VISUALIZING

THE WEB

Building Websites with HTML5 to Work with Mobile Phones

Matthew David

Focal Press

BUILDING WEBSITES WITH HTML5 TO WORK WITH MOBILE PHONES

MATTHEW DAVID

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON • NEW YORK • OXFORD
PARIS • SAN DIEGO • SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Focal Press is an imprint of Elsevier

Focal Press is an imprint of Elsevier
225 Wyman Street, Waltham, MA 02451, USA
The Boulevard, Langford Lane, Kidlington, Oxford, OX5 1GB, UK

Notices
Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

ISBN: 978-0-240-81906-8

CONTENTS

BUILDING WEBSITES WITH HTML5 TO WORK WITH MOBILE PHONES

Do you have a mobile phone? Back in the mid-1990s there is a good chance you did not. Today? Well, today, there is a good chance you do not have a landline phone, but you certainly have a mobile phone. According to Gartner, one in three people on the planet have a mobile phone, with that number expected to increase to two in three over the span of this decade. What does that mean? Four billion people will have mobile phones by the year 2020.

Today, mobile phones are broken into three broad categories: call only, feature phone, and smart phone.

The call-only phone allows you to make calls and maybe to send and receive text messages. Nothing fancy. A feature phone comes with a camera, texting, and possibly a Facebook app, as shown in Figure 1.1.

The third category is smart phone. One phone has come to symbolize all smart phones: Apple's iPhone. It is fair to compare the iPhone to a computer. With an iPhone you have the following:

- GPS
- Hi-res camera
- Video recording
- Accelerometer
- Gyroscope
- Internet access

When the iPhone was launched in 2007, Apple CEO Steve Jobs hailed the phone as three devices in one: the best phone, the best iPod, and the best way to experience the web, as shown in Figure 1.2. Using an iPhone to surf the web you will see that the mobile experience is phenomenal. Web pages simply render as they are meant to; *The New York Times* loads correctly, CNN looks like CNN, and Facebook just works.

Building Websites with HTML5 to Work with Mobile Phones. DOI: 10.1016/B978-0-240-81906-8.00001-1

Figure 1.1 Feature phones.

Figure 1.2 Steve Jobs with the original iPhone presented in January 2007.

The reason for this is due to the browser, Mobile Safari. Mobile Safari is not a stripped-down version of a browser, as you will find in older smart phones such as Windows Mobile 6.5, but a browser that stands shoulder-to-shoulder with leading desktop browsers such as Google's Chrome or Mozilla's Firefox.

Apple is able to do this because Mobile Safari is built on top of the Open Source platform called WebKit. The same WebKit that is used in Mobile Safari is used in the desktop version of Safari and under the hood of Google's Chrome. The key standout feature for WebKit is its massive support for HTML5, the new set of standards that allows you to build print quality websites.

While Apple may have raised the bar for smart phones, it is not the only player in town. It is becoming increasingly clear that Google, with its mobile Android operating system (Figure 1.3), is now standing shoulder-to-shoulder with Apple.

Google's Android OS is now currently the most popular mobile OS for smart phones. There is a simple reason for this: Google gives the OS away for free as an Open Source project. Anyone can download and use the Android OS. They can even customize the OS and control how it is deployed. This is clear when you buy a Verizon phone or an HTC phone. Both run Android, but both can look very different, as shown in Figure 1.4.

At the heart of the Android phone experience is another WebKit-enabled web browser. There are subtle differences between Apple's WebKit implementation and Android's (we will cover that in more detail elsewhere), but on the whole a page that loads in one will load in another.

Today, Android and iPhones are the two leading phones, but the whole smart phone market is very small and is expected to grow exponentially. At the January 2011 quarterly result conference, Tim Cook, Apple's COO, made the comment that "in the future there will not be feature phones or smart phones; they will be all smart phones." Cook's comments are accurate. The rate of adoption of smart phones is like nothing the tech industry has seen. To this end, both Apple and Google are going to find their market space getting very crowded.

During 2011 three strong mobile operating systems will join Android and iOS: Microsoft's Windows Phone 7, RIM's BlackBerry 6, and HP/Palm's WebOS.

Figure 1.3 The Google Android logo.

