. Some of the jQuery Mobile features are listed here:

- Compatible with major mobile platforms (Android, iOS, and others)
- Uses HTML5 markup
- Adopts progressive enhancement approach
- Provides a compact toolkit (about 12K compressed)
- Supports plugins and themes
- Supports touch and mouse-based user gestures
- Supports WAI-ARIA

In addition, jQuery Mobile supports the following components (and others that are not

- Buttons
- Form elements
- List views
- Pages and dialogs
- Toolbars

The jQuery code samples in this book use a simple naming convention: the names of HTML5 Web pages that contain jQuery code start with “JQ,” and the names of HTML5 Web pages that contain jQuery Mobile start with the letters “JQM.” Keep in mind that this naming convention is only for this book.

A Minimal jQuery Mobile Web Page

Listing 7.1 displays the contents of JQMHelloWorld1.html that illustrates how to display the message “Hello World” in an HTML5 Web page when rendered on a desktop browser, tablet, or smart phone.

LISTING 7.1 JQMHelloWorld1.html

```
<!DOCTYPE html>
<html lang=“en”>
  <head>
   <meta charset=utf-8” />
   <title>Hello World from jQuery Mobile</title>

   <link rel=“stylesheet”
    href=“http://code.jquery.com/mobile/1.1.0/
                        jquery.mobile-1.1.0.min.css” />
   <script src=“http://code.jquery.com/jquery-2.0.0b1.js”>
   </script>
   <script src=“http://code.jquery.com/jquery-migrate-1.1.0.js”>
   </script>

   <script
  src=“http://code.jquery.com/mobile/1.1.0/
                        jquery.mobile-1.1.0.min.js”>
   </script>
  </head>

  <body>
    <div  data-role=“page”>
      <div data-role=“header”>
```

```
    </div>
    <div data-role="content">
      <p>Hello World from a Simple jQuery Mobile Page</p>
    </div>
    <div data-role="footer">
      <h2>This is the Footer</h2>
    </div>
  </div>
 </body>
</html>
```

Listing 7.1 is straightforward: it consists of a single page view (an HTML element with a data-role="page" attribute) that contains three elements: a header section, the content section, and the footer section, respectively.

Figure 7.1 displays the result of rendering the HTML Web page in Listing 7.1 in a landscape-mode screenshot taken from an Asus Prime tablet with Android ICS.

Compare Figure 7.1 with Figure 7.2, which shows the sample jQuery Mobile application running on a Sprint Nexus S 4G smart phone with Android ICS in landscape mode, using the same Android apk binary that was used for capturing Figure 7.1. Notice how the header and footer extend automatically to the width of the screen in both screenshots.

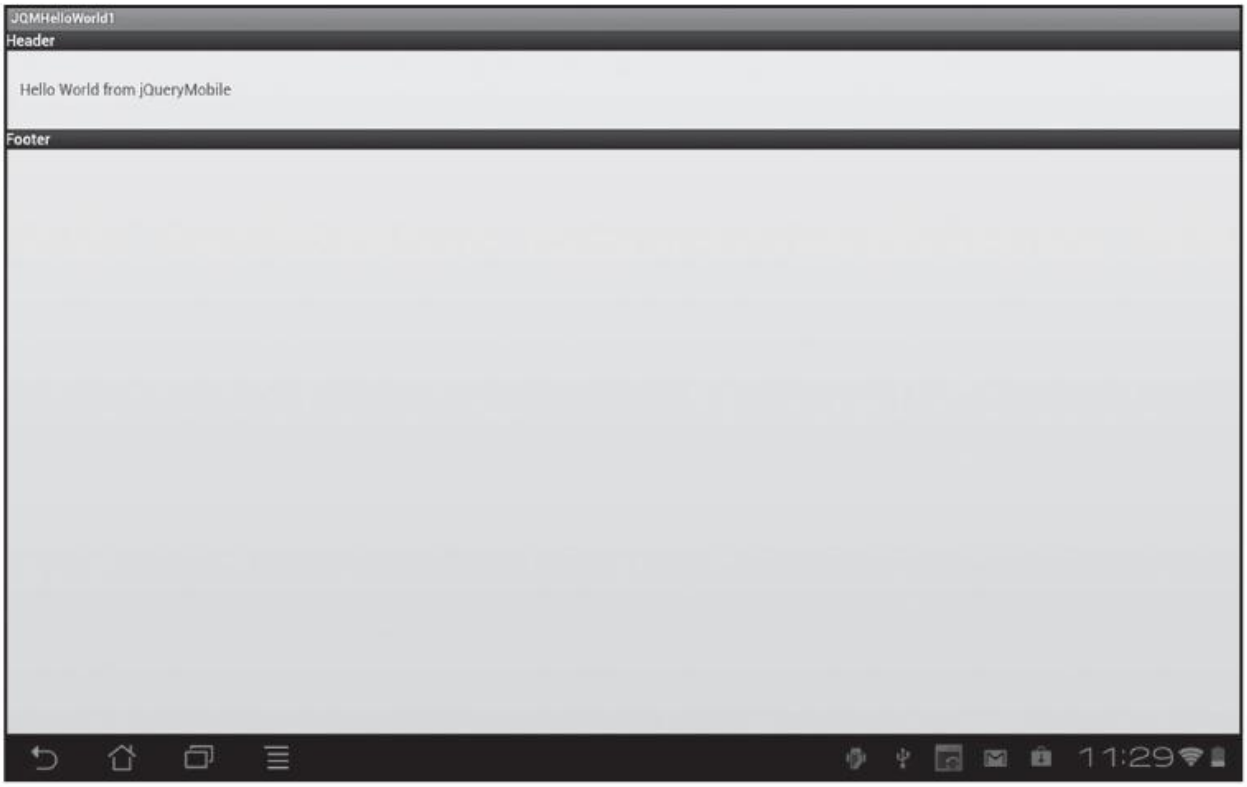


Figure 7.1 Hello World on an Asus Prime tablet with Android ICS (landscape mode).

landscape mode on a Sprint Nexus S 4G smart phone with Android ICS.

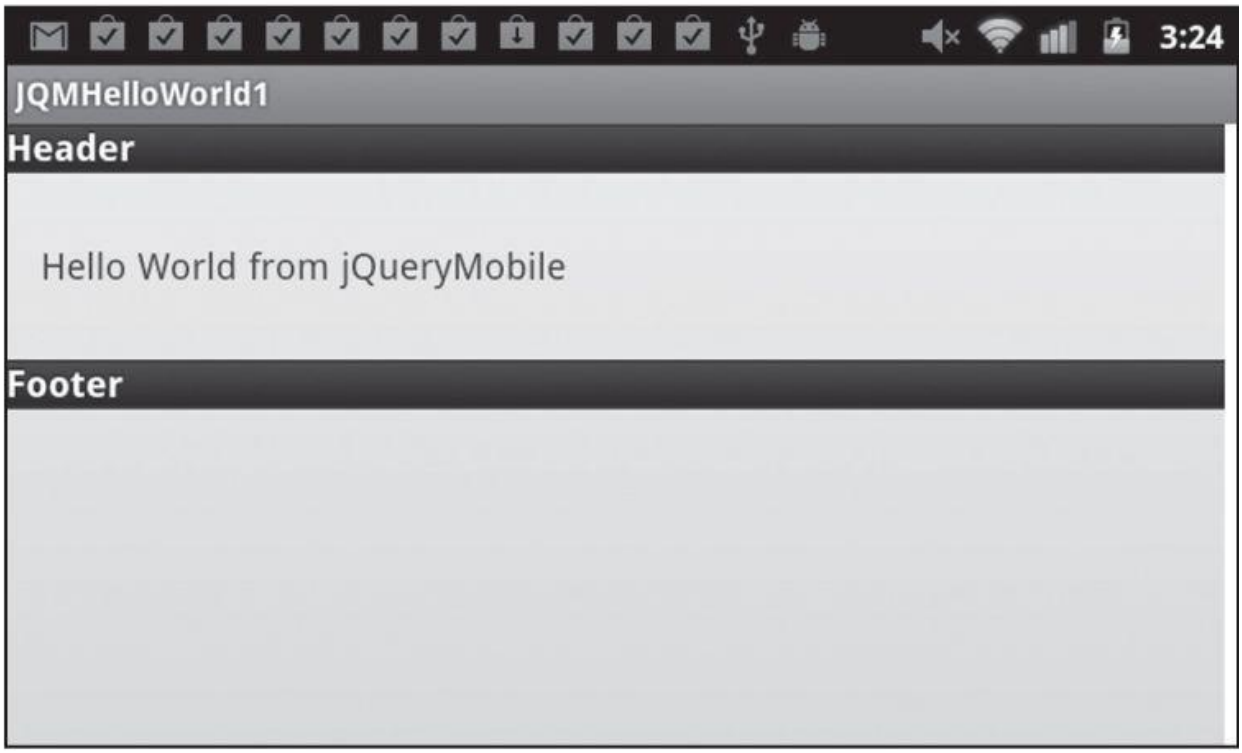


Figure 7.2 Hello World on a Sprint Nexus S 4G with Android ICS (landscape mode).

Now launch Listing 7.1 in a Webkit-based browser, and compare what you see in Figure 7.1 and Figure 7.2.

Earlier you learned that jQuery Mobile enhances a mobile Web page with additional tags and CSS classes. jQuery Mobile "injects" various CSS classes (that are part of jQuery Mobile) into the HTML5 Web page in Listing 7.1 In general, you will not need to be concerned with these details; however, you can delve into the jQuery Mobile source code if you need a deeper understanding of these details. The HTML5 page JQMHelloWorld1Enhanced.html on the CD shows you how jQuery "enhances" the contents of the HTML5 Web page HelloWorld1.html.

More Differences between jQuery and jQuery Mobile

jQuery is a toolkit for creating HTML Web pages on a desktop browser, whereas jQuery Mobile provides support for additional functionality that is relevant for mobile devices:

- Support for multiple page views
- Custom attributes with a data- prefix for page views and transitions
- Page transitions (pagebeforehide, pagebeforeshow, and so forth)
- The jqmData() custom selector
- The mobileInit event

As you saw in a previous code sample, each page view in a jQuery Mobile application is defined by an HTML element with a data-role="page" attribute, along with an optional header element, a mandatory content element, and an optional footer element.

Navigation between page views is straightforward: simply add a link to the element with the data-role="content" attribute in the jQuery Mobile application, as shown here:

```
<div data-role="content">
  <p>A second page view<a href="#home">Home</a></p>
</div>
```

The transition between page views occurs when users tap on a link, and jQuery Mobile automatically handles the necessary details of the transition. A complete example of a jQuery Mobile Web page with multiple page views (and how to navigate between the page views) is provided later in this chapter.

jQuery Mobile Custom Attributes

jQuery Mobile uses the data-role attribute to identify different parts of a "page view" (which is essentially one screen), and some of its supported values are page, header, content, and footer The data-role attribute is also used to enhance HTML elements. For example, if you specify the attribute data-role="listview" as part of the HTML <ul> tag of an unordered list, then jQuery Mobile will make the necessary enhancements (such as inserting markup, adding CSS classes, and exposing listeners) so that you can treat the unordered list as though it were a widget.

As another example, you can create a navigation bar by adding the attribute data-role="navbar" to the block-level HTML5 <nav> element, and the text strings in the associated list items (which are enclosed in the HTML5 <nav>element) are displayed as tab elements in the navigation bar.

The data-transition attribute specifies transition effects when changing page views or when displaying dialogs. Since these transitions are based on CSS3, these transitions work only in browsers that support CSS3 (such as WebKit -based browsers). The allowable values for the data-transition attribute are fade, flip, pop, slide, slidedown, and slideup.

Some of the other custom jQuery Mobile attributes are data-backbtn, data-divider, data-direction, data-icon, data-inline, data-position, data-rel, and data-url.

jQuery Mobile Page Transitions

jQuery Mobile provides page transitions (as well as the event and ui objects) that you can reference in custom code blocks that you bind to any page transition. Keep in mind that a "page" can be a separate HTML Web page as well as an HTML element inside the currently loaded HTML Web page.

Here is the sequence of page events that occurs during a page initialization:

pagebeforecreate: fires first

pagecreate: fires when DOM is populated

pageinit: after initialization is completed

pageshow: fires on ‘to’ page after transition

Whenever users tap a link that navigates to a page that is loaded for the first time, the following sequence of events occurs:

pagebeforehide: fires on the ‘from’ page before transition

pagebeforeshow: fires on the ‘to’ page before transition

pagehide: fires on the ‘from’ page after transition

pageshow: fires on the ‘to’ page after transition

Whenever a new page is loaded using AJAX, the following sequence of events occurs:

pagebeforeload: before the AJAX call is made

pageload: after the AJAX call is completed

pageloadfailed: fired if an AJAX call has failed

During page transitions, `ui.nextPage` is assigned the target page of the transition, or an empty jQuery object if there is no next page. Similarly, `ui.prevPage` is assigned the current page prior to the transition, or an empty jQuery object if there is no previous page.

jQuery Mobile uses AJAX-based asynchronous method invocations for its internal functionality, so it distinguishes page load events from page show and hide events. Page load events occur when a file is loaded into the browser in a synchronous manner, and the `jQuery(document).ready()` method is available, along with other initialization events. Note that in some cases you can explicitly specify synchronous instead of asynchronous method invocation, but this feature is not discussed in this chapter (check the online jQuery Mobile documentation if you want more details).

As you will see later in this chapter, a single HTML Web page may contain multiple jQuery Mobile page views, and users can navigate among those page views multiple times. These transitions do not fire page load events; jQuery Mobile provides a set of events that happen every time a page transition occurs.

Since the page hide and show events are triggered every time a page transition happens, make sure that you do not bind the event handlers more than once by first checking if the event handler is not already bound (otherwise do nothing), or by clearing the binding prior to rebinding to a given event.

Listing 7.2 displays the contents of `JQMPageEvents1.html` that illustrates the sequence in which page events are executed.

LISTING 7.2 JQMPageEvents1.html

```
<!DOCTYPE html>
<html lang=“en”>
<head>
 <meta charset=“utf-8”>
 <title>JQuery Mobile Page Events`</title>


 <link rel=“stylesheet”
```

```
                    jquery.mobile-1.1.0.min.css" />

 <script src="http://code.jquery.com/jquery-2.0.0b1.js">
 </script>
 <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
 </script>

 <script src="http://code.jquery.com/mobile/1.1.0/
                jquery.mobile-1.1.0.min.js">  </script>
</head>

<body>
 <div data-role="page" id="page1">
   <div data-role="header">
     <h3>JQuery Mobile Page Events</h3>
   </div>
   <div data-role="content" id="content">
   </div>
   <div data-role="footer"><h3>Footer</h3></div>
 </div>

 <script>
   $("#page1").live('pagebeforecreate', (function(event){
       console.log("pagebeforecreate event");
     })
   );

   $("#page1").live('pagecreate', (function(event){
       console.log("pagecreate event");
     })
   );

   $("#page1").live('pageinit', (function(event){
       console.log("pageinit event");
     })
   );
```

```
    console.log("pagebeforehideevent");
  })
);

$("#page1").live('pagebeforeshow', (function(event){
    console.log("pagebeforeshow event");
  })
);

$("#page1").live('pagehide', (function(event){
    console.log("pagehide event");
  })
);

$("#page1").live('pageshow', (function(event){
    console.log("pageshow event");
  })
);
</script>
</body>
</html>
```

Listing 7.2 is a simple HTML5 Web page with HTML markup and jQuery Mobile code for a single page view. The main block of code uses the jQuery live() method to bind various page-related events, which displays a message in the console whenever the page event occurs.

Figure 7.3 displays the result of rendering the HTML Web page in Listing 7.2 in the Chrome browser on a MacBook. The Chrome Inspector at the bottom of Figure 7.3 shows you the sequence of paged-related events that are fired in jQuery Mobile.

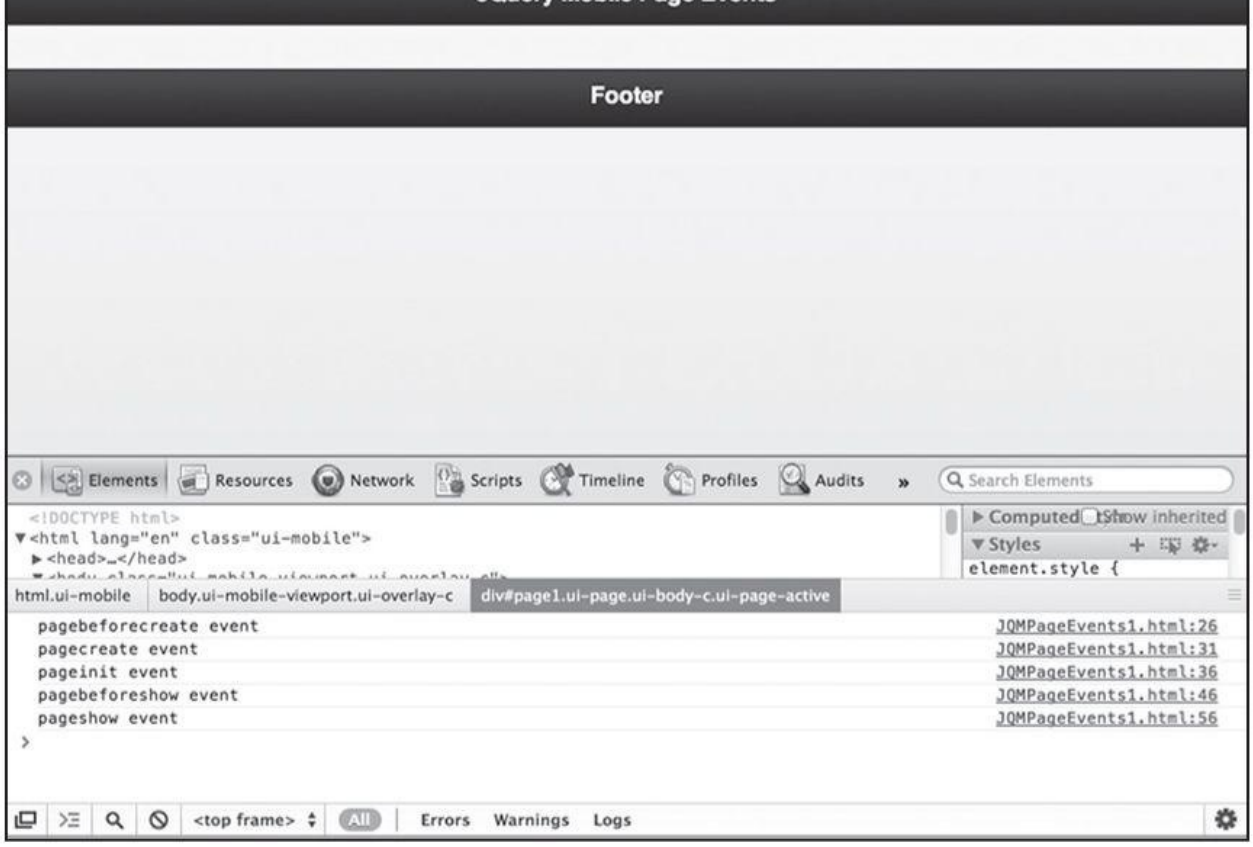


Figure 7.3 jQuery Mobile page events on the Chrome browser on a MacBook.

jQuery Mobile and CSS-Related Page Initialization

In addition to exposing page-related events, jQuery Mobile performs additional processing on an HTML Web page before the Web page is rendered in a browser. First, jQuery Mobile triggers the `beforecreate` event, then adds the `ui-page` class to all page elements, and finally adds the `ui-nojs` class to all page elements that have `data-role="none"` or `data-role="nojs"` applied to them.

Next, jQuery Mobile searches for child elements with a `data-` attribute and adds theming classes, an appropriate ARIA role, and (if necessary) also adds a back button to the header for any pages beyond the first page.

Finally, jQuery Mobile enhances buttons, control groups, and form controls, and makes any necessary adjustments to toolbars.

Thus, jQuery Mobile performs a significant amount of work on your behalf, which means that you are unencumbered with these tedious and low-level details so that you can concentrate on the functionality of your mobile applications.

There is even more good news: jQuery Mobile automatically handles page transitions and back buttons as users navigate through the various pages of your mobile application. jQuery Mobile also handles external pages by performing an asynchronous fetch (using AJAX) and then integrating that external page into the current document (and an error message is displayed if the external page was not found). The external page is incorporated into the first element with a `data-role="page"` attribute into the current document (while ignoring all other content of that page).

the Web page does not contain an element with a data-role="page" attribute. Make sure that the id values in the external Web page are distinct from the id values of the current Web page.

By the way, you can override the default page loading in two ways: specifying a target attribute on a link (such as _blank), or by specifying a rel="external" attribute on the link.

The mobileinit Event

jQuery Mobile triggers the mobileinit event on the document object immediately upon execution, so you can bind to it and override any default configuration.

For instance, suppose you need to prevent jQuery Mobile from applying styling rules to specific types of HTML elements throughout a mobile application. The following code block prevents jQuery Mobile from applying its styling rules (on a global level) to HTML <input> and <textarea> elements:

```
$(document).bind('mobileinit',function () {
  $.mobile.page.prototype.options.keepNative = "input,
                                textarea";
});
```

Note that the data-role="none" attribute serves the same purpose as the previous code block, except that it is applied only to the specific element that includes this attribute.

jQuery Mobile Options and Customization

jQuery Mobile provides options and methods for various objects, including .mobile, .mobile.path, and .mobile.history. There are many options that you can configure for your mobile application, and we'll cover just a few of them in this section. One method that is obviously useful is the jQuery Mobile pageLoading() method that shows and hides the jQuery Mobile loading dialog. You call this method with a Boolean value of true to hide the dialog, and call this method without a parameter to show the dialog, as shown here:

```
// Show the page loading dialog
$.mobile.pageLoading();


// Hide the loading dialog
$.mobile.pageLoading(true);
```

You can also customize the "loading message" and also the type of transition effect, as and shown here:

```
$.mobile.loadingMessage = "wait a few moments ";
$.mobile.defaultPageTransition = "pop";
```

Another way of doing the same thing as the previous two lines is shown here:

```
$.extent($.mobile, {
   "loadingMessage" = "wait a few moments",
   "defaultPageTransition" = "pop"
});
```

create a script that loads before jQuery Mobile is loaded, and 2) bind an event handler to the mobileinit event.

If you want more information about jQuery Mobile custom initialization, options, and methods, read the jQuery documentation for an in-depth explanation of how you can use them.

Page Navigation and Changing Pages

As users navigate around your mobile Web application, jQuery Mobile also updates the location.hash object, with the unique URL of each page view (which is defined by an element with a data-role="page" attribute) . jQuery Mobile automatically stores the URL for each page is stored in the data-url attribute, which jQuery Mobile assigns to the "container" element of a page.

jQuery Mobile also provides a set of methods that enable you to programmatically handle page changes and scrolling. One of these methods is changePage(), whose syntax looks like this:

```
changePage(to, transition, back, changeHash);
```

The to parameter is a string that specifies an element id or a filename (along with many other options), and it is a reference to the target page. The transition parameter is the name of the transition effect that is created when the application goes to the target page. The back parameter is a Boolean value that specifies whether or not a transition is in reverse. Finally, the changeHash parameter is a Boolean that specifies whether or not to update the location.hash object.

The changePage() method enables you to create more sophisticated page transition effects. For example, the following code snippet goes to page #first when users click on the back-btn, with a "flip" effect in reverse without updating the location hash:

```
$(".back-btn").bind("click", function() {
  changePage("#first", "flip", true, false);
});
```

jQuery Mobile also provides the silentScroll() method with a single integer value that specifies the y-position of the destination. When this method is invoked, the scroll event listeners are not triggered. As an example, the following code snippet scrolls down to position 200:

```
$.mobile.silentScroll(200);
```

The jqmData() Custom Selector

In Chapter 1, you learned about the jQuery data() method. jQuery Mobile provides a corresponding method called jqmData(), which is a custom selector specifically for selecting custom data- attributes.

For example, in jQuery you can select all the elements in a Web page that contain a data-role attribute whose value is page using this code snippet:

```
$("[data-role='page']")
```

```
$(":jqmData(role='page')")
```

Select all elements with any custom `data-` attribute within those selected pages:

```
$(":jqmData(role='page')").jqmData(role)
```

Note that the `jqmData()` selector automatically handles namespacing for you by specifying a value for the string `namespace-` (which is empty by default), thereby avoiding tagname collisions.

Multiple Page Views in One HTML5 Web Page

jQuery Mobile enables you to conveniently define multiple page views in a single HTML5 Web page, along with a simple mechanism for users to navigate among the different page views in the HTML5 Web page. The use of a single HTML5 Web page is recommended because this approach is more efficient than creating a mobile application with multiple HTML5 Web pages. Although the initial download for the HTML5 Web page might be longer, there are no additional Internet accesses required when users navigate to different parts of the Web page.

Listing 7.3 displays the contents of `JQMMultiPageViews1.html` that illustrates how to navigate between multiple internal page views in a single HTML5 Web page.

LISTING 7.3: JQMMultiPageViews1.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset=utf-8" />
  <title>jQuery Mobile: Multiple Page Views</title>

  <link rel="stylesheet"
    href="http://code.jquery.com/mobile/1.1.0/jquery.
                              mobile-1.1.0.min.css" />

  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>

   <script src="http://code.jquery.com/mobile/1.1.0/
                            jquery.mobile-1.1.0.min.js">
   </script>
</head>
```

```
<div data-role="page" id="home">
  <div data-role="header"> <h1>Home Page Header </h1></div>
  <div data-role="content">
    <p>This is the content of the main page</p>
    <p><a href="#about">Click here to get more
                              information</a></p>
  </div>
  <div data-role="footer"> <h1>Home Page Footer</h1></div>
</div>

<div data-role="page" id="about">
  <div data-role="header"><h1>About This Page Header</h1></div>
  <div data-role="content">
    <p>A second page view (that's all for now)</p>
    <a href="#home"> Click Here or the 'Back' Button to go
                                    Home</a>
  </div>
  <div data-role="footer"><h1>About This Page Footer</h1></div>
</div>
</body>
</html>
```

Listing 7.3 contains two page views, as specified by the HTML `<div>` elements whose `id` attribute has values `home` and `about` (shown in bold in Listing 7.3). When users navigate to the second page view, this code snippet returns to the first page view:

```
<a href="#home"> Click Here or the 'Back' Button to go Home</a>
```

The page views or screens in a jQuery Mobile application are top-level sibling elements, each of which contains the attribute `data-role="page"` (so pages cannot be nested).

The jQuery Mobile framework automatically generates a back button and a home button on every page view, but you can suppress the back button by specifying the attribute `data-backbtn="false"` attribute, as shown here:

```
<div data-role="header" data-backbtn="false">
  <h1>Page Header</h1>
</div>
```

Figure 7.4 displays the result of rendering the HTML Web page in Listing 7.3 in a landscape-mode screenshot taken from a Sprint Nexus S 4G with Android ICS.

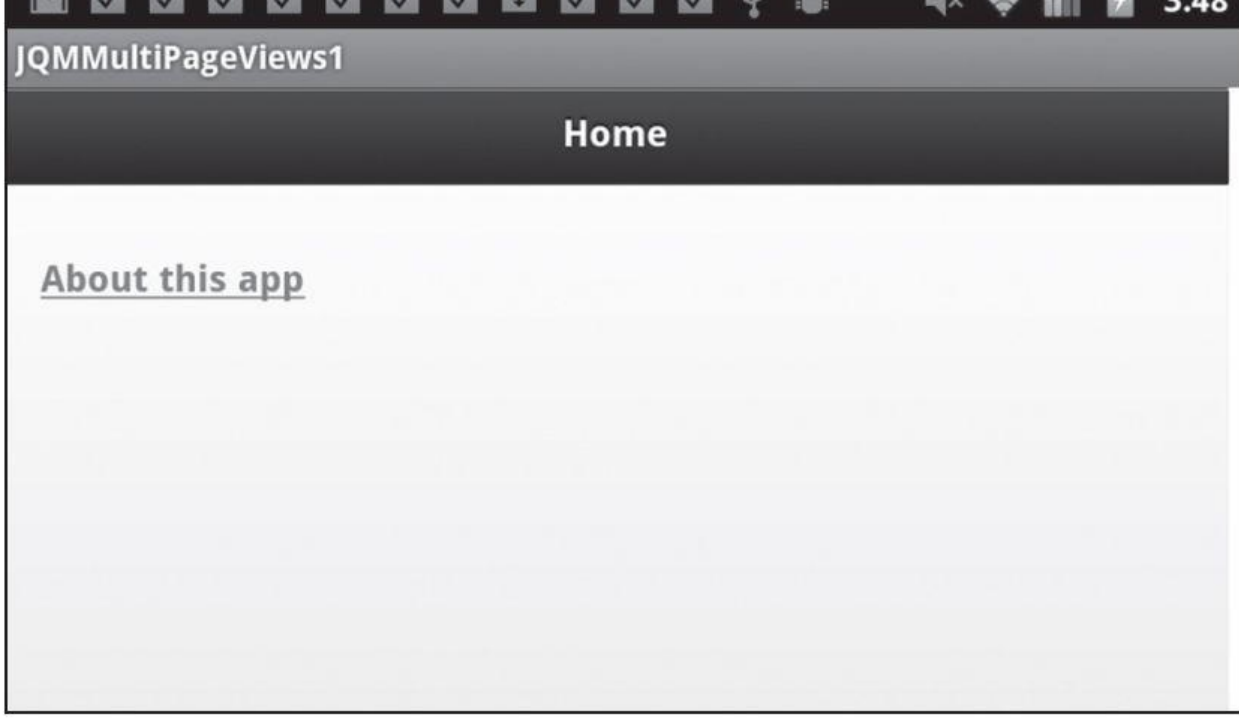


Figure 7.4 Multi-page app on a Sprint Nexus S 4G with Android ICS.

When you click on the link on the Web page, the application navigates to the second screen, whose code definition is also included in Listing 7.3 .

Positioning the Header and Footer in Page Views

jQuery Mobile can dynamically position the header and footer toolbars in three ways:

Standard: The toolbars are presented according to the document flow, scrolling into and out of the viewport as the user scrolls through data. This is the default.

Fixed: The header and footer will appear at the top and bottom of the viewport and remain there as the user scrolls. Tapping on the screen will cause them to return to their regular position in the document flow.

Fullscreen: The header and footer will appear within the viewport and stay present as the user scrolls, regardless of where the user is in the content. Tapping on the screen will hide them. Essentially, the header and footer are removed from the document flow and are always dynamically positioned at the top and bottom of the viewport.

Include the data-position="fixed" attribute in the <header> or <footer> tags to create a fixed header and footer, as illustrated in Listing 7.4 that displays the contents of JQMFixed1.html.

LISTING 7.4 JQMFixed1.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8" />
  <title>Fixed Header and Footer in jQueryMobile</title>
```

```
  <link rel="stylesheet"
    href="http://code.jquery.com/mobile/1.1.0/jquery.
                                mobile-1.1.0.min.css" />
  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>
   <script src="http://code.jquery.com/mobile/1.1.0/
                            jquery.mobile-1.1.0.min.js">
   </script>
 </head>

 <body>
   <div id="page1" data-role="page">
     <header data-role="header" data-position="fixed">
       <h1>jQuery Mobile</h1>
     </header>
     <div class="content" data-role="content">
       <h3>Content area.</h3>
     </div>
     < footer data-role="footer" data-position="fixed">
       <h3>Fixed Footer</h3>
     </footer>
   </div>
  </body>
 </html>
```

Listing 7.4 contains HTML markup and a single page view that also specifies the attribute `data-position="fixed"` for the header and footer elements.

You can also create a fullscreen header or a fullscreen footer by including the attribute `data-fullscreen="true"` in the element that contains the `data-role="page"` attribute, along with the attribute `data-position="fixed"` to the header and footer elements, as shown here:

```
 <section id="page1" data-role="page" data-fullscreen="true">
   <header data-role="header" data-position="fixed">
     <h1>jQuery Mobile</h1>
   </header>
   <div class="content" data-role="content">
     <h3>Content area</h3>
```

```
  <footer data-role="footer" data-position="fixed">
    <h3>Mercury Learning </h3></footer>
  </div>
</section>
<p><a href="#about"data-transition="flip">About this app</a></p>
```

Figure 7.5 displays the result of rendering the HTML5 Web page in Listing 7.5, in the Chrome browser on a MacBook.

Figure 7.5 Fixed headers/footers in the Chrome browser on a MacBook.

Launch the HTML5 Web page in Listing 7.5 in the Chrome browser. As you resize your browser horizontally or vertically, notice how the header and footer "bar" remain anchored to the top and the bottom of the screen, respectively.

Now that you understand the structure of an HTML5 Web page in jQuery Mobile, let's see how to add jQuery Mobile buttons to an HTML5 Web page.

Working with Buttons in jQuery Mobile

jQuery Mobile provides a set of button markup options that enable you to style links as buttons, and also built-in support for automatically handling various types of HTML <input> elements as though they were buttons. In addition, jQuery Mobile supports inline buttons, grouped buttons (both vertical and horizontal), and an assortment of theming effects for buttons.

jQuery Mobile creates stylized buttons by applying data-role="button" to HTML <input> buttons, <button> tags, and anchor links. Buttons are as wide as their containing element; however, if you specify the attribute data-inline="true", then buttons are rendered only as wide as their content.

In some of the earlier versions of jQuery Mobile (prior to 1.7), you could prevent

an HTML element located inside the HTML5 <header> element. However, this functionality is no longer available in version 1.7 and beyond; keep this in mind in case you encounter this functionality in Web pages that use older versions of jQuery Mobile.

You can also create a reverse transition without going back in history by using the attribute data-direction="reverse" instead of data-rel="back", as shown here:

```
<div data-role="header">
  <a href="index.html" data-direction="reverse">Reverse Transition</a>
  <h1>Left and Right Buttons</h1>
  <!— This appears on the right —>
  <a href="info.html" data-icon="plus">Add</a>
</div>
```

Buttons can be moved to the left or right, as shown here:

```
<div data-role="header">
<a href="index.html" class="ui-btn-right">Right</a>
 <h1>Page title</h1>
<a href="info.html" class="ui-btn-left">Left</a>
</div>
```

Navigation Buttons as Anchor Links

In jQuery Mobile, the recommended practice is to define navigation buttons as *anchor links* and to define form submission buttons as *button elements* .

An anchor link that contains the attribute data-role="button" is styled as a button, as shown here:

```
<a href="external.html" data-role="button">Link-based button</a>
```

A jQuery Mobile button element or input element whose type is submit, reset, button, or image does not require the attribute data-role="button" because jQuery Mobile automatically treats these elements as a link-based button.

Although the original form-based button is not displayed, it is included in the HTML markup. When a click event occurs on a link button, a corresponding click event occurs on the original form button.

Groups of Buttons and Column Grids

jQuery Mobile enables you to render a set of buttons (or checkboxes or radio buttons) as a block by defining a container HTML element that specifies the attribute data-role="controlgroup", and then include one or more button elements. The default rendering of the included buttons is horizontal (a row), and you can render the buttons as a vertical group by specifying the attribute data-type="vertical" .

jQuery Mobile renders the buttons, removes margins (and drop shadows) between buttons, and also "rounds" the first and last buttons in order to create a group-like effect. The following code block shows you how to specify a horizontal group of buttons:

```
<a href="index1.html" data-role="button">One</a>
<a href="index2.html" data-role="button">Two</a>
<a href="index3.html" data-role="button">Three</a>
</div>
```

One point to keep in mind: the buttons will "wrap" to additional rows if the number of buttons exceeds the screen width. Listing 7.5shows you how to render a horizontal group and a vertical group of buttons.

The jQuery Mobile framework enables you to render CSS-based columns through a block style class convention called ui-grid. You can render two-column, three-column, four-column, and five-column layouts by using the class value ui-grid-a, ui-grid-þ ui-grid-ç and ui-grid-d , respectively. The following code block illustrates how to render a two-column layout:

```
<div class="ui-grid-a">
<div class="ui-block-a">Block 1 and text inside will wrap</div>
<div class="ui-block-b">Block 2 and text inside will wrap</div>
</div>
```

Notice the difference in the following code block that displays a three-column layout:

```
<div class="ui-grid-b">
<div class="ui-block-a">Block 1 and text inside will wrap</div>
<div class="ui-block-b">Block 2 and text inside will wrap</div>
<div class="ui-block-c">Block 3 and text inside will wrap</div>
</div>
```

You can find more details and code samples here:

http://jquerymobile.com/demos/1.0b1/#/demos/1.0/docs/content/content-grids.html

Rendering Buttons with Themes

jQuery Mobile has an extensive theming system, involving themes labeled "a" through "e," for controlling the manner in which buttons are styled. When a link is added to a container, jQuery Mobile will assign it a theme "swatch" letter that matches the theme of its parent. For example, a button placed inside a content container with a theme of "a" (black in the default theme) is automatically assigned the button theme of "a" (charcoal in the default theme).

An example of a button theme is shown here:

```
<div data-role="footer" class="ui-bar">
<a href="left.html" data-icon="arrow-l"
data-role="button" data-theme="a">Left</a>
</div>
```

Listing 7.5displays the contents of JQMButtons1.html , which illustrates how to render buttons horizontally and vertically, how to apply different themes, and how to specify

LISTING 7.5 JQMButtons1.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
   <meta charset="utf-8" />
   <title>Multiple Buttons and Themes in jQuery Mobile</title>

   <link rel="stylesheet" href="CSS3Background1.css" />

   <link rel="stylesheet"
    href="http://code.jquery.com/mobile/1.1.0/
                          jquery.mobile-1.1.0.min.css" />
  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>

   <script src="http://code.jquery.com/mobile/1.1.0/
                                jquery.mobile-1.1.0.min.js">
   </script>
  </head>

  <body>
    <div id="page1" data-role="page">
      <div data-role="header">
        <!— This button appears on the left —>
        <a href="abc.html" data-icon="arrow-l"
                      data-theme="b">Index</a>

        <h1>Rendering Buttons in jQuery Mobile</h1>

        <!— This button appears on the right —>
        <a href="def.html" data-icon="plus" data-theme="e">Add</a>
      </div>

      <div class="content" data-role="content">
        <div data-role="controlgroup" data-type="horizontal">
```

```
    <a href="index2.html" data-role="button">Two</a>
    <a href="index3.html" data-role="button">Three</a>
    <a href="index4.html" data-role="button">Four</a>
    <a href="index5.html" data-role="button">Five</a>
  </div>

    <a href="index6.html" class="shadow"
      data-role="button">Vertical1</a>
    <a href="index7.html" class="shadow"
      data-role="button">Vertical2</a>
    <a href="index8.html" class="shadow"
      data-role="button">Vertical3</a>

  <div data-role="controlgroup">
    <a href="index6.html" data-role="button">Vertical1</a>
    <a href="index7.html" data-role="button">Vertical2</a>
    <a href="index8.html" data-role="button">Vertical3</a>
  </div>

  <div>
    <a href="index9.html" data-role="button"  id="cancel"
      data-inline="true">Cancel</a>
    <a href="index10.html" data-role="button" id="submit"
      data-inline="true" data-theme="b">Save</a>
  </div>
</div>

<!— Display four directional array keys —>
<div data-role="footer" class="ui-bar">
 <a href="left.html" data-icon="arrow-l"
   data-role="button" data-theme="a">Left</a>
 <a href="up.html" data-icon="arrow-u"
   data-role="button" data-theme="b">Up</a>
 <a href="down.html" data-icon="arrow-d"
   data-role="button" data-theme="c">Down</a>
 <a href="right.html" data-icon="arrow-r"
   data-role="button"data-theme="e">Right</a>
```

```
    </div>
  </body>
</html>
```

Listing 7.5 contains HTML markup and references to jQuery files, followed by a single page view that renders an assortment of buttons (they have a data-role="button" attribute), some of which are in a horizontal group, while others are in a vertical group.

Notice that the vertical group of buttons is rendered with a reddish-tinged background shadow, which creates a richer visual effect. This is accomplished simply by specifying a value of shadow for the class attribute, as shown in bold in this code block:

```
<a href="index6.html" class="shadow"
        data-role="button">Vertical1</a>
<a href="index7.html" class="shadow"
        data-role="button">Vertical2</a>
<a href="index8.html" class="shadow"
        data-role="button">Vertical3</a>
```

In addition, the "cancel" and "submit" buttons specify class="shadow", and they are rendered with a psychedelic effect, which you obviously don't want to render in most cases. Nevertheless, you see how easily you can style buttons with custom CSS3-based visual effects, and the definition of the shadow selector is shown in Listing 7.6.

The bottom row of four HTML <a> elements in Listing 7.5 renders four directional arrows because they have a data-icon attribute that is set to arrow-l, arrow-u, arrow-d, and arrow-r, respectively. In addition, these four HTML <a> elements specify different values for the attribute data-theme, and they are rendered according to the visual display for each of these pre-defined jQuery Mobile themes.

Note that the HTML <a> elements in Listing 7.6 reference HTML Web pages that are not defined for this mobile application. Thus, when users click on these links, they will see an error message.

LISTING 7.6 CSS3Background1.css

```
#cancel, #submit, .shadow {
 background-color:white;
 background-image:
  -webkit-radial-gradient(red 4px, transparent 18px),
  -webkit-repeating-radial-gradient(red 2px,  green 4px,
                    yellow 8px, blue 12px,
                    transparent 28px,
                    green 20px, red 24px,
                    transparent 28px,
```

```
    -webkit-repeating-radial-gradient(red 2px,  green 4px,
                                      yellow 8px, blue 12px,
                                      transparent 28px,
                                      green 20px, red 24px,
                                      transparent 28px,
                                      transparent 32px);


    background-size: 50px 60px, 70px 80px;
    background-position: 0 0;
    -webkit-box-shadow:  30px 30px 30px #400;
   }
```

Listing 7.6 contains definitions that are familiar to you from Chapter 2. You can refer to the relevant sections in that chapter if you need to refresh your memory.

Figure 7.6 displays the result of rendering the HTML Web page in Listing 7.5 on a Sprint Nexus S 4G with Android ICS.

Figure 7.6 Buttons on a Sprint Nexus S 4G with Android ICS.

As you can see, there are many styling-related options available in jQuery mobile, so a summary of the discussion about jQuery Mobile buttons might be helpful:

- An anchor link with the attribute data-role="button"is rendered as a button.
- Any input element with the type button, submit, reset, or image is assigned a class of

- Add a `data-icon` attribute to create a button with an icon.
- Add a `data-iconpos` attribute to change the position of the icon.
- Specify `data-inline="true"` to render a button only as wide as its text.
- `data-role="controlgroup"` (`data-type="horizontal"`) groups buttons vertically (horizontally) in a `<div>` element.
- `data-theme="e"` attribute is the fifth of five jQuery themes ("a" through "e").

List Views in jQuery Mobile

jQuery Mobile supports list views (which are very common in mobile applications) and several variations, including list view buttons, nested list views, and list view split buttons. jQuery Mobile can enhance ordered lists and unordered lists by applying the attribute `data-role="listview"` to a list. List view elements that are embedded inside anchor tags respond to user gestures, as shown in Listing 7.7, which displays the contents of the jQuery Mobile Web page `JQMSimpleListView1.html`.

LISTING 7.7 JQMSimpleListView1.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
   <meta charset=utf-8" />
   <title>Simple List Views in jQuery Mobile</title>

   <link rel="stylesheet"
    href="http://code.jquery.com/mobile/1.1.0/
                        jquery.mobile-1.1.0.min.css" />
  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>
   <script src="http://code.jquery.com/mobile/1.1.0/
                          jquery.mobile-1.1.0.min.js">
   </script>
  </head>

  <body>
    <div data-role="page">
      <div data-role="header"> <h2>This is the Header</h2></div>
      <div data-role="content">
        <h3>Simple Unordered List Example</h3>
        <ul data-role="listview">
```

```
      <li><a href="#">Unordered Item 2</a></li>
      <li><a href="#">Unordered Item 3</a></li>
    </ul>
    <h3>Simple Ordered List Example</h3>
    <ol data-role="listview">
      <li><a href="#">Ordered Item 1</a></li>
      <li><a href="#">Ordered Item 2</a></li>
      <li><a href="#">Ordered Item 3</a></li>
    </ol>
  </div>
  <div data-role="footer"><h2> This is the Footer</h2> </div>
 </div>
</body>
</html>
```

Whenever users tap on any list item in Listing 7.7, you can detect that event and execute custom code (if any) that you have bound to that event. In this example, the `href` attribute has value `#` so the page is reloaded when users tap or click on the list items.

Figure 7.7 displays the result of rendering the HTML Web page in Listing 7.7 in landscape mode on an Asus Prime tablet with Android ICS.

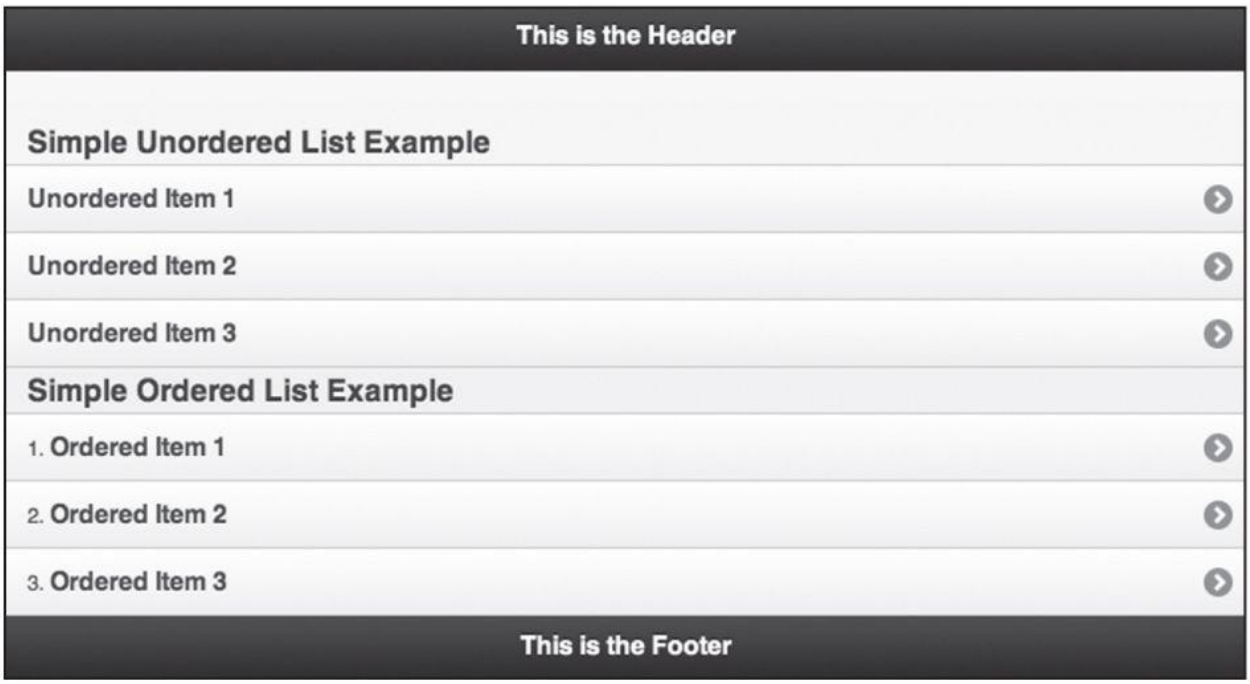


Figure 7.7 List view on an Asus Prime Tablet with Android ICS.