Figure 1.4 Android running on three different phones from Motorola, HTC, and Samsung.

Figure 1.5 Windows Phone 7 with the unique tile interface.

Microsoft lacked vision when it came to Mobile devices. At one point, Microsoft owned the market. Losing can, sometimes, be a great panacea. Microsoft's response is Windows Phone 7, a solid contender to Apple's iOS (shown in Figure 1.5). The interface is unique, employing a metaphor called tiles. Interestingly, though, when Windows Phone 7 launched, it did not come with an HTML5 browser. Microsoft addressed this issue during the summer of 2011 with a new release of the OS that includes a mobile browser that can view HTML5 websites.

Like, Microsoft, RIM was also a leader of smart phone development. Its response to Apple and Google has been slow, but it is clear that it is coming back with a strong solution in its adoption of the BlackBerry 6 operating system.

HP/Palm's WebOS is, to me, a success story waiting to happen. In many ways, when Palm launched the Pre (shown in Figure 1.6) and Pixi running WebOS, it was the bad hardware, not the OS, that let the product down and eventually saw Palm being purchased by HP. The core development environment for WebOS is HTML5 standards (CSS, HTML, JavaScript, etc.). Powering all this is an implementation of WebKit. HP has already promised that WebOS will be back in style in 2011.

What is becoming clear, in these early days of smart phone development, is that who the leader is today will change every 3 to 6 months. Unlike the days of the web back in the mid-1990s when only two companies were vying for your attention (Netscape and Microsoft), today you have many companies and phone carriers. In addition, buying a new phone every 12 to 18 months for around $100 to $150 is not unreasonable. Indeed,

Figure 1.6 WebOS running on a Palm Pre.

Apple has an agreement with AT&T that allows you to upgrade your phone every 12 months. The smart phone replacement cycle is forcing companies to upgrade their software and hardware on a rapid curve. Think about this for a moment: the smart phone category we think of today did not exist until mid-2007. Only four years ago.

If you look at all five companies, Apple, Google, Microsoft, RIM, and HP, and their mobile operating systems, one single common thread can be seen among all of them: HTML5-enabled browsing.

Designing for the Mobile Web

Designing websites for a smart phone is not the same as designing for a PC web browser. There are several top-level differences you need to consider when designing for mobile devices:

- Screen size
- Changing portrait/landscape views
- High-quality resolution
- Input devices
- HTML5 support

Over the last few years, a widescreen aspect has become the norm for many laptop screen sizes. Typical screen sizes now run 1280×1024 pixels. In contrast, the first iPhone ran at 320×480 pixels. The Android OS can run many different screen resolutions

(top-level devices such as the HTC EVO runs at 800×480; whereas the entry level Android phones have a screen resolution of 240×400). The iPhone 4 and 5 both have a screen resolution of 960×640, double the size of the first three generations of iPhone.

Physically, smart phones are unlikely to increase much more in screen resolution for a simple reason: a phone cannot be too large, otherwise you will not be able to hold it with one hand. Dell's Android-powered Streak failed because it was too large to hold with one hand. Come on, people, this is not 1989 anymore (check out Gordon Gekko's phone in *Wall Street*—wow!).

In addition to a smaller screen, web pages on smart phones have a second, unique experience: constant change between landscape and portrait view. All the leading smart phones will allow you to twist your phone around to get a better view of the web page. Hardware accelerators in the phone can detect that the phone is rotating and will change the view from landscape to portrait accordingly.

The challenge different screen sizes offer is simple: your design must be flexible, stretching to meet the correct proportions for the screen on which it is presented. You will see, as you read further, how this is accomplished with each of the frameworks we will work with.

An interesting challenge that smart phones provide designers is resolution. For many years web designers have been told that they can keep their web graphics set to 72 DPI (dots per inch). During 2010 this changed. First Apple and Google added functionality that allows for hi-resolution images to be added to apps and web pages. The reason for this is related to how we use our phones. Typically, you hold your phone about 8 to 12 inches from your face. Your eye can see the detail you will miss on a monitor. Top-end devices now have DPI resolution far in excess of 240 DPI (the iPhone 4 has a DPI of 334 that is branded as “Retina Display”). The result is close to print-quality graphics on your phone. Incredible and beautiful. The challenge this offers is that images that are higher in resolution are much larger in file size, as shown in the comparison between iPhone 3GS and iPhone 4 in Figure 1.7.