Additional Code Samples on the CD

jQuery Mobile can create interactive views with nested lists, and users can "drill down" by tapping on list items. The first view displays the items in the top-level list, and tapping on one of those items will display the sublist for that item (and so forth). In addition, jQuery Mobile automatically provides a back button and also takes care of the URL mapping and transitions between pages.

The HTML5 Web page JQMNestedListViews1.html on the CD is a jQuery Mobile Web page that illustrates how to render nested lists and how to navigate between pages when users click on list items.

Navigation bars in mobile applications often consist of a set of buttons that enable users to navigate through the page views. jQuery Mobile allows you to include navigation bars in the header, footer, or content areas of a page view (and also provide the appropriate formatting for the navigation bars).

To designate a navigation bar, apply the data-role="navigation" to a block level element like the HTML 5 <nav> element. Anchor tags contained within a designated navigation element will be formatted as a button group. jQuery Mobile will automatically handle changing the active and inactive states of the buttons, as shown in the HTML5 Web page JQMNavigationBar1.html on the CD.

jQuery Mobile enables you to show and hide a list of menu options, and you can also use transitions to create animation effects as the menu list is displayed or hidden from view. The HTML5 Web page JQMMenu1.html on the CD illustrates how to create a sliding menu in an HTML5 Web page.

jQuery Mobile and AJAX

jQuery Mobile uses AJAX for form submission, and will attempt to integrate the server response into the DOM of the application, providing transitions as expected. If you wish to prevent jQuery Mobile from using AJAX to handle a form, apply the attribute data-ajax="false" to the form tag.

Listing 7.8 displays the contents of JQMAjax1.html that illustrates how to handle AJAX invocations in a jQuery Mobile application.

LISTING 7.8 JQMAjax1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>jQuery Mobile and AJAX</title>


  <link rel="stylesheet"
```

```
 href=“http://code.jquery.com/mobile/1.1.0/
                        jquery.mobile-1.1.0.min.css” />
 <script src=“http://code.jquery.com/jquery-2.0.0b1.js”>
 </script>
 <script src=“http://code.jquery.com/jquery-migrate-1.1.0.js”>
 </script>
 <script src=“http://code.jquery.com/mobile/1.1.0/
                          jquery.mobile-1.1.0.min.js”>
 </script>
</head>

<body>
  <script>
   var xmlData = ””, count = 0;

   $(“#page1”).live(‘pageinit’, (function(event){
     $.ajax({
       url: ‘http://localhost:9000/sample.xml’,
       dataType: ‘xml’,
       success: function(data) {
                $(data)
                  .find(‘svg’)
                  .children()
                  .each(function() {
                    var node = $(this);
                    var x      = node.attr(‘x’);
                    var y      = node.attr(‘y’);
                    var width  = node.attr(‘width’);
                    var height = node.attr(‘height’);
                    var stroke = node.attr(‘stroke’);
                    var fill   = node.attr(‘fill’);

                    // other processing here…
                    ++count;

                    console.log(“element: “+node);
                });
                console.log(“element count: “+count);
```

```
        error: function() {
          alert('no data found');
        }
       })
      })
     );
    </script>

    <div id="page1" data-role="page">
      <div data-role="header">Header</div>
      <div data-role="content">
        <p>Hello World from jQuery Mobile</p>
      </div>
      <div data-role="footer">Footer</div>
    </div>
  </body>
</html>
```

Listing 7.8 contains HTML markup and a jQuery Mobile AJAX invocation that is executed after this Web page is loaded into a browser. Then, the code processes all the child elements of the <svg> element in the XML document sample.xml (displayed in Listing 7.9), as shown here:

```
var node = $(this);
var x      = node.attr('x');
var y      = node.attr('y');
var width  = node.attr('width');
var height = node.attr('height');
var stroke = node.attr('stroke');
var fill   = node.attr('fill');
```

For the purpose of illustration, this code makes the assumption that the XML document sample.xml in Listing 7.9 contains XML elements that have the attributes specified in the preceding code block.

LISTING 7.9 sample.xml

```
<?xml version='1.0' encoding='iso-8859-1'?>
<svg xmlns="http://www.w3.org/2000/svg"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    width="100%" height="100%">
 <rect x="50"  y="10" width="100" height="200"
```

```
  <rect x="200" y="10" width="100" height="200"
      stroke="blue" fill="green" />
  <rect x="350" y="10" width="100" height="200"
      stroke="blue" fill="blue" />
</svg>
```

Place the files in Listing 7.8 and 7.9 in the same directory, and launch a Web server that serves requests on port 9000. Launch the HTML Web page in Listing 7.8 in the Chrome browser on a laptop or desktop, open Chrome Inspector, and inspect the contents of the Chrome console. You will see something similar to the contents of Figure 7.8, which displays the contents of the Chrome Inspector in the Chrome browser on a MacBook.

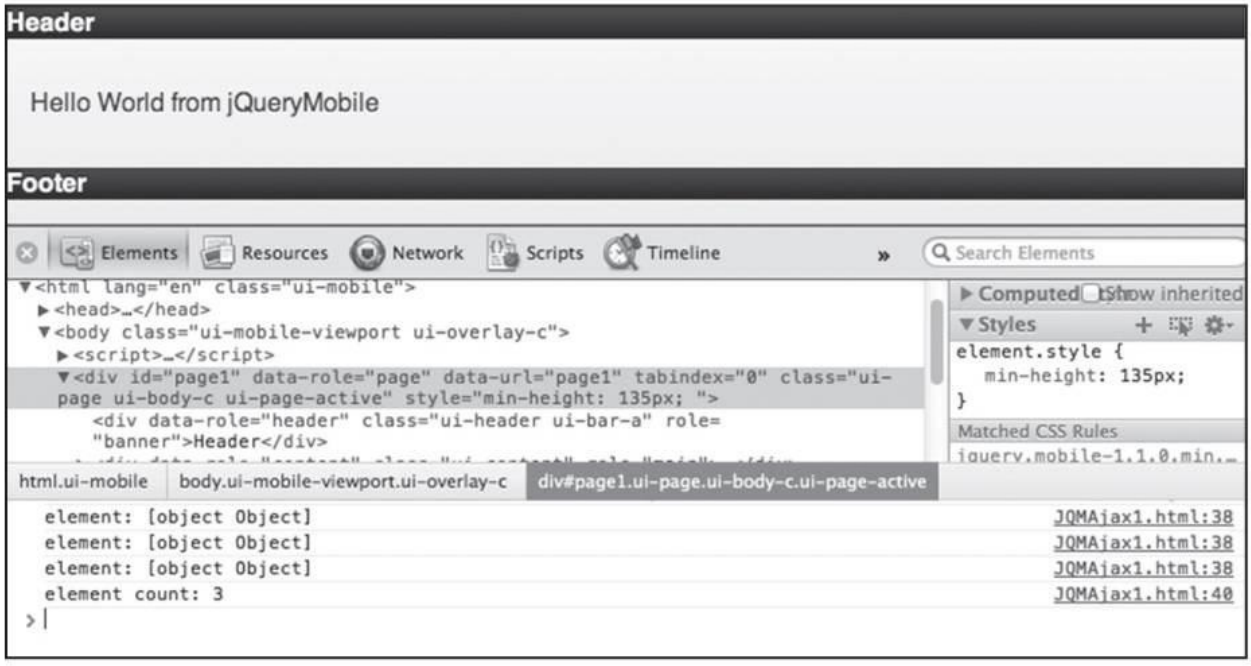


Figure 7.8 A jQuery Mobile AJAX invocation on a Chrome browser on a MacBook.

jQuery Mobile and Geolocation

In Chapter 5, you learned how to obtain Geolocation information for users. In this section, you will see how to use jQuery Mobile in order to obtain Geolocation information.

Listing 7.10 displays the contents of JQMGeoLocation1.html that illustrates how to obtain Geolocation information in an HTML5 Web page.

LISTING 7.10 JQMGeoLocation1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
```

```
  <link rel="stylesheet"
   href="http://code.jquery.com/mobile/1.1.0/
                        jquery.mobile-1.1.0.min.css" />
  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>
  <script src="http://code.jquery.com/mobile/1.1.0/
                          jquery.mobile-1.1.0.min.js">
  </script>

  <!— google maps API —>
  <script
    src="http://maps.google.com/maps/api/js?sensor=false">
  </script>

  <style>
#theMap, #theCoords {
font-size: 16px;
height: 200px;
width:  400px;
}
  </style>

<script>
  function findUserLocation() {
    // specify the 'success' and 'fail' JavaScript functions
    navigator.geolocation.getCurrentPosition(successCallback,
                                errorCallback);
  }

function successCallback(position) {
  var latitude  = position.coords.latitude;
  var longitude = position.coords.longitude;
  var latlong   = new google.maps.LatLng(latitude, longitude);
```

```
      zoom: 14,
      center: latlong,
      mapTypeId: google.maps.MapTypeId.ROADMAP
  };

  // use Google Maps to display the current location
  var map = new google.maps.Map(document.getElementById("theMap"), myOptions);
  map.setCenter(latlong);

/*
    var marker = new google.maps.Marker({
      position: initialLocation,
      map: map,
      title: "I Am Here!"
  });
*/

  // display position details in the console
  positionDetails(position);
}

function errorCallback(error) {
  if(error.code = error.PERMISSION_DENIED) {
     console.log("Error: you must enable geolocation access");
  } else if(error.code = error.PERMISSION_UNAVAILABLE) {
     console.log("Error: geolocation unavailable");
  } else if(error.code = error.TIMEOUT) {
     console.log("Error: timeout occurred");
  }

//console.log(error);
}

function positionDetails(pos) {
 var positionStr =
   "Latitude:"+ pos.coords.latitude +"<br>"+
   "Longitude:"+ pos.coords.longitude +"<br>"+
```

```
  "Altitude:"+ pos.coords.altitude +"<br>"+
  "AltitudeAccuracy:"+ pos.coords.altitudeAccuracy +"<br>"+
  "Heading:"+ pos.coords.heading +"<br>"+
  "Speed:"+ pos.coords.speed +"";

  $("#theCoords").html(positionStr);
  console.log(positionStr);
}
</script>
</head>

 <body>
   <div data-role="page" id="page1">
     <div data-role="header"><h3>JQuery Mobile
                          Geolocation</h3></div>
     <div data-role="content" id="content">
       <div id="theMap"> </div>

       <form id="geoLocationForm">
         <input type="button" id="geobutton"
              value="Click to Find My Current Location">
       </form>
       <div id="theCoords"> </div>
     </div>
     <div data-role="footer"><h3>Footer</h3></div>
   </div>

   <script>
     $("#page1").live('pageinit', (function(event){
       $("#geobutton").bind("vmousedown",function(event, ui){
         findUserLocation();
       });
     })
    );
   </script>
  </body>
```

Listing 7.10 contains code that is very similar to the code sample that you saw in Chapter 5. The only difference is that the code in Listing 7.10has been adapted to jQuery Mobile, whereas the code in Chapter 5 uses jQuery. As such, you can read the explanation that is provided immediately after the related code sample in Chapter 5.

If you are interested in mobile maps, there are many such examples available, including the details on this Website:

http://jquery-ui-map.googlecode.com/svn/trunk/demos/jquery-google-maps-mobile.html

Summary

In this chapter, you learned about various features of jQuery Mobile, along with code samples that showed you examples of how to handle events, create Web pages with multiple page views, and how to create forms with HTML widgets. You also saw how to use CSS3 shadow and gradient effects (which you learned in Chapter 3) to style buttons in jQuery Mobile Web pages. You learned how to do the following in HTML Web pages that use jQuery Mobile:

- Render buttons
- Display simple and nested lists
- Render sliding menus
- Use jQuery Mobile with AJAX
- Use jQuery Mobile with Geolocation

USER GESTURES AND ANIMATION EFFECTS IN JQUERY MOBILE

This chapter contains a number of code samples that demonstrate how to handle user gestures and also how to create animation effects in jQuery Mobile. Fortunately, jQuery Mobile emits events for user gestures, such as orientationchange, scrollstart, scrollstop, swipe, swipeleft, swiperight, tap, and taphold. The first part of this chapter shows you how to handle some of these user gestures. You will also see code samples that show you how to create slide-related and fade-related animation effects in jQuery. The final portion of this section briefly discusses jQuery Mobile virtual mouse events, which can simplify your code when you want to handle mouse-related events as well as touch-related events in an HTML Web page.

The second part of this chapter shows you how to create CSS3 2D/3D animation effects with jQuery Mobile. One code sample uses jQuery Mobile to create a 3D cube effect. Also, there are samples on the CD that render "bouncing balls," and show how to access accelerometer values for a mobile device and then display the real-time values in a jQuery Mobile Web page.

Recall that in Chapter 2 you learned how to create CSS3 2D and 3D effects, and in Chapter 3 you learned how to combine those effects with jQuery in HTML5 Web pages. In this chapter, you will see how to create some of the corresponding effects using jQuery Mobile.

Handling User Gestures and Events in jQuery Mobile

jQuery Mobile emits an assortment of events for user gestures, such as orientationchange, scrollstart, scrollstop, swipe, swipeleft, swiperight, tap, and taphold.

If you need to handle many user gestures, consider the option of using a jQuery plugin instead of writing your own code. There are various plugins available, and the next section briefly discusses two of those plugins.

When you want to detect user gestures on a HTML Web page, you can write custom jQuery code using the jQuery .on("click") method to handle those gestures. Please review the section in Chapter 1 regarding the jQuery event-related methods that are deprecated in jQuery 1.7 and beyond, as well as the recommended jQuery method for handling events.

As you probably know, a tap event occurs whenever users tap on an element, whereas a taphold event occurs when users touch an element and maintain contact for one second. You can bind to a tap event and a taphold event in jQuery Mobile with the following code

```
$("body").bind("tap", function () {
  console.log("Tap Event on the body Element");
  return false;
});

$("body").bind("taphold", function () {
  console.log("Tap Hold Event on the body Element");
  return false;
});
```

The `return false` statement inside a jQuery event handler is effectively the same as invoking `e.preventDefault()` (which prevents the default event from occurring) and also `e.stopPropagation()` (which *does* prevent the event from "bubbling up") on the jQuery `Event` object that is passed as a parameter to the JavaScript function (which is not supplied in this example). Keep in mind that the `return false` statement in non-jQuery event handlers does *not* prevent "bubbling up" behavior on the event that occurred.

A `swipe` event occurs when there is a horizontal drag at the rate of `30px` (or greater) during a one-second interval. The `swipeleft` and `swiperight` events occur when the swipe event is toward the left or toward the right, respectively.

Listing 8.1 displays the contents of `JQMSwipeEvents1.html`, illustrating how to handle tap events as well as swipe left and swipe right events in jQuery Mobile.

LISTING 8.1 JQMSwipeTapEvents1.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
   <meta charset=utf-8" />
   <title>Swipe and Tap Events in jQuery Mobile</title>

  <link rel="stylesheet"
   href="http://code.jquery.com/mobile/1.1.0/
                        jquery.mobile-1.1.0.min.css" />
  <script src="http://code.jquery.com/jquery-2.0.0b1.js">
  </script>
  <script src="http://code.jquery.com/jquery-migrate-1.1.0.js">
  </script>

   <script src="http://code.jquery.com/mobile/1.1.0/
                          jquery.mobile-1.1.0.min.js">
   </script>
```

```
<body>
  <div data-role="page" id="page1">
    <div data-role="header"><h3>Header</h3></div>
    <div data-role="content">
     <img id="sample1" src="Sample1.png" width="150"
                                    height="150" >
     <img id="sample2" src="Sample2.png" width="150"
                                     height="150" >
     <img id="sample3" src="Sample3.png" width="150"
                                     height="150" >
     <div id="result1">Left Image:</div>
     <div id="result2">Middle Image:</div>
     <div id="result3">Right Image:</div>
    </div>
    <div data-role="footer"><h3>Footer</h3></div>
  </div>

  <script>
    <!— jQuery event handling code starts here —>
    $("#page1").live('pageinit', (function(event){
      // handler for tap hold event
      $("#sample1").bind("taphold",function(event, ui){
        console.log("Left image: tap hold event");
        $("#result1").html("Left image: tap hold event");
      })

      // handler for swipe left event
      $("#sample2").bind("swipeleft",function(event, ui){
        console.log("Middle image: swipe left event");
        $("#result2").html("Middle image: swipe left event");
      })

      // handler for swipe right event
      $("#sample3").bind("swiperight",function(event, ui){
        console.log("Right image: swipe right event");
        $("#result3").html("Right image: swipe right event");
```

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

Listing 8.1displays a horizontal row of three images. Under those three images are three text strings that act as “labels” for user gestures that are associated with the left, middle, and right image, respectively.Listing 8.1 also containsconsole.log() messages that are displayed in Chrome Inspector whenever users tap or swipe one of the rendered images. Incidentally, the code inListing 8.1 was deployed as an Android application from inside Eclipse; after switching to the DDMS (Dalvik Debug Monitor Server) perspective, you might see the following type of message in the Eclipse console if you swipe your finger too quickly:

“Miss a drag as we are waiting for Webcore’s response for touch down.”

Figure 8.1 displays the result of rendering JQMSwipeTapEvents1.html inListing 8.1 in the Chrome browser on a MacBook Pro.

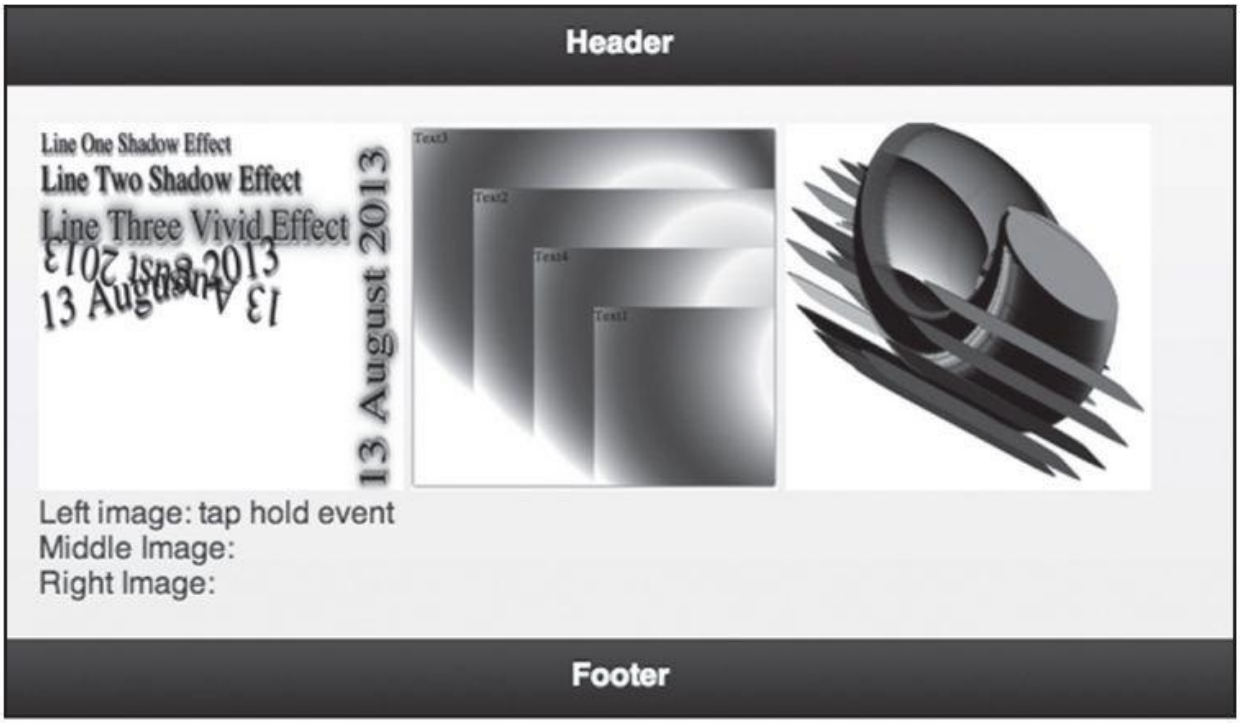


Figure 8.1 Swipe and tap events on the Chrome browser on a MacBook Pro.

Two jQuery Plugins for Detecting User Gestures

ThejGestures jQuery plugin handles a vast set of user gestures, including pinch, rotate, swipe-related and tap-related user gestures. This jQuery plugin is available for download here:

https://jgestures.codeplex.com/

can bind a swipe gesture with the jGestures plugin with the following code snippet:

```
jQuery('#swipe').bind('swipeone',eventHandler);
```

Another jQuery Mobile plugin that handles various user gestures is here:

https://github.com/eightmedia/hammer.js

This JavaScript toolkit has a good collection of samples that illustrate how to handle user gestures. Since there are many jQuery Mobile plugins available for handling user gestures, it's a good idea to perform an Internet search to find many of those plugins, after which you can assess them to determine which ones best fit your needs.

Scroll Events in jQuery Mobile

jQuery Mobile supports the following custom events: orientationchange (triggered by changing the orientation of a device, either vertically or horizontally), scrollstart (triggered when a scroll begins), and scrollstop (triggered when a scroll ends). You can bind to these events like you would with other jQuery events, using live() or bind(), whose usage prior to version 1.7 of jQuery is discussed in Chapter 6. In addition, when your jQuery Mobile code binds to the orientationchange event, the callback function can specify a second argument that contains an orientation property equal to either portrait or landscape.

The HTML5 Web page JQMScrollEvents1.html on the CD illustrates how to handle scrolling events in JQuery Mobile, and its logic looks like this:

```
<script>
 $("#page1").live('pageinit', (function(event,ui){
   var eventsElement = $('#events');
   $(window).bind('scrollstart', function () {
    console.log('Scroll start');
    $('.ui-body-c').css('background', 'green');
    eventsElement.append('<li><a href="">Start
                              Scroll</a></li>');
   //eventsElement.listview('refresh');
    eventsElement.listview();
   });

   $(window).bind('scrollstop', function () {
     console.log('Scroll stop');
     $('.ui-body-c').css('background', 'red');
     eventsElement.append('<li><a href="">Stop
                              Scroll</a></li>');
    //eventsElement.listview('refresh');
     eventsElement.listview();
```

```
    $(window).bind('orientationchange', function () {
      console.log('Orientationchange change');
      $('.ui-body-c').css('background', 'red');
      eventsElement.append('<li><a href="">
                          Orientation</a></li>');
     //eventsElement.listview('refresh');
      eventsElement.listview();
    });
  });
<script>
```

Portrait Mode versus Landscape Mode

In addition to detecting orientation of a mobile device, you will probably need to change the CSS class that is used to style elements in a Web page when there is a change of orientation.

The HTML5 Web page JQMOrientation1.html on the CD illustrates how easily you can change the CSS classes associated with HTML elements based on the orientation of a mobile device. The key logic is contained in this code block:

```
$("#page1").live('pageinit', (function(event,ui){
  $(window).bind ('orientationchange', function (e) {
    console.log("new orientation: "+e.orientation);
    $("#content").removeClass('portrait landscape')
      .addClass (e.orientation ? 'landscape' : 'portrait');
  });
});
```

Notice that the orientationchange event is bound to the window object. In the current example, the background color is set to red and the width is set to 320px in portrait mode, whereas the background color is set to blue and the width is set to 480px in landscape mode.

One other thing to keep in mind is that window.orientation has the value 0 for portrait mode and either -90 or 90 for landscape mode.

Another approach is to use the following type of code in your HTML Web page:

```
@media all and (orientation: portrait) { body {background-color: red} }
<link rel="stylesheet"
    media="all and (orientation: landscape)"
                        href="landscape.css" />
```

Figure 8.2 displays the result of rendering JQMOrientation1.html in a landscape-mode

Figure 8.2 Detecting orientation on an Asus Prime tablet with Android ICS.

Animation Effects Using jQuery Mobile

There are various jQuery-based animation effects that you can create with jQuery Mobile, which includes fading effects, sliding effects, and custom animation effects. The methods for creating animation effects (available both in jQuery and jQuery Mobile) include: animation(), clearQueue() ,delay() ,dequeue() ,fadeIn(), fadeOut(), fadeTo() fadeToggle(),hide(), queue(), show(), slideDown() slideToggle(), slideUp(),stop(), and toggle().

This section contains code samples that illustrate how to create some of these animation effects in jQuery Mobile.

Fade-related Methods

The jQuery .fadeIn() and .fadeOut() methods provide an easy way to create simple animation effects.

The HTML5 Web page JQMFadeInOut1.html on the CD illustrates how to use the jQuery .fadeIn() and .fadeOut() methods in a jQuery Mobile page, and its main logic looks like this:

```
<body>
  <div data-role="page" id="page1">
    <div id="header" data-role="header">
      <h3>Click Header To Show Text</h3>
    </div>
    <div id="context" data-role="content">
```

```
                    width="200" height="200" >
  <img id="sample2" src="Sample2.png"
                    width="200" height="200" >
  <img id="sample3" src="Sample3.png"
                    width="200" height="200" >
  <p>Hello World from jQuery Mobile</p>
  <p>Goodbye World from jQuery Mobile</p>
  <p>Click on this text to toggle this paragraph.</p>
 </div>
 <div data-role="footer"><h3>Footer</h3></div>
</div>

<script>
 $("#page1").live('pageinit', (function(event,ui){
  var imgs = $("img");

  // fade out when users click on the first <img>
  $("img:first").click(function () {
   $("img:first").fadeOut("slow");
  });

  // display first <img> when users click on the last <img>
  $("img:last").click(function () {
   $("img:first").fadeIn("slow");
  });

  // fade when users click on the first <p>
  $("p:first").click(function () {
   $("p:first").fadeOut("slow");
  });

  // display first <p> when users click on the last <p>
  $("p:last").click(function () {
   $("p:first").fadeIn("slow");
  });
 })
);
```

```
</body>
```

The HTML5 Web page JQMFadeInOut1.html contains the jQuery Mobile code for a single page view that responds to click events by storing a reference to the images with this code snippet:

```
var imgs = $("img");
```

If you have an HTML page with many <img> elements, the use of a variable such as imgs can sometimes be more efficient, because jQuery performs a DOM traversal only once. In this code sample, we don't use the imgs variable because there are just three <img> elements, so there is no noticeable performance penalty involved in performing a search each time we reference an <img> element. However, it's important for you to be aware of this coding technique.

The click event on the first image causes the image to slowly fade, as shown here:

```
// fade out when users click on the first <img>
$("img:first").click(function () {
  $("img:first").fadeOut("slow");
});
```

The first image is slowly displayed again whenever users click on the right-most image:

```
// display first <img> when users click on the last <img>
$("img:last").click(function () {
  $("img:first").fadeIn("slow");
});
```

You can use similar code blocks to capture events on other HTML elements. In this example, we could have also captured all the <p>elements with this snippet:

```
var para = $("p");
```

However, in JQMFadeInOut1.html we reference the first and last HTML <p>elements using $("p:first") and $("p:last"), and we then perform similar fade effects.

Figure 8.3 displays the result of rendering JQMFadeInOut1.html in landscape-mode in the Chrome browser on a MacBook.

Figure 8.3 Fade effects on the Chrome browser on a MacBook.

Slide-Related jQuery Methods

As you can surmise, the jQuery .slideUp() and .slideDown()methods provide slide-related functionality for jQuery Mobile.

The HTML5 Web page JQMSlideUpDown1.htmlon the CD illustrates how to use the jQuery .slideUp() and .slideDown() methods in a jQuery Mobile Web page, and a portion of the code is displayed here:

```
<body>
  <div data-role="page" id="page1">
    <div data-role="header"><h3>Header</h3></div>
    <div data-role="content">
      <p>Click me or one of the Rectangles:</p>
      <div name="first" id="first">First</div>
      <div id="second">Second</div>
      <div id="third">Third</div>
      <div id="fourth">Fourth</div>
    </div>
    <div data-role="footer"><h3>Footer</h3></div>
  </div>

 <script>
   $("#page1").live('pageinit', (function(event,ui) {
     $("#page1").click(function() {
      if ($("div:first").is(":hidden")) {
```

```
    } else {
     $("div[name='first']").slideUp(2000);
    }

   $("div[name='first']").css({background:'#00f'})
         .show(2000).hide(2000).slideDown(2000);
  })
 })
)
</script>
</body>
```

The HTML5 Web page JQMSlideUpDown1.html contains a <style> element that uses CSS to apply some styling to four rectangles, followed by a code block that handles a click event anywhere in the page view:

```
$("#page1").click(function() {
 // handle a click event on the page
}
```

The heart of the jQuery Mobile code consists of one line of code that uses method chaining to create a multi-part animation effect that 1) changes the background color to red, 2) performs a "show" and "hide" animation effect for 2 seconds, followed by 3) a "slide" effect that occurs during 2 seconds, whenever users click on a rectangle, as shown here:

```
$("div[name='first']").css({background:'#00f'})
      .show(2000).hide(2000).slideDown(2000);
```

Figure 8.4 displays the result of rendering JQMSlideUpDown1.html in landscape-mode on an Asus Prime tablet with Android ICS.

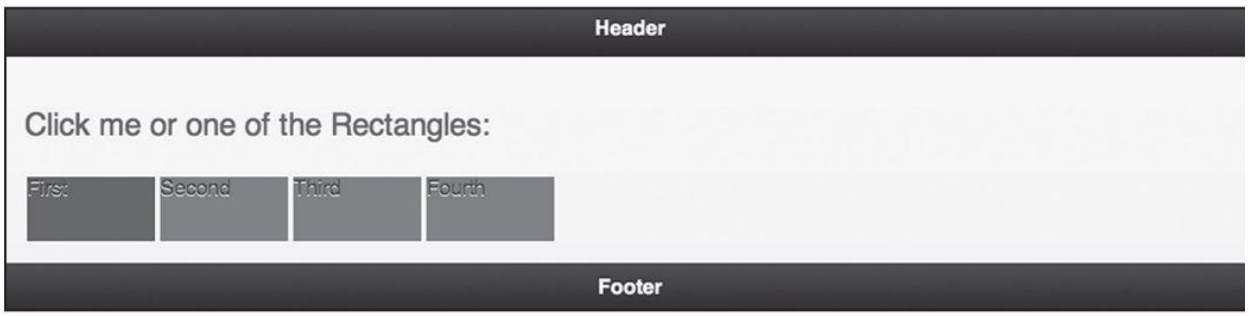


Figure 8.4 Fade effects on an Asus Prime tablet with Android ICS.

jQuery Mobile and Transition Effects

jQuery Mobile uses WebKit -based CSS3 transforms for animating the page transitions, and WebKit -based browsers currently provide the best support for CSS3 transforms (but

hardware acceleration, CSS animation effects appear smooth on mobile devices as well as laptops and desktops.

You specify a transition by applying the data-transitionproperty, whose seven supported values are:

- fade : simply fade the page or dialog in over the previous content
- flip: an animated page flip, rotating the current view out with the other view on the reverse side
- pop : the page springs into view from the center of the screen
- slide: slide in from the left or right, pushing previous content out of the way
- slidedown : slide down from the top, over the top of the current content
- slideup : slide up to the top, revealing the next content below

jQuery Mobile also provides the animationCompleteevent, which you can be useful after adding or removing a class that applies a CSS transition.

Listing 8.2 displays the contents of JQMTransition1.html that illustrates how to useslidedown and flip transitions in a jQuery Mobile page.

LISTING 8.2 JQMTransition1.html

```
<body>
  <div data-role="page" id="first">
    <div data-role="header"> <h2> This is the first page header
    </h2></div>

    <div data-role="content">
     <p>
      <a href="#second" data-transition="slidedown">Go to Page Two</a>
     </p>
    </div>

    <div data-role="footer"><h2>This is the first page footer
    </h2></div>
  </div>

  <div data-role="page" id="second">
    <div data-role="header">
      <h2> This is the second page header</h2>
    </div>

    <div data-role="content">
```

```
    <a href="#first" data-transition="flip"
                data-direction="reverse">
      Go to Page #1 via 'flip' and reverse</a>
    </p>
  </div>

  <div data-role="footer">
    <h2>This is the second page footer</h2>
  </div>
 </div>
</body>
```

Listing 8.2 contains the code for a jQuery Mobile Web page. When you click on the link in the first page view, you will see a slide effect created by the following code snippet in the content HTML of the first page view (whose id attribute has value first):

```
<a href="#second" data-transition="slidedown">Go to Page Two</a>
```

The second page view displays the following text and hyperlink:

You saw a slidedown effect Go to Page #1 via 'flip' and reverse

When you click on the preceding link, you will see a flip effect that is created with the following code snippet in the content HTML of the second page view (whose id attribute has value second):

```
<a href="#first" data-transition="flip"
            data-direction="reverse">
```

Note that jQuery Mobile will attempt to use the reverse transition when using the automatic back button or when hiding a dialog. This simple example illustrates some page transition effects that you can create when you navigate between page views. You can experiment with a number of different transition effects to find the ones that suit your requirements.

Figure 8.5 displays the result of rendering JQMTransition1.html in Listing 8.2 in landscape mode in the Chrome browser on a MacBook.

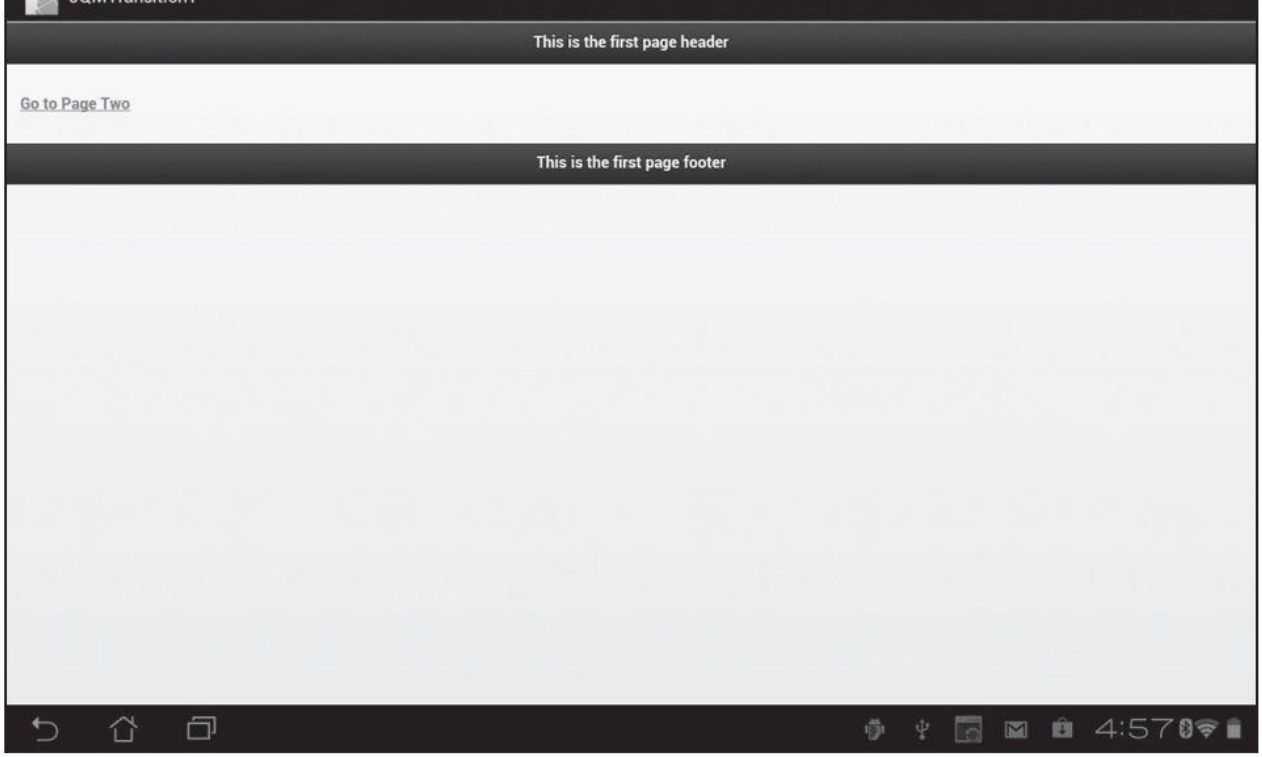


Figure 8.5 Transition effects on the Chrome browser on a MacBook.

jQuery Mobile and Animation Effects with CSS3

Recall that jQuery Mobile uses WebKit -based CSS3 transforms for animating the page transitions. WebKit browsers use hardware acceleration, so that CSS animation effects appear smooth on mobile devices, laptops, and desktops.

Listing 8.3 displays the contents of JQMJPG1.html that illustrates how to use slideDown() and slideUp() in conjunction with JPG files in a jQuery Mobile page.

LISTING 8.3 JQMRenderJPG1.html

```
<body>
   <div data-role="page" id="page1">
     <div data-role="content">
      <img id="sample1" src="Sample1.png" width="200"
                                height="200" >
      <img id="sample2" src="Sample2.png" width="200"
                                height="200" >
      <p>
        <a id="button1" href="#" data-role="button">
          Click Me To Display The Right Image</a>
        <a id="button2" href="#" data-role="button">
          Click Me To Shrink The Right Image</a>
      </p>
```

```
    </div>

    <script>
      $("#page1").live('pageinit', (function(event,ui) {
        $("#sample2").hover(function() {
          $(this).slideUp(800);
        });

        $("#sample2").hover(function() {
          $(this).css({'position':'relative',
                       'width':'200px',
                       'height':'200px'
                       });
        });

        $("#button1").click(function() {
          $("#sample2").slideDown(800);
        });

        $("#button2").click(function() {
          $("#sample2").css({'position':'relative',
                             'width':'100px',
                             'height':'100px'
                            });
        });
      })
     )
    </script>
   });
  </body>
```

Listing 8.3 renders two images and contains two buttons that bind to click events. The visual effects depend on the sequence in which the buttons are clicked. When users click on the second button ("Click Me To Shrink The Right Image"), the right-most image is reduced from its initial dimensions of 200x200 to dimensions of 100x100 using this code block:

```
$("#button2").click(function() {
  $("#sample2").css({'position':'relative',
                     'width':'100px',
```

```
        });
});
```

As you can see, the preceding code block uses the jQuery `css()` function to modify the width and height of the right-most image, which means there is no transition effect.

Next, if users hover over the right-most image (which is now reduced in size), the image will shrink and disappear with a transitional effect because of the following code block:

```
$(“#sample2”).hover(function() {
  $(this).slideUp(800);
});
```

Now if users click on the first button (“Click Me To Display The Right Image”), the second image will reappear (again using a transitional effect) with its original dimensions because of the following code:

```
$(“#button1”).click(function() {
  $(“#sample2”).slideDown(800);
});
```

This code samples illustrates how easy it is to write jQuery code for creating pleasant animation effects with images. Moreover, you can combine the jQuery animation effects with the jQuery `css()` function. For example, you can use the `css()` function to render an image with a background radial gradient effect when users click on the image, or update any other CSS properties of the image (or element) in question.

Thus, you can combine your knowledge of jQuery animation, CSS, and CSS3 graphics and animation to create interesting effects by means of simple and compact jQuery code.

Figure 8.6 Resizing JPGs with animation effects on the Chrome browser on a MacBook.

Figure 8.6 displays the result of rendering JQMRenderJPG1.html in Listing 8.3 in landscape mode on the Chrome browser on a MacBook.

jQuery Mobile Virtual Mouse Events

jQuery Mobile provides a set of "virtual" mouse events for handling mouse and touch events. These events are useful because each one handles a mouse event as well as its corresponding touch-based event, which can reduce the amount of code that you need to write in your Web pages. In addition, the use of virtual mouse events can support both the desktop and mobile style interactions in the same Web page.

The name of a virtual mouse event in jQuery Mobile starts with the letter "v" (for "virtual") followed by the common or standard name for a mouse event. For example, the vmousedown virtual event "delivers" the mousedown event and also the touchstart event. The list of virtual mouse events (and their corresponding touch-related and mouse-related events) is shown here:

vclick (touchend or mouse click events)

vmousecancel (touch or mouse mousecancel events)

vmousedown (touchstart or mousedown events)

vmousemove (touchmove or mousemove events)

vmouseover (touch or mouseover events)

vmouseup (touchend or mouseup events)

Keep the following in mind: on touch-enabled devices, the vclick event is dispatched after the vmouseup event; however, vmouseup is dispatched before vmousedown, and vmousedown is dispatched before vclick, as you would expect. Furthermore, the event object contains the properties pageX, pageY, screenX, screenY, clientX, and clientY that contain coordinate information, as you will see in a subsequent code sample.

The HTML5 Web page JQM2DAnimationRGrad4Reflect1.html and the CSS stylesheet JQM2DAnimationRGrad4Reflect1.css on the CD illustrate how to create CSS3 2D animation effects in jQuery Mobile.

LISTING 8.4 JQM2DAnimationRGrad4Reflect1.html

```
<body>
  <div  data-role="page">
   <div data-role="content">
    <div id="outer">
     <div id="radial3">Text3</div>
     <div id="radial2">Text2</div>
     <div id="radial4">Text4</div>
     <div id="radial1">Text1</div>
    </div>
   </div>
  </div>
</body>
```

Listing 8.4 is straightforward: the jQuery Mobile code contains four HTML elements that have corresponding CSS3 selectors in the associated CSS stylesheet. The four CSS3 selectors define radial gradients (defined in the accompanying CSS stylesheet on the CD), which are very similar to CSS3 code samples in Chapter 2. (Read the appropriate section if you need to refresh your memory.) The key point to notice is that even though the visual effects in this code sample are not new, this code demonstrates you how easily you can combine jQuery Mobile code with CSS3 2D/3D graphics and animation effects.

memory.) The key point to notice

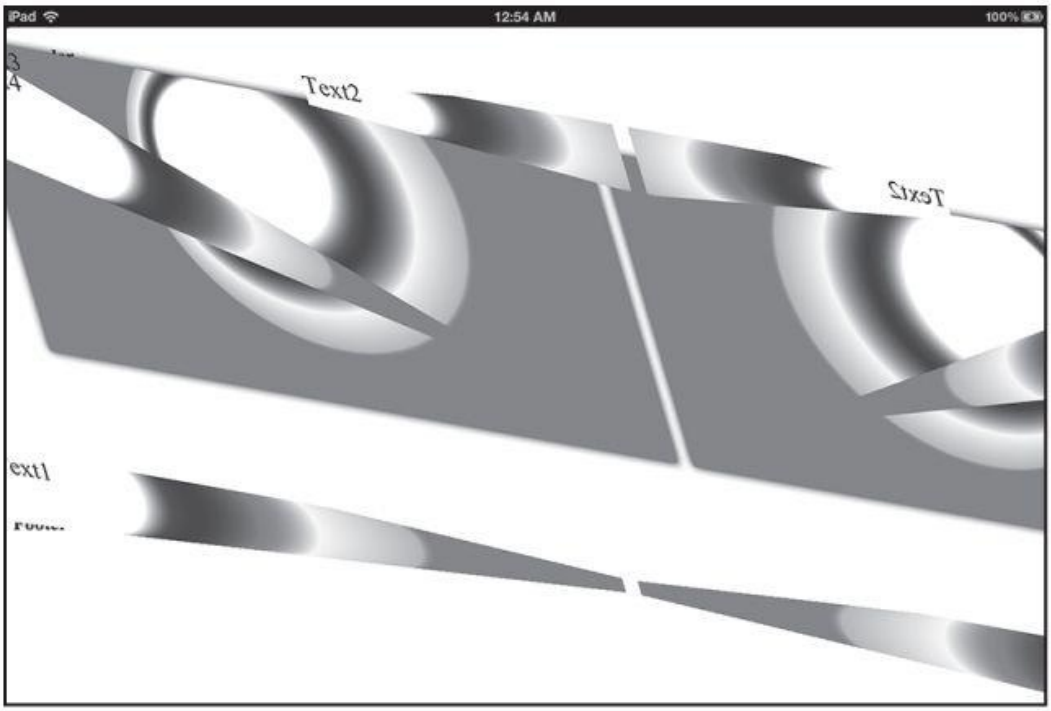


Figure 8.7 CSS3 2D effects on an iPad3.

Figure 8.7 renders JQM2DAnimationRGrad4Reflect1.html in landscape mode on an iPad3.

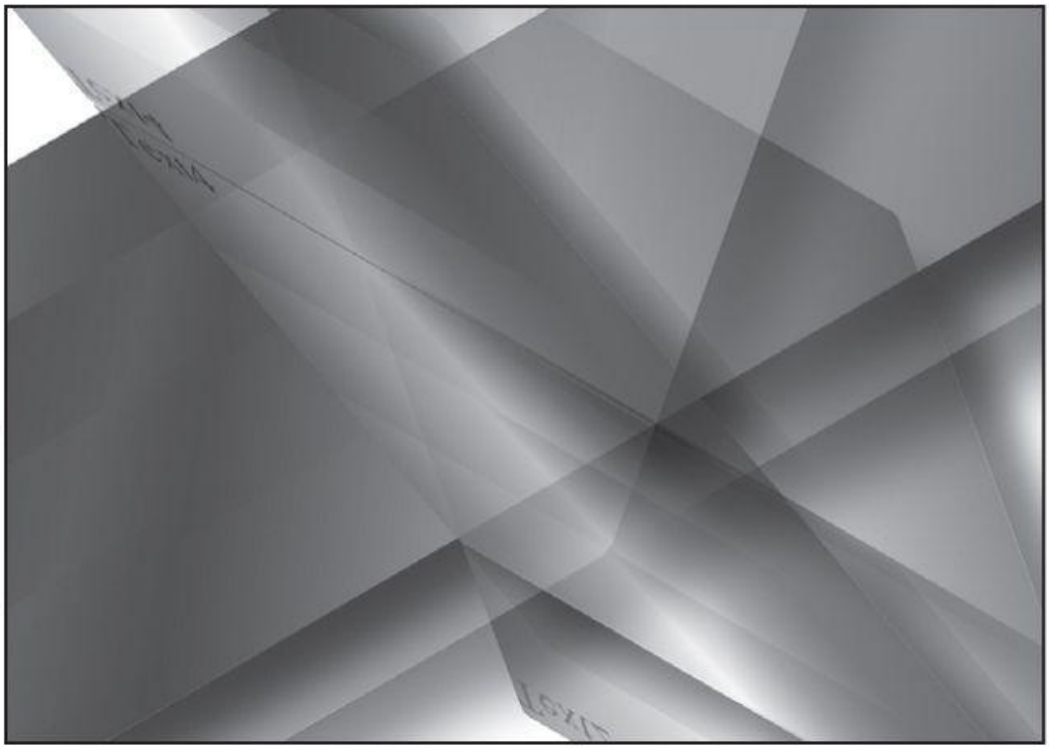


Figure 8.8 CSS3 3D effects on an Asus Prime tablet with Android ICS.

creates CSS3 3D animation effects. Figure 8.8 displays JQM3DAnimRotate3DLGrad2SkewOpacityRep4Reflect1.html in landscape mode on an Asus Prime tablet with Android ICS.

The HTML5 Web page JQMCSS3AnimTap.html on the CD shows you how to detect user tap events and then create CSS3 3D animation effects. Figure 8.9 displays the result of launching this Web page in a browser, in landscape mode, on an Asus Prime tablet with Android ICS.

Figure 8.9 CSS3 3D Tap animation on an Asus Prime tablet with Android ICS.

Earlier in this chapter, you learned about jQuery Mobile virtual mouse events. The Web page JQMSketchSolidDynamicDOM1.html on the CD illustrates how to capture user gestures to dynamically create and append new HTML elements into the HTML element whose id attribute has value content in order to create a "sketching" program.

LISTING 8.5 JQMSketchSolidDynamicDOM1.html