Desktop and laptop computers have an input model of a mouse and keyboard. Both of these inputs are very precise. The primary input device for your smart phone is your finger (if you are lucky, you have eight and two thumbs versus the one mouse a computer has).

A digit is not precise. Apple's human user interface manual suggests that buttons that you tap with your finger are 44×44

Figure 1.7 Retina display quality on iPhone 4.

pixels at a minimum. When your screen size is only 320×480, you can see how much space you must provide for buttons tapped with a finger.

A key element that is supported across leading smart phone manufacturers is support for HTML standards. HTML5 is a great buzzword (throw it in the same group as dHTML, Web 2.0, Ajax, Cloud, etc.) that means a lot to a lot of people. HTML5 even comes with its own logo, as shown in Figure 1.8. At its core, HTML5 is a new set of HTML elements and attributes—tags in other words. For the most part, the new tags are designed to make blocking content on your web pages easier. Some tags, such as VIDEO, AUDIO, and CANVAS, add rich media solutions that allow you to add standards-based video and audio and rich Flash-like animation.

Figure 1.8 HTML5 logo.

Just from this list you can see that mobile web development offers many challenges and opportunities you do not experience on a laptop. Do not think that coming to the mobile platform is the same as desktop. The customer experience is simply too different.

The Leading Mobile Web Browsers

Today, two companies dominate browser use for smart phones: Google and Apple. It would be fair to say that close to 99% of all mobile web traffic comes from these two platforms.

Apple's Mobile Safari and Android's web browser are both built using WebKit as a foundation. This does not mean they are both equal. For instance, Mobile Safari has supported SVG graphics since version 1.0 whereas Android did not start support for SVG until the release of Honeycomb (3.0).

The two browsers enjoy huge support for a simple reason: they are the default browsers installed on the hardware.

Android does allow you to install additional web browsers but adoption rates are very low. Apple takes things one step further and prohibits additional web browsers from being submitted to the App Store.

Fortunately both browsers do have great support for modern web technologies allowing you to deliver amazing web experiences to your customers.

Additional Web Browsers

Mobile Safari and Android are not the only browsers in town. In addition, there are:

- Mobile Firefox (known as Fennec)
- Mobile Opera
- Chrome OS
- Mercury

Figure 1.9 Mobile Firefox (code name Fennec) running on an HTC Windows 6 phone.

Mobile Firefox is a port of Firefox 3.6 for the mobile platform. Currently it has limited support on Nokia Maemo phones, but there is a beta release for Android and Windows Phone 6, as shown in Figure 1.9.

Outside of the default browsers that come with Android and iOS, Opera Mobile is the most popular browser. Opera has been creating a mobile version of its browser since 2000, with each major release supporting almost all the same features as its desktop version. The current release has broad support for HTML5. Figure 1.10 shows Opera running on an HP iPAQ.

Opera Mobile runs on many platforms including Android, Windows Mobile, Maemo, and Symbian. The following phones all ship with Opera Mobile installed:

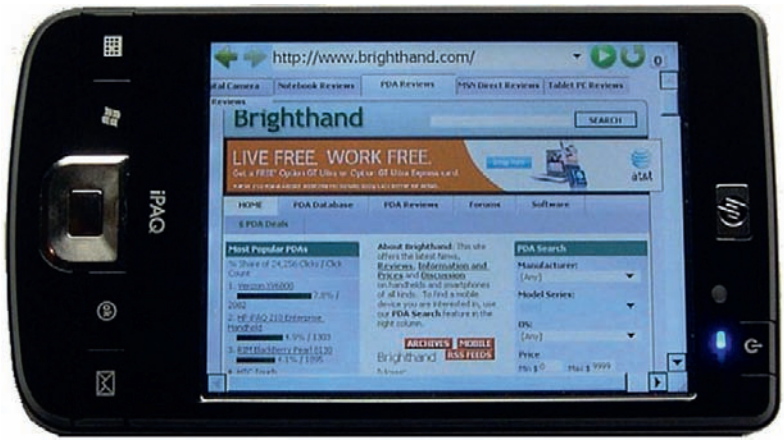


Figure 1.10 Opera browser running on an iPAQ.