```
<body>
  <script>
   $("#page1").live('pageinit', (function(event,ui){
     var insertNode = false;
     var newNode;

     // mouse-down means insertNode:
     $("#content").bind('vmousedown', function() {
       console.log("start");
       insertNode = true;
     });
```

```
    // mouse-up means no insertNode:
    $("#content").bind('vmouseup', function() {
      console.log("stop");
      insertNode = false;
    });

    $("#content").bind('vmousemove', function(e) {
       // are users are moving their mouse?
       if(insertNode == true) {
          console.log("move");

          // create a rectangle at the current position
          newNode = $('<div>').css({'position':'absolute',
                                    'background-color':'#ff0000',
                                    'width':'8px',
                                    'height':'8px',
                                     top: e.pageY,
                                     left: e.pageX
                                   });

          //append the rectangle to the content <div>
          $("#content").append(newNode);
       }
    });
  })
 );
</script>
</body>
```

Listing 8.5 contains jQuery Mobile code that binds to the vmousedown, vmouseup, and vmousemove virtual events. The code uses the first two virtual mouse events to determine if a vmousemove event is accompanied with a vmousedown event; if the latter is true, then a new HTML element is programmatically created at the current location of a user's mouse and then inserted into the DOM.

Figure 8.10 displays JQMSketchSolidDynamicDOM1.html in landscape mode in the Chrome browser on a MacBook.

Figure 8.10 Sketching effects in the Chrome browser on a MacBook.

In an earlier chapter, you saw how to create an HTML5 Web page using CSS3 to create a 3D cube whose faces move when users hover their mouse over any of the three faces of the cube. The HTML5 Web page JQM3DCube1.html on the CD illustrates how to render a 3D cube with CSS3.

LISTING 8.6 JQM3DCube1.html

```
<body>
   <div data-role="page" id="page1">
    <div data-role="header">
      <h3>Tap on the Cube Faces:</h3>
    </div>
    <div data-role="content">
      <div id="outer">
       <div id="top">Text1</div>
       <div id="left">Text2</div>
       <div id="right">Text3</div>
      </div>
    </div>
    <div data-role="footer"><h3>Footer</h3></div>
   </div>

   <script>
    $("#page1").live('pageinit', (function(event,ui){ }));
   </script>
</body>
```

 elements serving as placeholders for the left, top, and right faces of the cube, and a tiny jQuery Mobile code snippet, as shown here:

```
<script>
  $("#page1").live('pageinit', (function(event,ui){ }));
</script>
```

All the real "action" in this code sample takes place in the CSS3 stylesheet JQM3DCube1.css, a portion of which is shown in Listing 8.7 .

LISTING 8.7 JQM3DCube1.css

```
/* animation effects */
#right:hover {
-webkit-transition: -webkit-transform 3.0s ease;
transition: transform 3.0s ease;

-webkit-transform : scale(1.2) skew(-10deg, -30deg) rotate(-45deg);
transform : scale(1.2) skew(-10deg, -30deg) rotate(-45deg);
}

#left:hover {
-webkit-transition: -webkit-transform 2.0s ease;
transition: transform 2.0s ease;

-webkit-transform : scale(0.8) skew(-10deg, -30deg)
                    rotate(-45deg);
transform : scale(0.8) skew(-10deg, -30deg) rotate(-45deg);
}

#top:hover {
-webkit-transition: -webkit-transform 2.0s ease;
transition: transform 2.0s ease;

-webkit-transform : scale(0.5) skew(-20deg, -30deg)
                    rotate(45deg);
transform : scale(0.5) skew(-20deg, -30deg) rotate(45deg);
}
// details omitted for brevity
```

The CSS3 selectors in Listing 8.7 are taken from a code sample in Chapter 2 that also illustrates how to render a cube using CSS3 selectors, and you can review the code details

effects whenever users tap or hover over one of the faces of the 3D cube.

Figure 8.11displays JQM3DCube1.htmin landscape mode on an Asus Prime tablet with Android ICS.

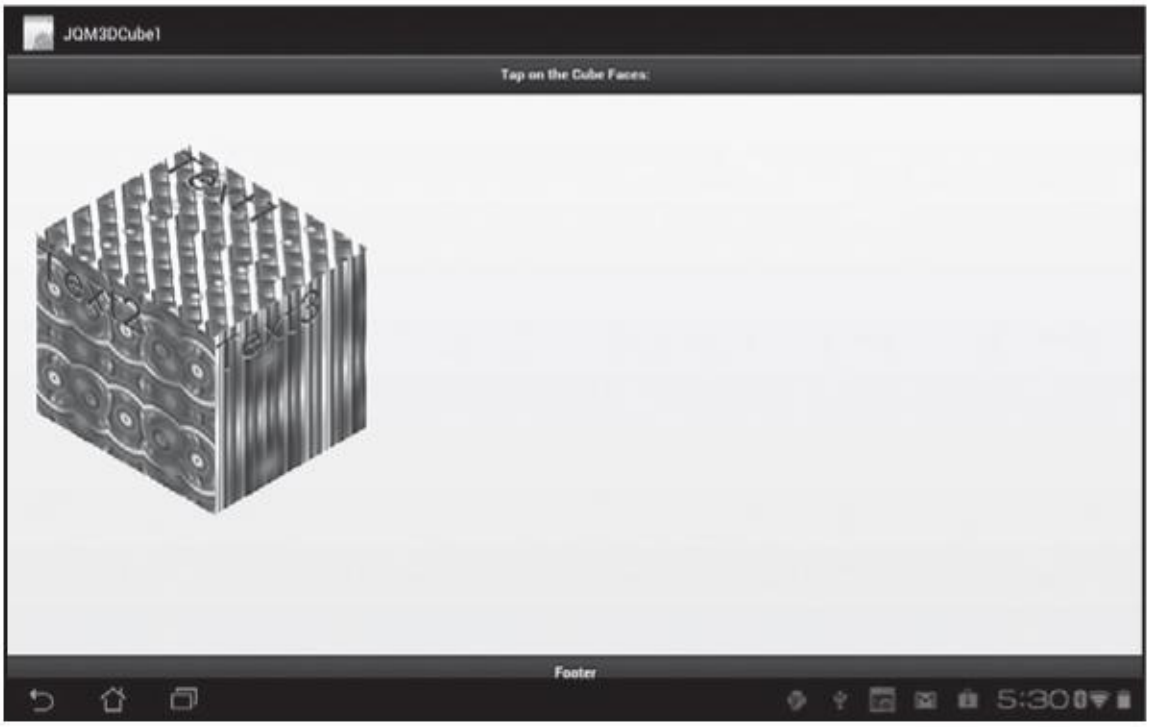


Figure 8.11 jQuery Mobile and CSS3 on an Asus Prime tablet with Android ICS.

Additional Code Samples on the CD

The CD contains several other code samples with similar techniques using jQuery Mobile. You can use these techniques as "building blocks," perhaps also with some of your own, in order to create more complex visual effects.

ON THE CD The code samples are primarily for fun, and you can experiment with them to create other interesting effects by modifying additional CSS properties dynamically.

You can create page-turning effects in jQuery Mobile applications in various ways, and in this section you will see how to use the jQuery plugin Turn.js whose homepage is here:

http://www.turnjs.com

This code sample contains many examples of handling jQuery Mobile virtual events as well as user gestures and key events. Although space precludes us from discussing every detail of this code sample, it's well worth your time to read the code in detail to learn some useful techniques.

ON THE CD The HTML5 Web pageJQMPageTurn1.html on the CD illustrates how to use Turn.js in order to simulate page-turning effects in a jQuery Mobile application.

The HTML5 Web page JQMAccelerometer1.html on the CD illustrates how to display the accelerometer values of a mobile device whenever the device undergoes acceleration in any direction.

Summary

This chapter introduced you to animation effects in jQuery Mobile, and you saw code samples that illustrated how to perform various effects. You learned how to do the following in jQuery Mobile:

- Tap events and swipe events
- Scroll events
- Detect portrait versus landscape mode
- Fade-related animation effects
- Slide-related animation effects
- Transitions
- Animation effects with CSS3
- Virtual mouse events

INTRODUCTION TO HTML5 CANVAS

This chapter provides an overview of HTML5 Canvas, which is a technology that enables you to write graphics programs that draw directly to a part of a Web page. HTML5 Canvas supports various APIs for rendering 2D shapes with an assortment of graphics effects. Although there are many online Canvas-related and CSS3-related tutorials available (which you can confirm via a quick Internet search), few of them provide code examples of using both HTML5 Canvas and CSS3 graphics effects.

As you will see, various code samples in this chapter contain (sometimes striking) combinations of HTML5 Canvas, CSS3 graphics, and CSS3 2D/3D animation effects that you are unlikely to find in any online resources or topic-related books. These code samples provide a starting point for you to create your own visually compelling graphics effects.

In addition, most of the sections in this chapter start with the syntax of the APIs that are used in the associated code listings, partly because the code samples contain a lot of details and also illustrate multiple concepts. So, even though this is an "introductory" chapter about HTML5 Canvas, you will learn considerably more than you would expect from a basic overview that you might find in other books.

The first part of this chapter shows you how to render line segments, rectangles, and circles in HTML5 Canvas , and also how to combine HTML5 Canvas with CSS3 stylesheets.

The second part introduces you to linear and radial gradients in HTML5 Canvas , with examples of how to apply them to Bézier curves and JPG files. The third part of this chapter discusses jCanvas, which is a jQuery plugin for HTML5 Canvas, and also an example of combining Canvas -based graphics with jQuery Mobile.

The concepts and code samples in this chapter will help you understand the HTML5 Canvas -based charts and graphs. If you want to explore additional HTML5 Canvas graphics after you have finished reading this chapter, an extensive set of code samples is available here:

http://code.google.com/p/html5-canvas-graphics

This chapter provides techniques for creating various visual effects that you can use in your custom charts and graphs. At the same time, it's also important for you to assess the trade-off (time, effort, and cost) between writing low-level Canvas-based graphics code, such as the code samples in this chapter, versus the availability of open source projects and commercial products.

One more point to keep in mind: HTML5 Canvas does not have any sort of "dependency" on jQuery or jQuery Mobile. If necessary, you can create hybrid HTML5

PhoneGap) or even without a toolkit.

What is HTML5 Canvas?

Several years ago Canvas began in OS X as a widget toolkit. After Canvas had already been available in the Safari browser, it became a specification for the Web. Now, it is commonly referred to as HTML5 Canvas .

HTML5 Canvas and SVG both allow you to programmatically render graphics in a browser via JavaScript. However, HTML5 Canvas uses “immediate mode,” which is a write-and-forget approach to rendering graphics. Thus, if you want to write a sketching program in HTML5 Canvas and you also want to provide an “undo” feature, then you must programmatically keep track of everything that users have drawn on the screen. On the other hand, SVG uses a “retained mode,” which involves a DOM (Document Object Model) structure that keeps track of the rendered objects and their relationship to one another.

If you are going to write HTML Web pages that make extensive use of graphics effects, you’ll probably need to understand the differences between HTML5 Canvas and SVG in terms of performance. You have the freedom to use one technology exclusively, but you can also create HTML Web pages that contain a mixture of HTML5 Canvas , SVG, and CSS3. Performance-related information can help you decide how you are going to code your HTML Web pages.

Although this chapter does not delve into the preceding points in any more detail, you can find a good overview of some features/advantages of HTML5 Canvas here:

http://thinkvitamin.com/code/how-to-draw-with-html-5-canvas/

Incidentally, if you need HTML5 Canvas support in Internet Explorer 8, you can use ExplorerCanvas , which is an open source project that is available here:

http://code.google.com/p/explorercanvas/

You can use the preceding code project simply by including the following code snippet in your HTML Web pages:

```
<!— [if IE lt 8]><script src=“excanvas.js”></script><![endif]—>
```

HTML5 **Canvas** versus SVG

One point to consider is when it’s advantageous to use HTML5 Canvas instead of a technology such as SVG. The following short list contains some features to consider when you are making this type of analysis:

- Native versus plug-in browser support
- Level of SVG support in different browsers
- Animation support
- Support for filters (SVG only)
- Built-in support for HTML-like widgets
- Third-party support

Adobe's SVG viewer can be used with Microsoft's Internet Explorer. If you need filter-based visual effects, then SVG provides a very rich (perhaps even the best) functionality. If you need built-in support for HTML controls, then frameworks such as Wijmo might be a good solution for your needs. Another point to consider is that Adobe no longer supports its SVG viewer. This is a significant decision, because Adobe's SVG viewer had been the *de facto* standard for SVG viewers for many years. Although Firefox and Opera have made significant progress in terms of their support for SVG, and both are enhancing their support for SVG, they still lack the feature support of Adobe's SVG viewer. Thus, you need to weigh the most important factors in order to make the decision that will meet your project-related needs.

A very good article containing examples and diagrams that compares the use of HTML5 Canvas and SVG is here:

http://blogs.msdn.com/b/ie/archive/2011/04/22/thoughts-on-when-to-use-canvas-and-svg.aspx

The HTML5 Canvas Coordinate System

Think back to your days in high school, where you learned that the Cartesian coordinate system identifies any point in the Euclidean plane by means of a pair of numbers, often written as (x,y). The first number represents the horizontal value and the second number represents the vertical value. The horizontal axis is labeled the x-axis, and positive values on the x-axis are to the right of the vertical axis (i.e., toward the right). The vertical axis is labeled the y-axis, and positive values on the y-axis are above the horizontal axis. The origin is the intersection point of the x-axis and the y-axis.

The situation is almost the same in the HTML5 Canvas coordinate system. The x-axis is horizontal and the positive direction is toward the right. The y-axis is vertical, but the positive direction is *downward*, which is the opposite direction of most graphs in a typical mathematics textbook. In the HTML5 Canvas coordinate system, the origin is the upper-left corner of the screen (not the lower-left corner), and the unit of measurement is the pixel. As this book goes to print, the largest visible display is 2880×1800, and undoubtedly larger displays will be available in the future.

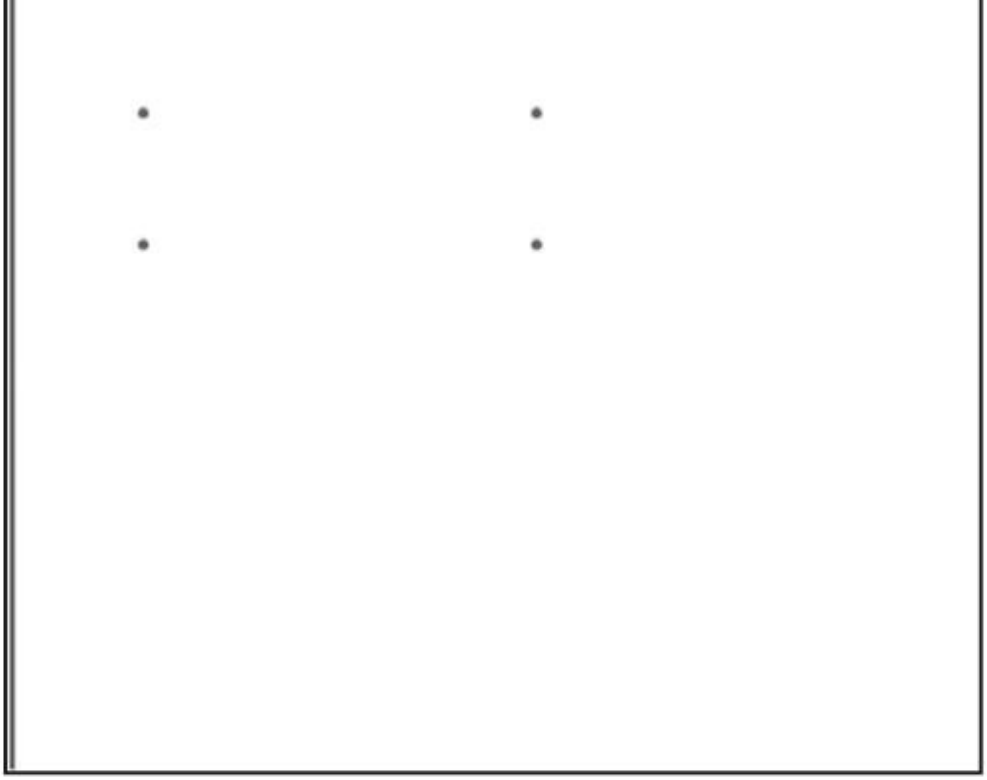


Figure 9.1 Four points rendered in HTML5 Canvas .

As a simple illustration, Figure 9.1 displays four points in an HTML5 <canvas> element.

If you start from the origin (the upper-left corner of the screen) and move 50 pixels to the right, followed by 50 pixels downward, you will reach the upper-left point in Figure 9.1 . Next, if you start from the origin and move 200 pixels to the right and 50 pixels downward, you will reach the upper-right point in Listing 9.1 In a similar fashion, the two points in the second "row" have coordinates (50,100) and (200,100). Notice that the two points in the first row have the same value for the y-coordinate, which makes sense because they are the same distance away from the top of the Web page; the same is true for the two points in the second row. Similarly, the two points in the left "column" have the same x-coordinate because they are both the same distance from the left side of the Web page.

Now that you have an understanding of the HTML5 Canvas coordinate system, let's take a look at the contents of Listing 9.1 which displays a minimal HTML5 Web page that is ready for rendering HTML5 Canvas -based graphics. Every Canvas -based code sample in this book uses the code (or some variant) that is displayed in Listing 9.1 . Note that if you launch this code in a browser session, you will only see a blank screen.

LISTING 9.1 Canvas1.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
  <title>Canvas Drawing Rectangles</title>
```

```
    window.addEventListener('load', function () {
      // Get the canvas element
      var elem = document.getElementById('myCanvas');
      if (!elem || !elem.getContext) {
        return;
      }

      // Get the canvas 2d context
      var context = elem.getContext('2d');
      if (!context) {
        return;
      }

     // Insert your custom Canvas graphics code here
      });
    // --></script>
  </head>

  <body>
    <canvas id="myCanvas" width="300" height="300">
                                No support for Canvas.
    </canvas>
  </body>
</html>
```

Listing 9.1 contains an HTML <head> element that checks for the existence of an HTML <canvas> element inside the HTML <body> element of the Web page, and then gets the 2D context from the HTML <canvas> element. If you skip over the various conditional statements in Listing 9.1, there are two lines of code that enable us to get a reference to the variable context, which represents a drawable surface:

```
var elem = document.getElementById('myCanvas');
var context = elem.getContext('2d');
```

If you launch Listing 9.1 in a browser that does not support HTML5 Canvas, the text message "No support for Canvas." is displayed.

The following code snippet is executed whenever you launch the Web page because of an anonymous JavaScript function that is executed during the load event:

```
<script><!--
window.addEventListener('load', function () {
  // do something here
```

```
// —></script>
```

Now that you understand the underlying code for rendering Canvas-based 2D shapes, you can focus on the code that actually draws some 2D shapes, starting with the example in the next section.

Line Segments, Rectangles, Circles, and Shadow Effects

This section contains an assortment of code samples that illustrate how to render 2D shapes in HTML5 Canvas. There are many concepts introduced in this section, so before delving into the code sample, let's look at some of the HTML5 Canvas APIs that are used in this section. Chapter 2 contains a section that describes various ways for specifying colors, and the material in that section is relevant for the code sample in this chapter (so you can quickly review its contents now if you need to do so).

HTML5 Canvas provides the fillRect() method for rendering a rectangle, which requires four parameters: the upper-left vertex (defined by its x-coordinate and its y-coordinate) of the rectangle, the width of the rectangle, and the height of the desired rectangle. The Canvas fillRect() API looks like this:

```
context.fillRect(x, y, width, height);
```

HTML5 Canvas allows you to render line segments by specifying the (x,y) coordinates of the two endpoints of a line segment. The two new APIs that are used in the code sample in this section are moveTo() and lineTo(), and they look like this:

```
context.moveTo(x1, y1);

context.lineTo(x2, y2);
```

The preceding code snippet represents the line segment whose two endpoints are specified by the points (x1, y1) and (x2, y2). Note that you can also render the same line segment with the following code snippet:

```
context.moveTo(x2, y2);

context.lineTo(x1, y1);
```

Shadow effects provide a richer visual experience that is an improvement over the use of non-shadow effects. You create a shadow effect by assigning values to three shadow-related attributes that control the size of the underlying shadow and also the extent of the "fuzziness" of the shadow, as shown here:

```
context.shadowOffsetX = shadowX;

context.shadowOffsetY = shadowY;

context.shadowBlur    = 4;
```

You can also assign (R,G,B) or (R,G,B,A) values to shadowColor (which is an attribute o the drawing context) as shown here:

```
context.shadowColor   = "rgba(0,0,64,1.0)";
```

The HTML5 Web page RandRectanglesShadow.html in Listing 9.2 (with code sections omitted for brevity) uses this technique in order to render a set of randomly generated rectangles with a shadow effect.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Canvas Random Rectangles With Shadow Effects</title>
  <link href="CSS3Background2.css"
        rel="stylesheet" type="text/css">

  <script><!—
   window.addEventListener('load', function() {
    redrawCanvas = function() {
      // clear the canvas before drawing new set of rectangles
      context.clearRect(0, 0, elem.width, elem.height);

      for(var r=0; r<rectCount; r++) {
        basePointX = canWidth*Math.random();
        basePointY = canHeight*Math.random();

        // Alternate shadow effect based on an even/odd
        // click count with different (R,G,B,A) values
        if(clickCount % 2 == 0) {
          context.shadowColor   = "rgba(0,0,64,1.0)";
        } else {
          context.shadowColor   = "rgba(64,0,0,1.0)";
        }

        // code that specifies the size and also the
        // "fuzziness" of the underlying shadow effect
        context.shadowOffsetX = shadowX;
        context.shadowOffsetY = shadowY;
        context.shadowBlur    = 4;
        context.lineWidth     = 1;

        // render a colored rectangle
        colorIndex = Math.floor(basePointX)%fillStyles.
                     length;
        context.fillStyle = fillStyles[colorIndex];
```

```
            context.fillRect(basePointX, basePointY,
                        rectWidth, rectHeight);


            ++clickCount;
          }
        }


        // render a set of random rectangles
        redrawCanvas();
      });
      // —></script>
    </head>


    <body>
      <canvas id="myCanvas" width="800" height="350">No support for
       Canvas
      </canvas>
    </body>
    </html>
```

The HTML5 code in Listing 9.2 starts by initializing some JavaScript variables and then defining the JavaScript function `redrawCanvas()` that contains a loop for rendering the rectangles on the screen. The loop calculates the coordinates of the upper-left vertex of each rectangle as shown here:

```
basePointX = canWidth*Math.random();
basePointY = canHeight*Math.random();
```

The next part of the loop assigns the background color (which alternates between a dark blue and dark red shadow), and then sets up the shadow effect by specifying values for the attributes `shadowOffsetX`, `shadowOffsetY`, and `shadowBlur`, as shown here:

```
context.shadowOffsetX = shadowX;
context.shadowOffsetY = shadowY;
context.shadowBlur    = 4;
```

The actual rendering of each rectangle is performed by the following code:

```
context.fillRect(basePointX, basePointY,
            rectWidth, rectHeight);
```

Notice that the `clickCount` variable is incremented each time users click inside the HTML5 `Canvas` element, and its value determines which shadow color is applied to the randomly generated rectangles.

performance. If you need shadow-like effects but performance becomes an issue, one alternative is to render a background shape in black (or some other dark color), and then rendering the same shape (with a small offset) using a different color.

For example, you can create a shadow effect for rectangles by first rendering a black rectangle and then rendering a red rectangle on top of the black rectangle, as shown here:

```
// render a black rectangle
context.fillStyle = '#000';
context.fillRect(50+shadowX, 50+shadowY, 200, 100);

// render a red rectangle
context.fillStyle = '#f00';
context.fillRect(50, 50, 200, 100);
```

The values for `shadowX` and `shadowY` determine the size of the background "shadow," and the choice of positive versus negative values for `shadowX` and `shadowY` will determine the relative position of the black rectangle with respect to the red rectangle.

The CSS stylesheet `CSS3Background2.css` that is referenced in the HTML5 Web page `RandRectanglesShadow1.html` is available on the CD. This CSS stylesheet contains two similar CSS3 selectors for rendering the HTML5 `<canvas>` element defined in Listing 9.2, as well as a hover-based selector that changes the background of the HTML5 `<canvas>` element whenever users hover over this element with their mouse. The `#myCanvas` selector defines a radial gradient, followed by two repeating radial gradients that specify various combinations of red, green, yellow, and blue at different pixel locations. A key point involves the use of `transparent`, which changes the gap between consecutive colors that are rendered.

Figure 9.2 Canvas random rectangles on an Asus Prime tablet with Android ICS.

Figure 9.2 displays a set of randomly generated rectangles with a shadow effect based on RandRectanglesShadow.html in Listing 9.2, rendered in landscape mode on an Asus Prime Tablet with Android ICS.

HTML5 Canvas Linear Gradients

HTML5 Canvas provides two primary types of color gradients (similar to SVG and CSS3): *linear gradients* and *radial gradients*.

Linear color gradients can be further sub-divided into three types: horizontal linear gradients, vertical linear gradients, and diagonal linear gradients. Thus, HTML5 Canvas provides color gradients that enable you to create pleasing visual effects.

A linear gradient is defined in terms of ColorStop elements, each of which contains a decimal (between 0 and 1) and a hexadecimal value that represents a color. For example, i you define a linear gradient with an initial color of #FF0000 (the hexadecimal value for red) and a final color of #000000 (the hexadecimal value for black), then the resultant color gradient will range (in a linear fashion) from red to black. Linear gradients enable you to create vivid and interesting color combinations, and they are available in three varieties: horizontal, vertical, and diagonal. Note that "linear gradient" and "linear color gradient" are used interchangeably in this book.

Horizontal, Vertical, and Diagonal Linear Gradients

As you learned in the introduction of this chapter, HTML5 Canvas supports the method createLinearGradient() that you can use to programmatically create linear gradients. Its syntax looks like this:

```
context.createLinearGradient(startX, startY, endX, endY);
```

The HTML5 page LGradRectangles1.html in Listing 9.4 demonstrates how to render a set of rectangles with horizontal, vertical, and diagonal linear gradients. Listing 9.3 references the CSS3 stylesheet HoverAnimation1.css that applies CSS3 keyframes-based 2D animation to the first HTML5 <canvas> element whenever users hover over this <canvas> element with their mouse. Listing 9.4 also references the CSS3 stylesheet HoverAnimation2.css, which acts in a similar fashion. However, this stylesheet applies CSS3 3D animation effects to the second HTML5 <canvas> element in Listing 9.3. Since the animation techniques in these CSS stylesheets were discussed in Chapter 2, we will omit them from this chapter, but the entire source code is available on the CD.

LISTING 9.3 LGradRectangles1.html

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
  <title>Canvas Linear Gradient Rectangles</title>
  <link href="HoverAnimation1.css"
       rel="stylesheet" type="text/css">
  <link href="HoverAnimation2.css"
       rel="stylesheet" type="text/css">

  <script><!—
   window.addEventListener('load', function () {
    redrawCanvas = function() {
      // clear the canvas before drawing new set of rectangles
      //context.clearRect(0, 0, elem.width, elem.height);
      //context2.clearRect(0, 0, elem.width, elem.height);

      // upper left rectangle: horizontal linear gradient
      currentX = basePointX;
      currentY = basePointY;

      gradient1 = context.createLinearGradient(
                          currentX,
                          currentY,
                          currentX+rectWidth,
                          currentY+0*rectHeight);

      gradient1.addColorStop(0, '#f00');
```

```
        context.fillStyle = gradient1;
        context.fillRect(currentX, currentY,
                         rectWidth, rectHeight);

        // upper right rectangle: vertical linear gradient
        // similar code omitted for brevity

        // render the lower rectangles in the second <canvas>
        // element
        // lower left rectangle: diagonal linear gradient
        // similar code omitted for brevity

        // lower right rectangle: diagonal linear gradient
        // similar code omitted for brevity

        ++clickCount;
        basePointX += 4;
        basePointY += 2;
      }

      // render linear gradient rectangles
      redrawCanvas();
    }, false);
    // —></script>
  </head>

<body>
    <div>
     <canvas id=“myCanvas” width=“600” height=“250”>
      No support for Canvas
           alt=“Rendering linear gradient rectangles.”>
     </canvas>
    </div>

    <div>
     <canvas id=“myCanvas2” width=“600” height=“250”>No support
      for Canvas
```

```
    </canvas>
  </div>

  <div>
   <input type="button" onclick="redrawCanvas();return false"
        value="Redraw the Rectangles" />
  </div>
 </body>
</html>
```

Listing 9.3 renders four rectangles with linear gradient shading. The linear gradients have two, three, or four invocations of the `addColorStop()` method, using various combinations of colors (expressed in hexadecimal form) so that you can see some of the gradient effects that are possible.

Experiment with different values for the color stop definitions to see how their values change the appearance of the rendered rectangles.

Figure 9.3 Linear gradient rectangles on a Nexus 7 tablet with Android JellyBean.

Figure 9.3 displays a set of randomly generated rectangles with a shadow effect based on LGradRectangles1.html, in landscape mode on a Nexus 7 tablet with Android JellyBean.

Radial Color Gradients

A radial color gradient is the second type of HTML5 Canvas-based color gradient. You can define a radial color gradient via the createRadialGradient() method, using the addColorStop() method to add color values. Its syntax (without the addColorStop() method) looks like this:

```
context.createRadialGradient(startCenterX, startCenterY,
  startRadius, endsCenterX, endCenterY, endRadius);
```

A *radial* color gradient can be compared to the ripple effect that is created when you drop a stone in a pond, where each "ripple" has a color that changes in a gradient fashion. Each ripple corresponds to a color stop element. For example, if you define a radial

black), then the resultant color gradient will range—in a radial fashion—from red to black. Radial gradients can also contain multiple start/stop color combinations. The point to keep in mind is that radial gradients change colors in a *linear* fashion, but the rendered colors are drawn in a set of expanding concentric circles. Note that "radial gradient" and "radial color gradient" are used interchangeably in this book.

The HTML5 Web page RGradRectangles1.html, which renders line segments, rectangles, and circles in an HTML5 <canvas> element using linear and radial gradients, is available on the CD.

```
gradient1 = context.createRadialGradient(currentX,
                                currentY,
                                0,
                               currentX+rectWidth,
                                currentY+rectHeight,
                                rectWidth);


gradient1.addColorStop(0, '#f00');
gradient1.addColorStop(1, '#00f');
context.fillStyle = gradient1;
context.fillRect(currentX, currentY,
            rectWidth, rectHeight);
```

The HTML5 Web page RGradRectangles1.html is similar to Listing 9.3, except for the use of a radial gradient (instead of a linear gradient) that ranges in a radial fashion from blue to red. The method addColorStop() is invoked four times in order to add four "color stop values" to the radial gradient. This Web page also references HoverAnimation1.css, whose entire source code is available on the CD.

Figure 9.4 Radial gradient rectangles on an iPad3.

Figure 9.4 displays a set of rectangles with a radial gradient based on RGradRectangles1.html in landscape mode on an iPad3.

HTML5 Canvas Transforms and Saving State

HTML5 Canvas enables you to rotate, scale, shear, or translate (shift horizontally and/or vertically) 2D shapes and text strings with the following methods:

```
rotate(x,y)
scale(x,y)
transform(x1,y1,x2,y2,x3,y3)
translate(x,y)
```

One thing to keep in mind is that you specify the transforms you want to apply (along with setting attributes values) *before* actually rendering a graphics shape in your HTML5 Web pages.

The following code snippets illustrate sample values that you can use in the preceding Canvas methods, where context is a JavaScript variable that references the context of an HTML5 <canvas> element:

```
context.rotate(30*Math.PI/180);
context.scale(0.8, 0.4);
context.translate(100, 200);
context.transform(1, 0, 0.5, 1, 0, 0);
```

The rotate() method in the preceding code block references the JavaScript constant Math.PI whose value represents PI radians. In case you have forgotten, PI radians equal

radians (or 30*Math.PI/180) is the same as 30 degrees. You won't need to know anything more about radians, but feel free to perform an Internet search if you want to read some tutorials that provide additional examples.

Two additional APIs in HTML5 Canvas are save() and restore(), which enable you to save the current state of a canvas state, make some changes, and then restore the original state of the canvas. The save() method "pushes" the current state on a stack, and the restore() method "pops" the most recent state that was pushed onto the stack.

You can save (and later restore) a canvas state after having applied any of the transformations listed in this section, and also after having specified values for shadow-related attributes (among others). You can invoke the save() and restore() methods multiple times on a canvas state, which makes these two methods very useful for game-related Web pages. We will not use these two methods in any code samples in this chapter, but you can perform an Internet search to read tutorials and also find code samples.

The HTML5 Web page JQMCanvasTransforms1.html on the CD illustrates how to apply four HTML5 Canvas transforms to a text string.

jCanvas: a jQuery Plugin for HTML5 Canvas

The jCanvas jQuery plugin enables you to use jQuery syntax in order to specify 2D shapes that are rendered in an HTML5 <canvas> element. Its homepage is here:

http://calebevans.me/projects/jcanvas/

The HTML5 Web page JCanvasSamples1.html on the CD illustrates how to render several 2D shapes using jCanvas.

Figure 9.5 The jQuery jCanvas plugin on an iPad3.

Figure 9.5 displays the result of rendering JCanvasSamples1.html in landscape mode on an

You can also use jQuery Mobile with HTML5 Canvas, as shown in the code sample in the next section.

HTML5 Canvas with CSS3 and jQuery Mobile

By now, we hope you understand how to render 2D shapes in HTML5 Canvas. This section contains a code sample that shows you how to combine jQuery Mobile, HTML5 Canvas , and the dynamic creation of HTML elements whenever users tap inside the HTML5 <canvas> element in this Web page.

Keep in mind that although the graphics effects are not necessarily relevant to your requirements, this code sample does illustrate how to handle dynamic creation of elements as well as tap events in jQuery Mobile (which are handled differently from tap events in jQuery).

Listing 9.4 displays the contents of the HTML5 Web page JQMCanvas1.html, and Listing 9.5 displays the CSS stylesheet JQMCanvas1.css whose CSS3 selectors match elements in the HTML5 Web page JQMCanvas1.html.

LISTING 9.4 JQMCanvas1.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>JQueryMobile and Canvas Graphics</title>

    <link rel="stylesheet" href="JQMCanvas1.css" />

    <link rel="stylesheet"
     href="http://code.jquery.com/mobile/1.1.0/
                        jquery.mobile-1.1.0.min.css" />

    <script src="http://code.jquery.com/jquery-2.0.0b1.js">
    </script>
    <script
      src="http://code.jquery.com/jquery-migrate-1.1.0.js">
    </script>

    <script
src="http://code.jquery.com/mobile/1.1.0/
                        jquery.mobile-1.1.0.min.js">
    </script>
```

```
<script>
  var tapCount = 0;
  var xCoord = 0, yCoord = 0;
  var rectWidth = 20, rectHeight = 20;
  var rectColors = new Array('#ff0', '#0f0', '#00f');
  var elem, context;

  var gradient1 = '-webkit-gradient(radial, 5 25%, 5, 10 50%, 20, from(red), color-stop(0.05, orange), color-stop(0.4, yellow), color-stop(0.6, red), color-stop(0.9, blue), to(#fff))';

  var gradient2 = '-webkit-gradient(radial, 5 25%, 5, 10 50%, 20, from(blue), color-stop(0.05, orange), color-stop(0.4, red), color-stop(0.6, black), color-stop(0.9, blue), to(#f00))';

  var gradient3 = '-webkit-gradient(radial, 5 25%, 5, 10 50%, 20, from(blue), color-stop(0.05, yellow), color-stop(0.4, green), color-stop(0.6, red), color-stop(0.9, blue), to(#fff))';

  var gradient4 = '-webkit-gradient(radial, 5 25%, 5, 10 50%, 20, from(blue), color-stop(0.05, yellow), color-stop(0.2, green), color-stop(0.6, blue), color-stop(0.8, red), to(#fff))';

  var currentBG;

  $("#page1").live('pageinit', (function(event){
    // Get the canvas element
    elem = document.getElementById('MyCanvas');
    if (!elem || !elem.getContext) {
      return;
    }

    // Get the canvas 2d context
    context = elem.getContext('2d');
    if (!context) {
      return;
    }

    // user tapped MyCanvas…
    $("#MyCanvas").live('vmousedown',function(event) {
      xCoord = 0.40*(event.clientX)*window.devicePixelRatio;
      yCoord = 0.40*(event.clientY)*window.devicePixelRatio;
```

```
                                s.length];
    context.fillRect(xCoord, yCoord, rectWidth,
                        rectHeight);

    $("#MyCanvas").hide("slow");
   });

   // this makes MyCanvas visible again
   $("#tapInside").live('tap',function(event) {
     $("#MyCanvas").show("slow");
   });

   $("#CanvasParent").live('vmousedown',function(event) {
    if(tapCount % 4 == 0) {
      currentBG  = gradient1;
    } else if(tapCount % 4 == 1) {
      currentBG  = gradient2;
    } else if(tapCount % 4 == 2) {
      currentBG  = gradient3;
    } else {
      currentBG  = gradient4;
    }

    newNode = $('<div>').css({'position':'absolute',
                          'background': currentBG,
                          'width':rectWidth+'px',
                          'height':rectHeight+'px',
                           top: event.clientY,
                           left: event.clientX
                          });

    //append the new rectangle to CanvasParent
    $("#CanvasParent").append(newNode);
  })
 })
);
</script>
```

```
  <style>
    #tapInside { color: #f00; }
    #MyCanvas  { width: 80%; height: 30%; }
  </style>
</head>

<body>
  <div data-role="page" id="page1"
       data-role="page" data-theme="b">
    <div data-role="header">
      <h2>JQuery Mobile and Canvas Graphics</h2>
    </div>

    <div data-role="content">
      <div id="tapInside">
        <p>Tap Inside the Red Rectangle to Hide and Tap Here
          to Show:</p>
      </div>

      <div id="CanvasParent" name="CanvasParent">
        <canvas name="MyCanvas" id="MyCanvas"
              style="background:#f00;width=80%;height=200px">
      </canvas>

      <div id="outer">
        <div id="radial3">Text3</div>
        <div id="radial2">Text2</div>
        <div id="radial4">Text4</div>
        <div id="radial1">Text1</div>
      </div>

      <!— jQuery toggle-handling code —>
      <script>
        $(document).ready(function() {
          $("#outer").toggle(function(){
            $("#radial1").show("slow");
            $("#radial2").hide("slow");
```

```
            $("#radial4").show("slow");
          },function(){
            $("#radial1").hide("slow");
            $("#radial2").show("slow");
            $("#radial3").show("slow");
            $("#radial4").hide("slow");
          });
        });
      </script>
    </div>

    <div data-role="footer">
      <h3>JQuery Mobile and Canvas Graphics</h3>
    </div>
  </div>
</body>
</html>
```

Notice that Listing 9.4 contains a single jQuery Mobile page view. After initializing some JavaScript variables, Listing 9.4 contains the definition of the JavaScript variables gradient1 , gradient2 , gradient3 , and gradient4 , each of which contains the definition of a Webkit -based radial gradient.

When users tap on the <canvas> element whose id has value MyCanvas , the code adds a new rectangle at the location of the tap event, and then the <canvas> element slowly disappears, as shown here:

```
$("#MyCanvas").live('vmousedown',function(event) {
  xCoord = 0.40*(event.clientX)*window.devicePixelRatio;
  yCoord = 0.40*(event.clientY)*window.devicePixelRatio;

  context.fillStyle = rectColors[++tapCount%rectColors.length];
  context.fillRect(xCoord, yCoord, rectWidth, rectHeight);

  $("#MyCanvas").hide("slow");
});
```

Keep in mind that you must use the vmousedown event, because a tap event in jQuery Mobile does not provide you with the coordinates of the location of the tap event. You also need to use the value of window.devicePixelRatio in the calculations for the location of the tap event.

When users tap on the <canvas> element whose id has value tapInside , the hidden <canvas>

the fillRect()method.

When users tap on the <canvas> element whose id value is CanvasParent , the code first uses conditional logic to determine which radial gradient to select and assign to the JavaScript variable currentBG.

Next, a new HTML element is dynamically created at the location of the tap event and appended to the CanvasParent element, as shown here:

```
newNode = $('<div>').css({'position':'absolute',
                    'background': currentBG,
                    'width':'35px',
                    'height':'35px',
                     top: event.pageY,
                     left: event.pageX
                    });

//append the new rectangle to CanvasParent
$("#CanvasParent").append(newNode);
```

Notice that the position property is set to absolute in the preceding code block, which means that this dynamically created element will remain visible whenever the MyCanvas element slowly fades from view. However, all the rectangles that are rendered using the fillRect() method will also slowly disappear.

Finally, whenever users click on any of the bottom four elements that are rendered with radial gradients, the code will cause them to disappear "out of sequence," and the remaining visible elements will be shifted accordingly.

LISTING 9.5 JQMCanvas1.css

```
#outer {
position: relative; top: 10px; left: 0px;
}

#radial1 {
color: red;
font-size: 24px;
height: 100px;
width:  300px;
position: relative; top: 0px; left: 0px;

background: -webkit-gradient(
  radial, 100 25%, 20, 100 25%, 40, from(blue), to(#fff)
 );
```

```
#radial2 {
color: red;
font-size: 24px;
width:  300px;
height: 100px;
position: absolute; top: 0px; left: 300px;

background: -webkit-gradient(
  radial, 100 25%, 20, 150 25%, 40, from(red), to(#fff)
 );
}

#radial3 {
color: blue;
font-size: 24px;
width:  300px;
height: 100px;
position: relative; top: 0px; left: 0px;
background: -webkit-gradient(
  radial, 100 25%, 30, 100 25%, 20, from(yellow), to(#fff)
 );
-webkit-box-shadow:  0px 0px 8px #000;
}

#radial4 {
color: red;
font-size: 24px;
width:  300px;
height: 100px;
position: absolute; top: 100px; left: 300px;

background: -webkit-gradient(
  radial, 100 25%, 20, 100 25%, 40, from(green),
  color-stop(0.2, orange), color-stop(0.4, yellow),
  color-stop(0.6, green), color-stop(0.8, blue),
  to(#fff)
```

}

Listing 9.5 contains four selectors that correspond to the HTML elements with id values radial1,radial2 ,radial3 , and radial4in JQMCanvas1.html. Since each of these selectors defines Webkit-based radial gradients that you have already seen in earlier examples in this book, we won't discuss the details of those gradients.

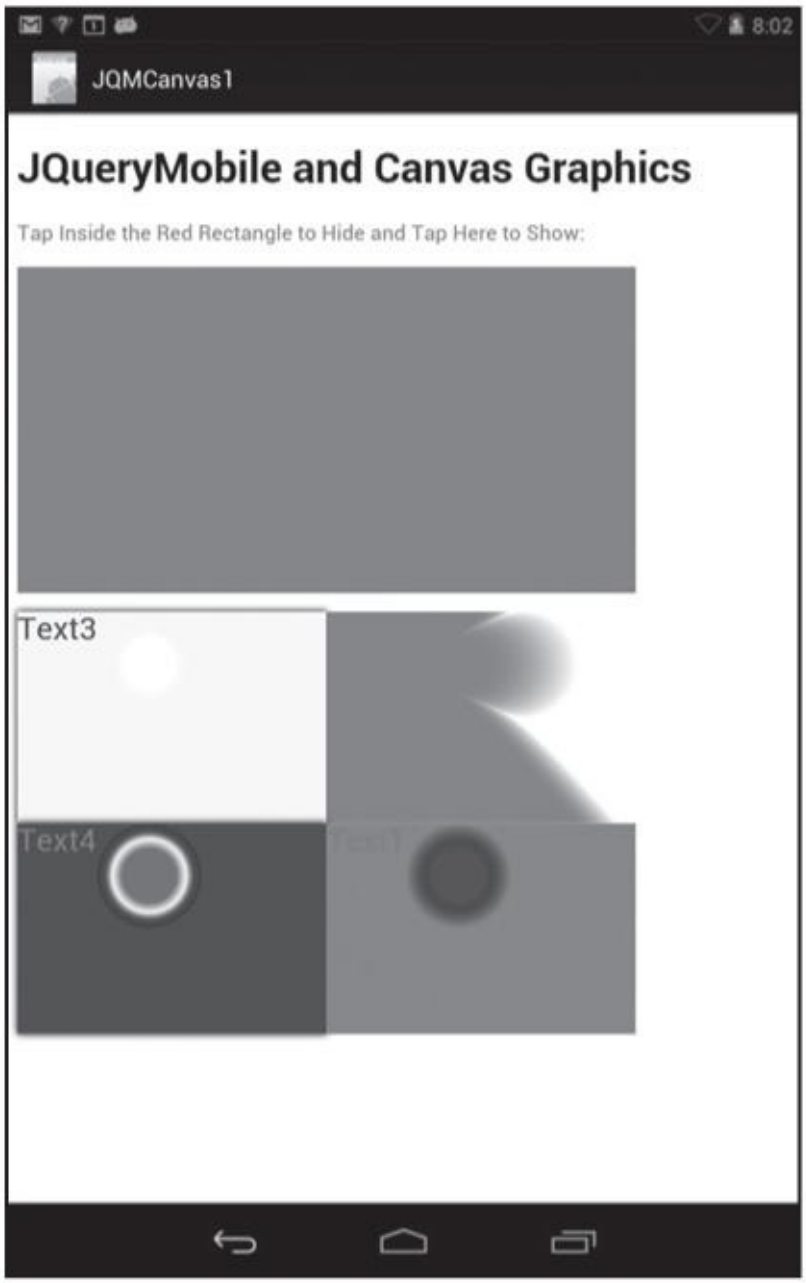


Figure 9.6 A JQuery Mobile application on a Nexus 7 tablet with Android JellyBean.

Figure 9.6 displays the result of rendering JQMCanvas1.html in landscape mode on a Nexus 7 tablet with Android 4.1.

Additional Code Samples on the CD

HTML5 Canvas provides support for both quadratic Bézier curves and cubic Bézier

combinations of Bézier curves by using various types of gradient shading. Later in this book, you will see how to use ECMAScript in order to programmatically change the values of attributes.

Bézier curves are named after Pierre Bézier, who promoted them during the 1970s. Bézier curves can represent many non-linear shapes, they can be found in interesting applications, including PostScript for the representation of fonts. An Internet search will yield many Web pages with interesting demonstrations (some of which also require additional plug-ins). You'll find computer programs written in C and Java, some of which are interactive, that demonstrate Bézier curves.

Cubic Bézier curves have two end points and two control points, whereas quadratic Bézier curves have two end points and a single control point. The x-coordinate and y-coordinate of a cubic Bézier curve can be represented as a parameterized cubic equation whose coefficients are derived from the control points and the end points. The beauty of HTML5 Canvas is that it allows you to define both quadratic and cubic Bézier curves via the <path> element without having to delve into the mathematical underpinnings of Bézier curves. If you're interested in learning the specific details, you can browse the Web, where you'll find books and plenty of articles that cover this interesting topic.

HTML5 Canvas provides the quadraticCurveTo() method for creating quadratic Bézier curves, which requires one control point and an end point. HTML5 Canvas also provides the bezierCurveTo() method for creating cubic Bézier curves, which requires you to specify two control points and an end point. The context point (which is the location of the most recently rendered point) is used as the start point for quadratic and cubic Bézier curves.

The syntax for the HTML5 Canvas quadraticCurveTo() method looks like this:

```
quadraticCurveTo(controlX, controlY, endX, endY);
```

The syntax for the HTML5 Canvas bezierCurveTo() method looks like this:

```
bezierCurveTo(controlX1,controlY1,controlX2,controlY2,endX,endY);
```

The HTML5 page LRGradQCBezier1.html demonstrates how to render a quadratic Bézier curve with linear gradient shading and a cubic Bézier curve with radial gradient shading. The two CSS stylesheets CSS3Background4.css and HoverAnimation1.css are available on the CD.

The cubic Bézier curve defined in the HTML5 page LRGradQCBezier1.html is rendered with a radial gradient using six color stops, based on several calculated points, as shown here:

```
context.bezierCurveTo(
    currentX+3*multiplier*rectWidth,
    currentY+2*multiplier*rectHeight,
    currentX+2*multiplier*rectWidth,
    currentY-multiplier*rectHeight,
    currentX+100, currentY+300);
```

other ways to create pleasing visual effects. The next portion of Listing 9.6 renders a quadratic Bézier curve using a linear gradient with five color stops.

Note that whenever users click on the "redraw" button, another cubic and quadratic Bézier curve are drawn, based on the value of the variable clickCount that is incremented each time that users click on the "redraw" button. The new curves are superimposed on the previous curves, thereby creating a nice visual effect. However, if you want to refresh the HTML5 <canvas>element prior to rendering another pair of Bézier curves, simply uncomment the second line in the following code snippet:

```
// clear the canvas before drawing new set of rectangles
//context.clearRect(0, 0, elem.width, elem.height);
```

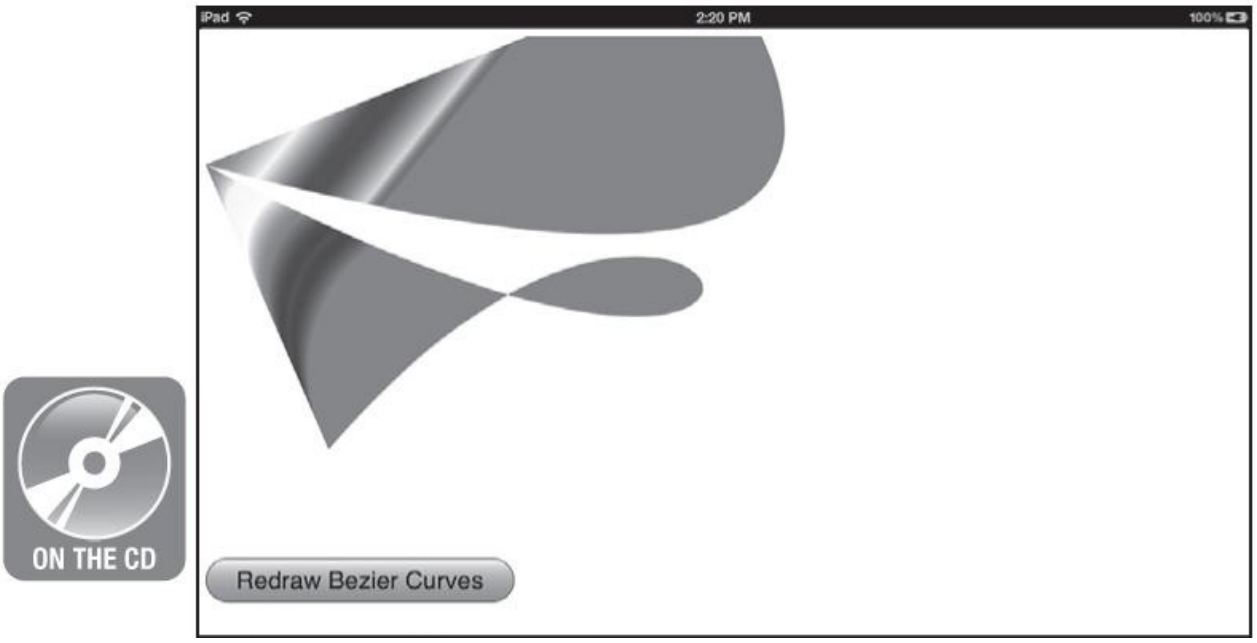


Figure 9.7 Gradient Bézier curves on an iPad3.

Figure 9.7 renders the quadratic and cubic Bézier curves that are defined in LRGradQCBezier1.html in landscape mode on an iPad3.

HTML5 Canvassupports the rendering of JPG files, and you can also apply CSS selectors to the HTML5 <canvas> element. The CD contains the HTML5 Web page Image1.html and Image1.css whose selectors match the HTML5 <canvas>element.

Incidentally, HTML5 Canvasalso supports a clip() method that enables you to "clip" JPG files in various ways. Moreover, you can perform compositing effects, and you can even manipulate the individual pixels of a JPG file. Search the Internet for articles that describe the technical details of these effects.

HTML5 Canvasprovides the method createPattern(image, type) that enables you to render a set of images according to a pattern type, whose values can be repeat , repeat-x, repeat-y, and no-repeat . An example of the syntax (and also how to use it) looks like this:

```
var pattern = canvas.createPattern(img,"repeat");
canvas.fillStyle = pattern;
```

The HTML5 Web page RepeatingImage1.html on the CD illustrates how to repeat a JPG image on an HTML5 <canvas> element.

Figure 9.8 displays RepeatingImage1.html in landscape mode on the Chrome browser on a MacBook.

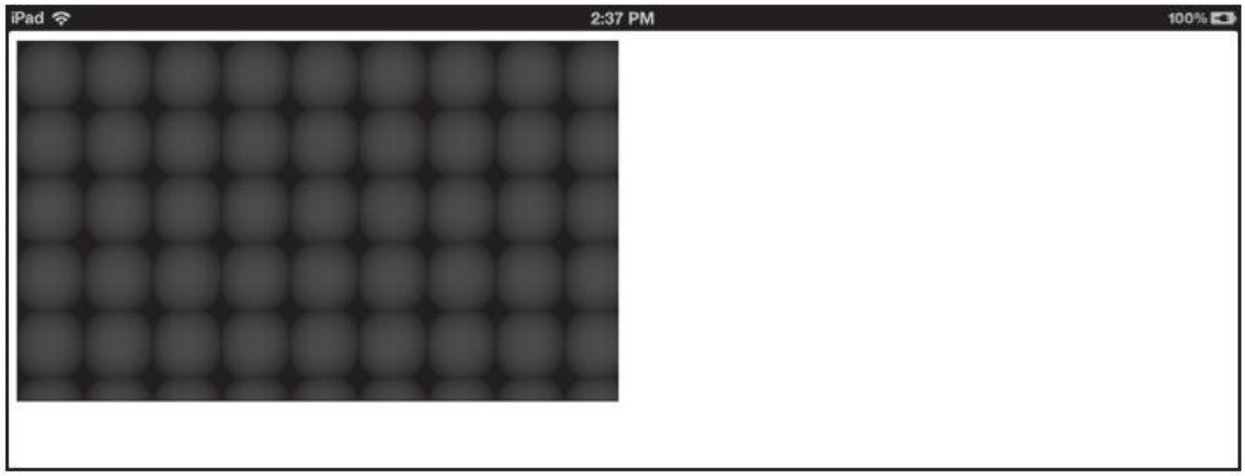


Figure 9.8 Repeating JPG on the Chrome browser on a MacBook.

Other HTML5 Canvas Toolkits

There are several very good JavaScript toolkits available that provide a layer of abstraction on top of HTML5 Canvas , including Easel.js , Fabric.js, and Paper.js .

In addition to the code samples that are available on the respective homepage of these toolkits, you might enjoy the contents of the following open source project, which uses Easel.js to create graphics code samples:

http://code.google.com/p/easeljs-graphics

Summary

This chapter introduced you to HTML5 Canvas and showed you examples of creating 2D shapes with the HTML5 <canvas> element. You also learned how to combine HTML5 Canvas with jQuery custom code so that you can manipulate HTML5 Web pages with Canvas -based 2D shapes. In particular, you learned how to do the following in HTML5 Canvas :

- Render line segments, rectangles, and circles
- Create linear and radial gradients
- Create Bézier curves
- Display JPG files
- Use jCanvas (a jQuery plugin for Canvas)
- Combine HTML5 Canvas with CSS3
- Combine HTML5 Canvas with jQuery Mobile

USING PHONEGAP FOR HTML5 MOBILE APPS

This chapter shows you how to create HTML5-based hybrid mobile applications for Android and iOS. The code samples in this chapter contain HTML5 and various combinations of HTML5, CSS3, and SVG.

As you will soon discover, there are more Android-based code samples than iOS-based code samples. The choice of mobile platform for the code samples is purely a stylistic one. However, every Android-based code sample does have an iOS-based counterpart, and vice versa (both platforms provide the necessary feature support for all the samples in this chapter). Although it was possible to include the same set of code samples for both platforms, doing so would have been needlessly redundant. Moreover, this chapter provides you with the information that you need in order to "convert" an Android-based code sample to its iOS counterpart (and vice versa).

The first part of this chapter provides an overview of how to develop hybrid Android applications using a "manual" approach instead of a toolkit such as PhoneGap. The code samples in this section use the same code that you have seen in earlier chapters, and they show you how to create the hybrid Android mobile applications that will enable you to create the same screenshots. If you feel ambitious, you can create Android-based mobile applications for all the code samples in this book!

The second part of this chapter contains Android-based code samples that show you how to combine native Android applications with CSS3, SVG, and HTML5 Canvas. This section contains an example of rendering a mouse-enabled multi-line graph whose values can be updated whenever users click on the button that is rendered underneath the line graph. Keep in mind that the discussion following the code samples moves quickly because the HTML Web pages contain simple markup, the CSS3 selectors contain code that you have seen in earlier chapters, and the SVG shapes are discussed in Chapter 4.

The third part of this chapter provides a quick overview of Apache Cordova (formerly known as PhoneGap), which is a popular cross-platform toolkit for developing mobile applications. In 2011 Adobe acquired Nitobi, the company that created PhoneGap, and shortly thereafter Adobe open sourced PhoneGap. This section explains what PhoneGap can do, and some toolkits that you can use with PhoneGap. You will learn how to create a PhoneGap-based Android application that renders CSS3-based animation effects. You can deploy this mobile application to Android-based mobile devices that support Android ICS or higher.

The final part of this chapter discusses how to create iOS hybrid mobile applications using the PhoneGap plugin for Xcode.

As you will see in this chapter, PhoneGap allows you to create mobile applications

platforms, including Android, iOS, BlackBerry and Windows Mobile. You can also create mobile applications that combine PhoneGap with Sencha Touch, another popular framework). However, due to space limitations, Sencha Touch is not discussed in this chapter.

If you are unfamiliar with any of the mobile platforms in this chapter, you can still work through the examples in this chapter because they consist of HTML5-based code. The sequence of steps for creating HTML5-based mobile applications on a mobile platform is essentially independent of the actual code.

HTML5/CSS3 and Android Applications

If you are unfamiliar with Android, you can read the Appendix for this book that contains a concise overview of the Android-specific concepts in the code samples in this chapter. You can refer to the appropriate section whenever you encounter an Android concept that is not clear to you.

The code sample in this section shows you how to launch an HTML5 Web page (which also references a CSS3 stylesheet) inside an Android application. The key idea consists of three steps:

1. Modify the Android `Activity` class to instantiate an Android `WebView` class, along with some JavaScript-related settings.
2. Reference an HTML5 Web page that is in the `assets/www` subdirectory of the Android project.
3. Copy the HTML5 Web page, CSS stylesheets, and JavaScript files into the `assets/www` subdirectory of the Android project.

In Step 3, you will probably create a hierarchical set of directories that contain files that are of the same type (HTML, CSS, or JavaScript), in much the same way that you organize your files in a Web application.

Now launch Eclipse and create an Android project called `AndroidCSS3`. Make sure that you select Android version 3.1 or higher, which is necessary in order to render CSS3-based effects.

After you have created the project, let's take a look at four files that contain the custom code for this Android mobile application. Listings 10.1, 10.2, and 10.3 respectively display the contents of the project files `main.xml`, `AndroidCSS3.html`, and `AndroidCSS3Activity.java`.

LISTING 10.1 main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/
                        res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
  <WebView android:id="@+id/webview"
```

```
        android:layout_height="fill_parent">
    </WebView>
</LinearLayout>
```

Listing 10.1 specifies a LinearLayout that contains an Android WebView, which will occupy the entire screen of the mobile device. This is the behavior that we want to see, because the Android default browser is rendered inside the Android WebView.

LISTING 10.2 AndroidCSS3.html

```
<!doctype html>
<head>
  <title>CSS Radial Gradient Example</title>
  <link href="AndroidCSS3.css" rel="stylesheet">
</head>

<body>
 <div id="outer">
  <div id="radial1">Text1</div>
  <div id="radial2">Text2</div>
  <div id="radial3">Text3</div>
  <div id="radial4">Text4</div>
 </div>
</body>
</html>
```

Listing 10.2 is a straightforward HTML Web page that references a CSS stylesheet AndroidCSS3.css (that is available on the CD), along with an HTML element (whose id attribute has value outer) that serves as a "container" for four more HTML elements.

The CSS stylesheet AndroidCSS3.csscontains a CSS selector for styling the HTML element whose id has value outer, followed by four CSS selectors radial1, radial2, radial3, and radial4 that are used to style the corresponding HTML elements in Listing 10.2. The contents of these selectors ought to be very familiar (you can review the material for CSS3 gradients in an earlier chapter), so we will not cover their contents in this section.

LISTING 10.3: AndroidCSS3Activity.java

```
package com.iquarkt.css3;


import android.app.Activity;
import android.os.Bundle;
```

```
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;

public class AndroidCSS3Activity extends Activity
{
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Get a reference to the declared WebView holder
    WebView webview = (WebView) this.findViewById(R.id.webview);

    // Get the settings
    WebSettings webSettings = webview.getSettings();

    // Enable Javascript for interaction
    webSettings.setJavaScriptEnabled(true);

    // Make the zoom controls visible
    webSettings.setBuiltInZoomControls(true);

    // Allow for touching selecting/deselecting data series
    webview.requestFocusFromTouch();

    // Set the client
    webview.setWebViewClient(new WebViewClient());
    webview.setWebChromeClient(new WebChromeClient());

    // Load the URL
    webview.loadUrl("file:///android_asset/AndroidCSS3.html");
  }
}
```

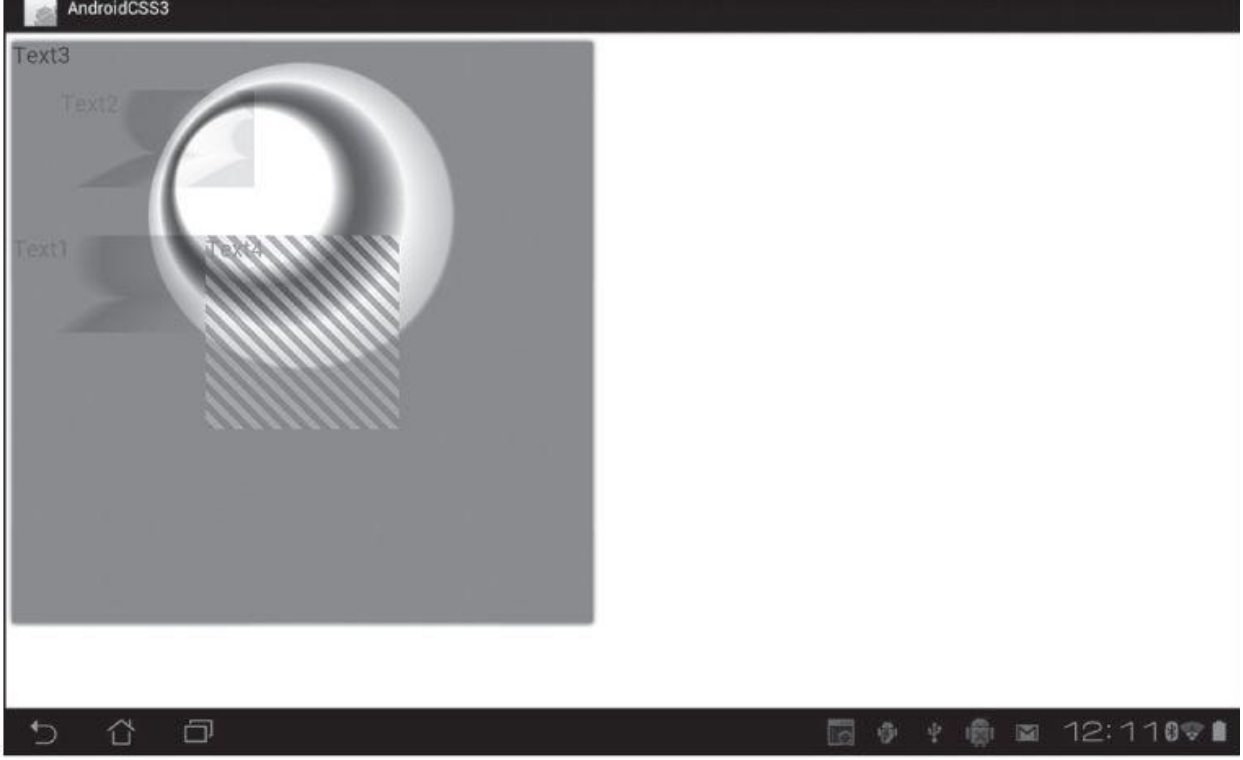


Figure 10.1 A CSS3-based 3D cube on an Asus Prime tablet with Android ICS.

Listing 10.3 defines a Java class AndroidCSS3Activity that extends the standard Android Activity class. This class contains the onCreate() method that "points" to the XML document main.xml (displayed in Listing 10.2) so that we can get a reference to its WebView child element via R.id.webview (which is the reference to the WebView element in Listing 10.2), as shown here:

```
WebView webview = (WebView) this.findViewById(R.id.webview);
```

Next, the webSettings instance of the WebSettings class enables us to set various properties, as shown in the commented lines of code in Listing 10.4.

The final line of code loads the contents of the HTML Web page AndroidCSS3.html (which is in the assets/www subdirectory), as shown here:

```
webview.loadUrl("file:///android_asset/AndroidCSS3.html");
```

Figure 10.1 displays a CSS3-based Android application on an Asus Prime tablet with Android ICS.

SVG and Android Applications

The example in this section shows you how to create an Android mobile application that renders SVG code that is embedded in an HTML5 Web page. Now launch Eclipse and create an Android project called AndroidSVG1, making sure that you select Android version 3.1 or higher, which is necessary in order to render SVG elements.

The example in the previous section contains four custom files, whereas the Android/SVG example in this section contains two files with custom code: the HTML5 Web page AndroidSVG1.html in Listing 10.4 and the Java class AndroidSVG1.java, which is

LISTING 10.4 AndroidSVG1.html

```
<!DOCTYPE html>
<html>
  <body>
    <h1>HTML5/SVG Example</h1>
    <svg>
      <ellipse cx=“300” cy=“50” rx=“80” ry=“40”
            fill=”#ff0” stroke-dasharray=“8 4 8 1”
            style=“stroke:red;stroke-width:4;”/>

      <line x1=“100” y1=“20” x2=“300” y2=“350”
            stroke-dasharray=“8 4 8 1”
            style=“stroke:red;stroke-width:8;”/>

      <g transform=“translate(20,20)”>
        <path
          d=“M0,0 C200,150 400,300 20,250”
          fill=”#f00”
          stroke-dasharray=“4 4 4 4”
          style=“stroke:blue;stroke-width:4;”/>
      </g>

      <g transform=“translate(200,50)”>
        <path
          d=“M200,150 C0,0 400,300 20,250”
          fill=”#00f”
          stroke-dasharray=“12 12 12 12”
          style=“stroke:blue;stroke-width:4;”/>
      </g>
    </svg>
  </body>
</html>
```

Listing 10.4 is an HTML Web page that contains an SVG document with the definitions for an ellipse, a line segment, and two cubic Bézier curves. Appendix A contains examples of these 2D shapes (among others), and you can review the appropriate material if you need to refresh your memory.

The Java class AndroidSVG1Activity.java is omitted, but its contents are very similar to Listing 10.3. The complete source code is available on the CD.

Figure 10.2 displays an SVG-based Android application on an Asus Prime tablet with Android ICS.

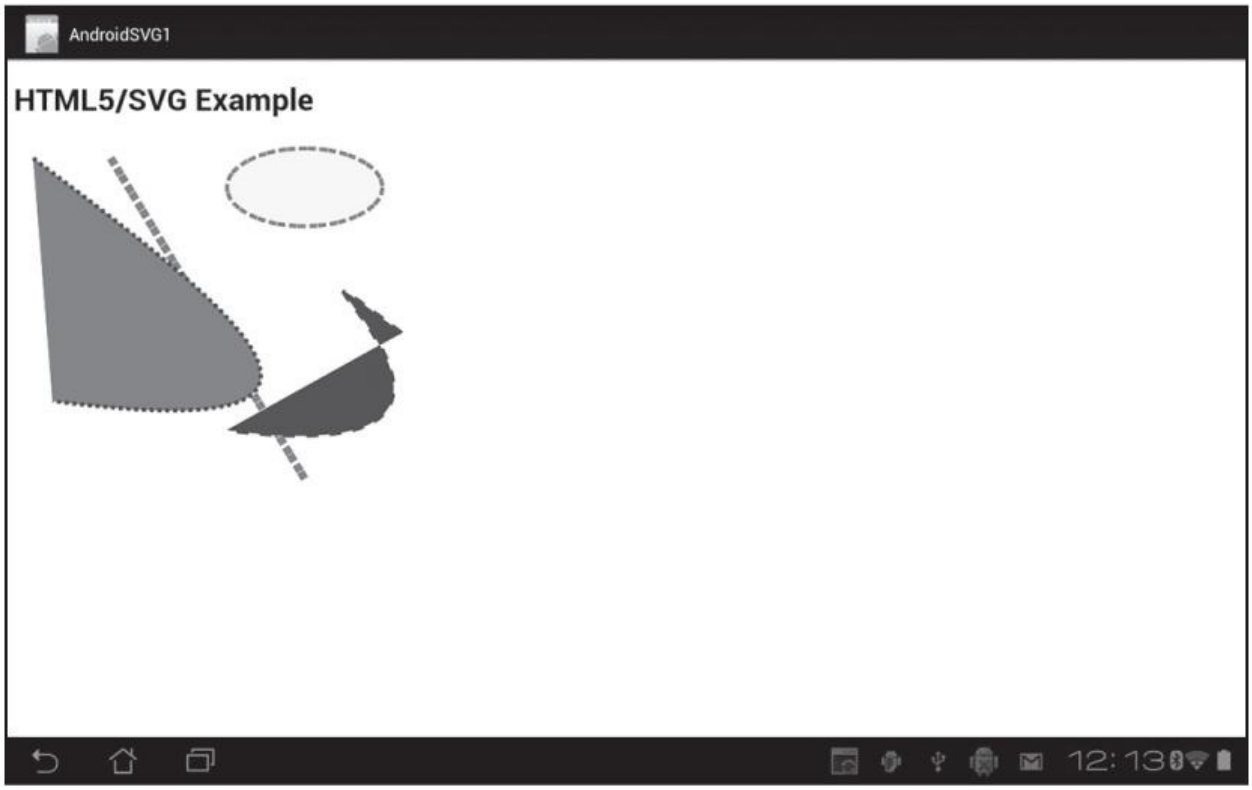


Figure 10.2 An SVG-based Android application on an Asus Prime tablet with Android ICS.

HTML5 Canvas and Android Applications

In addition to rendering CSS3-based effects and SVG documents, you can also render Canvas-based 2D shapes in an Android application. Launch Eclipse and create an Android project called AndroidCanvas1. Make sure that you select Android version 3.1 or higher, which is necessary in order to render SVG elements.

The example in this section contains one custom file called AndroidCanvas1.html, which is displayed in Listing 10.5.

LISTING 10.5 AndroidCanvas1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8" />
 <title>HTML5 Canvas Example</title>
```

```
function draw() {
  var basePointX  = 10;
  var basePointY  = 80;
  var currentX    = 0;
  var currentY    = 0;
  var startAngle  = 0;
  var endAngle    = 0;
  var radius      = 120;
  var lineLength  = 200;
  var lineWidth   = 1;
  var lineCount   = 200;
  var lineColor   = "";

  var hexArray    = new Array('0','1','2','3','4','5','6','7',
                        '8','9','a','b','c','d','e','f');

  var can = document.getElementById('canvas1');
  var ctx = can.getContext('2d');

  // render a text string…
  ctx.font = "bold 26px helvetica, arial, sans-serif";
  ctx.shadowColor = "#333333";
  ctx.shadowOffsetX = 2;
  ctx.shadowOffsetY = 2;
  ctx.shadowBlur = 2;
  ctx.fillStyle = 'red';
  ctx.fillText("HTML5 Canvas/Android", 0, 30);

  for(var r=0; r<lineCount; r++) {
     currentX = basePointX+r;
     currentY = basePointY+r;
     startAngle = (360-r/2)*Math.PI/180;
     endAngle   = (360+r/2)*Math.PI/180;

     // render the first line segment…
     lineColor = '#' + hexArray[r%16] + '00';
     ctx.strokeStyle = lineColor;
```

```
      ctx.beginPath();
      ctx.moveTo(currentX, currentY+2*r);
      ctx.lineTo(currentX+lineLength, currentY+2*r);
      ctx.closePath();
      ctx.stroke();
      ctx.fill();

      // render the second line segment…
      lineColor = '#' + '0' + hexArray[r%16] + '0';
      ctx.beginPath();
      ctx.moveTo(currentX, currentY);
      ctx.lineTo(currentX+lineLength, currentY);
      ctx.closePath();
      ctx.stroke();
      ctx.fill();

      // render the arc…
      lineColor = '#' + '00'+ hexArray[(2*r)%16];
      ctx.beginPath();
      ctx.fillStyle = lineColor;
      ctx.moveTo(currentX, currentY);
      ctx.arc(currentX, currentY, radius,
            startAngle, endAngle, false);
      ctx.closePath();
      ctx.stroke();
      ctx.fill();
   }
}
</script>
</head>

<body onload="draw()">
  <canvas id="canvas1" width="300px" height="200px"></canvas>
</body>
<html>
```

Listing 10.5 contains some boilerplate HTML markup and a JavaScript function draw()

contains JavaScript code that draws a set of line segments and arcs into the HTML5 <canvas> element, whose id attribute has value canvas1. You can review the code samples in chapter eleven that have similar functionality if you don't remember the details of the syntax of this JavaScript code.

Figure 10.3 A Canvas-based Android application on an Asus Prime tablet with Android ICS.

Figure 10.3 displays a Canvas -based Android application on an Asus Prime tablet with Android ICS.

The next portion of this chapter delves into PhoneGap, which is a toolkit that automatically creates the lower level "scaffolding" that you performed manually in the previous part of this chapter. You will get instructions for installing the PhoneGap plugin for Eclipse to create Android mobile applications. Later in this chapter, you will also learn how to install the PhoneGap plugin for Xcode in order to create HTML5-based mobile applications for iOS mobile devices.

What is PhoneGap?

PhoneGap is an open source device agnostic mobile application development tool that enables you to create cross-platform mobile applications using CSS, HTML, and JavaScript, and its homepage is here:

http://phonegap.com

The PhoneGap homepage provides documentation, code samples, and a download link for the PhoneGap distribution.

PhoneGap enables you to create HTML-based mobile application for Android,

for touch events, event listeners, rendering images, database access, different file formats (XML and JSON), and Web Services.

Note that if you want to develop iPhone applications, you must have a MacBook or some other OS X machine, along with other dependencies that are discussed later in this chapter.

How Does PhoneGap Work?

PhoneGap mobile applications involve a Web view that is embedded in a native "shell," and your custom code runs in the Web view. In addition, PhoneGap provides a JavaScript API for accessing native features of a mobile device, and your code can use PhoneGap in order to access those native features. For example, PhoneGap contains JavaScript APIs for accessing Accelerometer, Camera, Compass, Contacts, Device information, Events, Geolocation, Media, Notification, and Storage.

Keep in mind that PhoneGap does not provide HTML UI elements. Thus, if you need this functionality in your mobile applications, you can add other toolkits and frameworks, such as jQuery Mobile, Sencha Touch, or Appcelerator.

Now that you have a basic understanding of the capabilities of PhoneGap, install the PhoneGap 2.0 (which was released as this book goes to print) for Xcode and Eclipse by following the instructions here:

http://outof.me/phonegap-2-0-getting-started/

In case you prefer to compile your mobile applications in the "cloud," Adobe provides a Website for this purpose:

https://build.phonegap.com/

If you have completed the installation of the PhoneGap plugin for Eclipse, you are now ready to create a PhoneGap application for Android, which is the topic of the next section.

Creating Android Apps with The PhoneGap Plugin

Create an Android project in Eclipse by clicking on the icon for the PhoneGap plugin and then (for the purposes of this example) specify `PGJQM1` for the `Project Name`, check the checkbox for including the jQuery Mobile files, select the Android version that your Android device supports, and then enter `com.iquarkt.phonegap` as the package name.

Click the "Finish" button. After the project has been created, navigate to the `assets/www` subdirectory of the newly created Android project. You will find the following files (version numbers might be different when this book goes to print):

index.html

phonegap-1.3.0.js

There is also a generated Java file `PhoneGap1Activity.java`, whose contents are displayed in Listing 10.6

LISTING 10.6 PhoneGap1Activity.java

```
import com.phonegap.*;
import android.os.Bundle;

public class PhoneGap1Activity extends Activity
{
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    super.loadUrl("file:///android_asset/www/index.html");
  }
}
```

Listing 10.6 contains an `onCreate()` method that launches the HTML page `index.html`, as shown here:

```
super.loadUrl("file:///android_asset/www/index.html");
```

The HTML page `index.html` is located in the `assets/www` subdirectory of the project, and its contents are displayed in Listing 10.7.

LISTING 10.7 index.html

```
<!DOCTYPE HTML>
<html>
 <head>
  <meta name="viewport" content="width=320; user-scalable=no" />
  <meta http-equiv="Content-type" content="text/html;
                              charset=utf-8">
  <title>PhoneGap</title>

  <link rel="stylesheet" href="master.css"
      type="text/css" media="screen"
      title="no title" charset="utf-8">
  <script src="phonegap-1.0.0.js"></script>
  <script src="main.js"></script>
 </head>

 <body onload="init();" id="stage" class="theme">
  <h1>Welcome to PhoneGap!</h1>
```

```
    <div id="info">
      <h4>Platform: <span id="platform">  </span>,
        Version: <span id="version"> </span></h4>
      <h4>UUID: <span id="uuid">  </span>,
        Name: <span id="name"> </span></h4>
      <h4>Width: <span id="width">  </span>,
        Height: <span id="height"> </span>,
        Color Depth: <span id="colorDepth"></span></h4>
    </div>

    <dl id="accel-data">
      <dt>X:</dt><dd id="x"> </dd>
      <dt>Y:</dt><dd id="y"> </dd>
      <dt>Z:</dt><dd id="z"> </dd>
    </dl>

    <a href="#" class="btn large" onclick="toggleAccel();">
                        Toggle Accelerometer</a>
    <a href="#" class="btn large" onclick="getLocation();">
                        Get Location</a>
    <a href="tel://411" class="btn large">Call 411</a>
    <a href="#" class="btn large" onclick="beep();">Beep</a>
    <a href="#" class="btn large" onclick="vibrate();">
                            Vibrate</a>
    <a href="#" class="btn large" onclick="show_pic();">
                            Get a Picture</a>
    <a href="#" class="btn large" onclick="get_contacts();">
                        Get Phone's Contacts</a>
    <a href="#" class="btn large" onclick="check_network();">
                            Check Network</a>

    <div id="viewport" class="viewport" style="display: none;">
      <img style="width:60px;height:60px" id="test_img" src="" />
    </div>
  </body>
</html>
```

The first portion of Listing 10.7 contains a <script> element that includes the JavaScript

PhoneGap.

The second portion of Listing 10.7 displays the anchor elements that enable you to test media-related features of your phone, including accelerometer, geolocation, making phone calls (from inside the Android application), beep effects, vibration effects, and taking pictures with the camera on your smart phone or tablet.

Listing 10.8 displays the contents of the JavaScript file main.js , which contains selected portions of the JavaScript code that supports functionality in the HTML5 Web page index.html.

LISTING 10.8 main.js

```
var deviceInfo = function() {
   document.getElementById("platform").innerHTML = device.
                                        platform;
   document.getElementById("version").innerHTML = device.
                                       version;
   document.getElementById("uuid").innerHTML = device.uuid;
   document.getElementById("name").innerHTML = device.name;
   document.getElementById("width").innerHTML = screen.width;
   document.getElementById("height").innerHTML = screen.height;
   document.getElementById("colorDepth").innerHTML =
                                    screen.colorDepth;
};

// sections omitted for brevity
function dump_pic(data) {
   var viewport = document.getElementById('viewport');
   console.log(data);
   viewport.style.display = "";
   viewport.style.position = "absolute";
   viewport.style.top = "10px";
   viewport.style.left = "10px";
   document.getElementById("test_img").src =
                    "" + data;
}

function fail(msg) {
   alert(msg);
}
```

```
function show_pic() {
   navigator.camera.getPicture(dump_pic, fail, {
      quality : 50
   });
}

// details omitted for brevity
```

The first part of Listing 10.8 contains the code for getting the data from the accelerometer of your Android device. The second part of Listing 10.9 shows you the JavaScript code for taking a picture from this Android application.

Now navigate to Run > Android application in order to launch this Android project and on your Android device you will see something similar to Figure 10.4.

Figure 10.4 A PhoneGap-based Android mobile application.

Figure 10.4 displays a set of menu items that enable you to access hardware-related functionality.

Working with HTML5, PhoneGap, and iOS

This section shows you how to create iOS mobile applications using PhoneGap, which

(Their screenshots on an iPad3 are included in various chapters.) Every iOS mobile application in this book was developed on a MacBook OS X 10.8.2 with Apple's Xcode 4.5 and PhoneGap.

Earlier in this chapter you learned how to create Android applications in Eclipse, which is an IDE that runs on multiple OSes, but the situation is different for creating iOS applications (with or without PhoneGap).

First you need access to an Apple device (such as a MacBook, Mac Mini, or Mac Pro) with Apple's Xcode installed in order to create mobile applications for iOS mobile devices. If you register as a developer you can download Xcode for free, or for $4.99 in the Apple iStore. Although this section uses Xcode 4.5, it is possible to install a lower version of Xcode. (However, make sure that you check the minimum required version for OS X.)

Secondly, you need to install the PhoneGap plugin for Xcode 4 by following the detailed instructions here (which also contain a link for installing PhoneGap on Xcode 3):

http://wiki.phonegap.com/w/page/39991939/Getting%20Started%20with%20PhoneGa

Thirdly, you need to register as an Apple Developer (which costs $99 per year) *if you want to deploy your iOS mobile applications to iOS devices*. However, if you only plan to use the iOS Simulator, you can do so at no charge.

After you have set up a laptop with the required software, you will be ready to create an iOS mobile application with PhoneGap, which is the topic of the next section.

NOTE

PhoneGap applications always have the same filename index.html*, so in order to provide multiple PhoneGap project files in the same directory on the CD, the HTML Web page* index.html *for each PhoneGap project is saved in a Web page whose name is the same as the project. For example, the contents of the HTML Web page* ThreeDCube1.html *in the next section are actually the same as the generated Web page* index.html *that is specific to the PhoneGap project in the next section.*

A CSS3 Cube on iOS Using PhoneGap

Create an Xcode application called ThreeDCube1by selecting the PhoneGap plugin. (Make sure that your filenames start with an alphabetical character, or you will get errors when you attempt to compile and deploy your applications.)

Note that if you are using Xcode 4.3.1, then you will need to perform a manual copy of the generated www subdirectory into the project home directory of your current Xcode application. When you have performed this step correctly, you will no longer see an error message when you launch your mobile application in the Simulator or on your iOS device.

The CSS stylesheet ThreeDCube1.css is the same as the CSS file 3DCube1.css in Chapter 2,

page ThreeDCube1.html that is the same as the HTML Web page index.htmlthat is generated by the PhoneGap plugin in XCode.

LISTING 10.9 index.html

```
<!DOCTYPE html>
<html>
 <head>
 <title>CSS 3D Cube Example</title>
 <link href="ThreeDCube1.css" rel="stylesheet" type="text/css">

 <meta name="viewport"
  content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no;" />
 <meta charset="utf-8">

 <!— iPad/iPhone specific css below, add after your main css >
 <link rel="stylesheet" media="only screen and (max-device-width: 1024px)" href="ipad.css" type="text/css" />
 <link rel="stylesheet" media="only screen and (max-device-width: 480px)" href="iphone.css" type="text/css" />
 —>

 <!— If your application is targeting iOS BEFORE 4.0 you MUST put json2.js from
     http://www.JSON.org/json2.js into your www directory and include it here —>

 <script src="phonegap-1.3.0.js"></script>

 <script type="text/javascript">
 // If you want to prevent dragging, uncomment this section
 /*
 function preventBehavior(e)
 {
    e.preventDefault();
  };
 document.addEventListener("touchmove", preventBehavior, false);
 */

 function onBodyLoad()
 {
  document.addEventListener("deviceready", onDeviceReady,
   false);
```

```
function onDeviceReady()
{
  // do your thing!
//navigator.notification.alert("PhoneGap is working")
}
</script>
</head>

<body onload="onBodyLoad()">
<div id="outer">
 <div id="top">Text1</div>
 <div id="left">Text2</div>
 <div id="right">Text3</div>
</div>
</body>
</html>
```

Listing 10.9 is the result of combining the HTML Web page 3DCube1.html from Chapter 2 (which is essentially a set of HTML elements) with the HTML Web page index.html that is automatically generated by PhoneGap when you create a mobile application in Xcode using the PhoneGap plugin.

Now run this mobile application, either in the Xcode Simulator or on your mobile device, and you will see a graphics image that is similar to Figure 10.5.

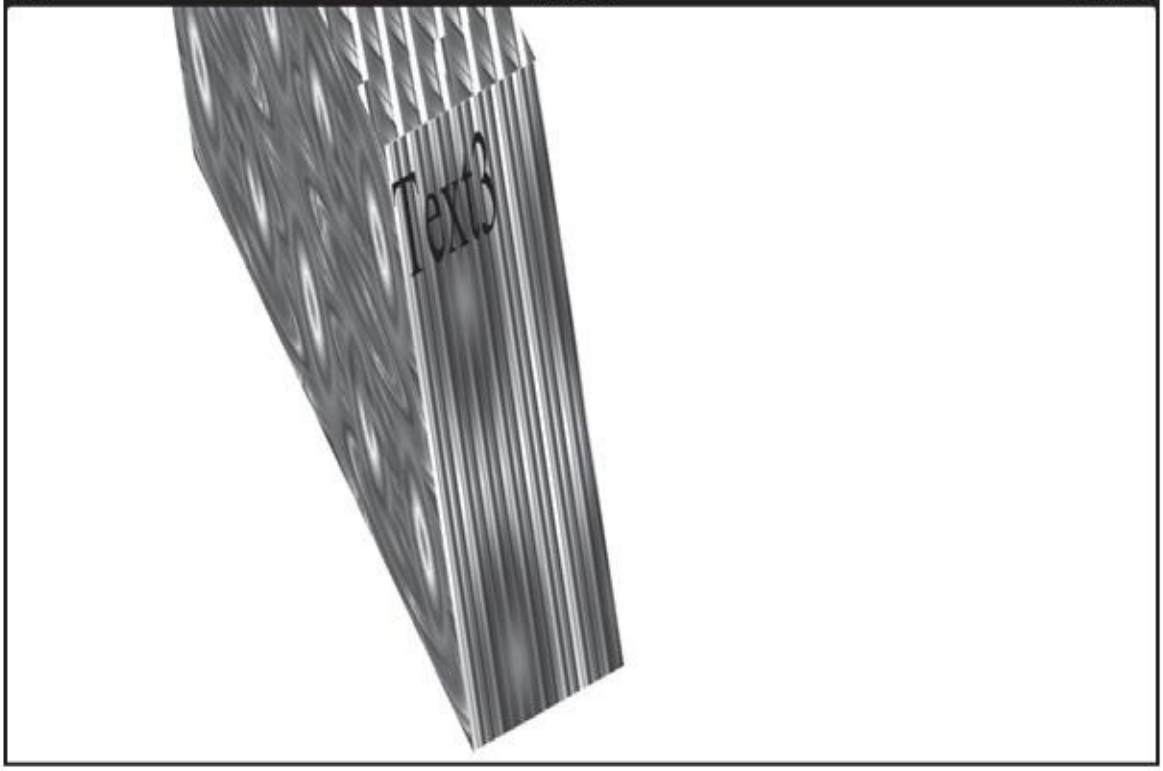


Figure 10.5 A CSS3 cube on an iPad3.

The process for creating the other iOS-based mobile applications in this chapter is identical to the process for the preceding iOS mobile application, so there is no need to include additional examples. However, it's worth your while to spend some time creating additional iOS mobile applications, which will increase your comfort level and perhaps also motivate you to learn about other features of Xcode.

Additional Code Samples on the CD

Although Android does not have built-in support for rendering charts and graphs, you can create them using `Canvas` -based code that is very similar to the code in the previous section.

Launch Eclipse and create an Android project called `AndroidCanvasMultiLine2` , making sure that you select Android version 3.1 or higher. The HTML5 Web page `AndroidCanvasMultiLine2.html` on the CD contains JavaScript code for rendering multiple line graphs using HTML5 `Canvas` .

Figure 10.6 displays a `Canvas`-based multi-line graph Android application on a Nexus S 4G with Android ICS.

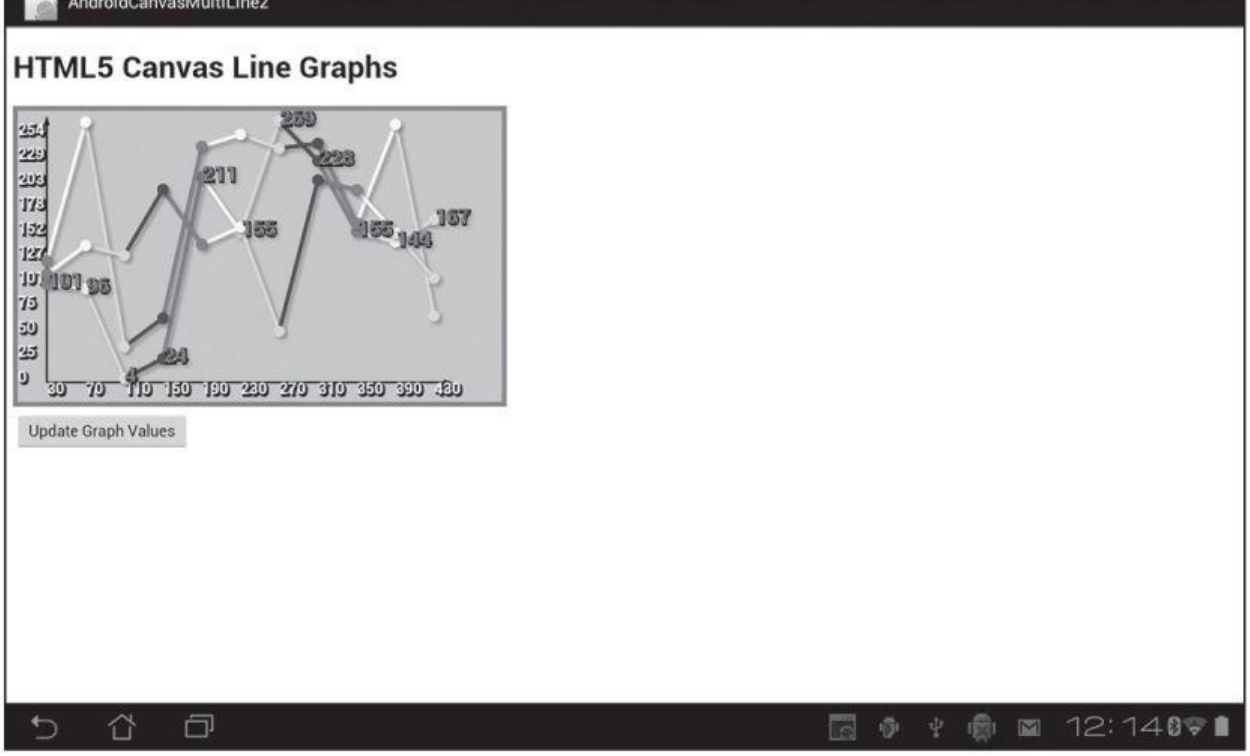


Figure 10.6 A Canvas-based multi-line Graph on an Android smart phone.

The Android project HTML5CanvasBBall2 contains the HTML5 Web page HTML5CanvasBBall2.html that contains JavaScript code for creating a bouncing ball effect in HTML5 Canvas .

The Android project PhoneGapForm1 contains the HTML5 Web page

PhoneGapForm1.html (which will actually be named index.html in your Android project) that illustrates how to create a form for various types of user input in PhoneGap, and the types of the input fields are such that the following occurs when users navigate to this form:

- Text input displays a standard keyboard
- Telephone input displays a telephone keypad
- URL input displays a URL keyboard
- Email input displays an email keyboard
- Zip code input displays a numeric keyboard

For the final code sample in this chapter, create an Xcode application called ThreeDCube1 by selecting the PhoneGap plugin. (Make sure that your filenames start with an alphabetical character or you will get errors when you attempt to compile and deploy your applications.) Copy the CSS stylesheet ThreeDCube1.css into your project, and replace index.html with the HTML Web page ThreeDCube1.html in your project.

The process for creating the other iOS-based mobile applications in this chapter is identical to the process for the preceding iOS mobile application, so there is no need to include additional examples. However, it's worth your while to spend some time creating additional iOS mobile applications. Not only will this increase your comfort level, but it will perhaps also motivate you to learn about other features of Xcode.

Summary

This chapter showed you how to create hybrid Android mobile applications that contain HTML5, CSS3, and SVG. You created such mobile applications manually, which involved creating Android projects in Eclipse, and then modifying the contents of the Android Activity class and populating an assets subdirectory with HTML-related files.

Next, you learned how to use the PhoneGap Eclipse plugin, which simplifies the process of creating an Android project. You also saw how the PhoneGap plugin creates a default page that allows you to use “live” features of your Android device.

ON THE CD-ROM

The CD contains an assortment of code samples, appendices, and figures that accompany the material in the book.

Code Samples

The CD that accompanies this book contains all the code samples to save you time and effort from the error-prone process of manually typing code into a text file. Samples are in their respective chapter folders.

Appendices

The CD contains appendices for the following topics:

- Appendix A: Overview of SVG
- Appendix B: Introduction to Android
- Appendix C: HTML5 and JavaScript Toolkits
- Appendix D: Introduction to Single-Page Applications

Figures

All of the figures from the book, including any images or screenshots that were originally 4-color, are including in their respective chapters on the CD-ROM.

A

B

C

D

E

H

I

J

L

M

N

P

Q

R

S

T

V

W

X