- Nokia N90
- Sony Ericsson P1
- Sony Ericsson XPERIA X1
- HTC Touch Viva
- HTC Touch Diamond
- HTC Touch Diamond2
- HTC Touch Pro
- HTC Touch Pro2
- HTC Touch HD
- HTC HD2
- Meizu M8
- Creative Zii
- Samsung i900 Omnia
- Samsung i8000 Omnia II
- Motorola ROKR E6

While Opera is still a niche player on the desktop, it is a major player in the mobile arena.

The final mobile browser worth considering as you design you web pages is Chrome OS. Google is performing a strange development cycle between Android and Chrome OS. If you did not know, you would think that they compete with one another. Chrome OS is built on top of Google's Chrome web browser. Google has confirmed that Chrome OS will be installed on netbooks but Google has not declared where else it will be installed.

HTML5 in Mobile Websites

The next section dives deep into HTML5. HTML5 is an emerging standard that is the most dramatic evolution of web development standards in more than a decade. HTML5, however, has

come to mean a lot more than just a new set of tags. The term now encompasses a whole set of technologies that include:

- HTML5 elements
- CSS3
- New graphic control (PNG, SVG, and CANVAS)
- Enhanced JavaScript
- Web APIs

There is even more. Amazingly, mobile browsers are ahead of desktop browsers in support for these technologies. All of the following technologies will work on Android, iOS, WebOS, and BlackBerry 6. You will need to wait for 2011 summer release of HTML5 support in Windows Phone 7 to support HTML5.

New HTML5 Elements

The blocking of content in HTML is traditionally accomplished using either complex tables or the infamous DIV element. HTML5 introduces several new elements that allow you to easily insert blocks of content into the page. Conveniently, these new elements have names that identify what the block of content accomplishes:

- HEADER
- SECTION
- ARTICLE
- ASIDE
- FOOTER
- NAV

The role of the new page layout elements is to better describe specific parts of a document. Think of the new tags as behaving in a similar way to how you approach writing a document in Microsoft Word. A typical Word document is built up of sections of content that can be separated in paragraphs, sidebars, and header and footer sections.

Blocking Content

There are few ways in HTML4 to define content. The most common is to use the P element to identify the start and end of a paragraph, or the DIV element to identify the start and end of a section of content. Both do not adequately describe the content. You can see blocking applied to most web pages.

With HTML5 a new element, the SECTION element, clearly identifies a block of content. This method is called block-level semantics. With HTML5 there are several elements that block content:

- SECTION
- ARTICLE
- HEADER
- FOOTER
- ASIDE

- FIGURE
- NAV

The new names for each of these elements identify the type of content they block on a page.

Using the SECTION Element

The SECTION element is part of a new set of elements that describe the content on a page. You can think of the SECTION element as enclosing a significant part of a page, in the same way that a chapter in a book is a significant section of the book. An example of the SECTION element follows.

```
    <SECTION>
    <ARTICLE>
    <P>Nulla facilisis egestas nulla id rhoncus. Duis eget
diam nisi, quis sagittis nulla. Fusce lacinia pharetra
dui, a rhoncus sapien egestas.</P>
    </ARTICLE>
    <ARTICLE>
    <P>Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Nunc vehicula ipsum sit amet eros adipiscing
volutpat. Sed gravida urna vel sapien commodo pretium.</P>
    </ARTICLE>
    <UL>
    <LI>Praesent ut sapien quam.</LI>
    <LI>Aliquam erat volutpat.</LI>
    </UL>
    </SECTION>
```

You can see clearly that the two paragraphs, wrapped in the P element, and the two bullet points are part of the same content wrapped in the SECTION element.

The SECTION element is an efficient way to organize content in your code.

Using the ARTICLE Element

The ARTICLE element is used to clearly identify content in a web page. Blogs are a good example where content is clearly identified. The main section of a page is the content that you can wrap using the ARTICLE element. You can have additional HTML elements included within an ARTICLE. The following blog from *http://blog.whatwg.org/* is an example that shows how you can use the ARTICLE element in HTML.

```
    <ARTICLE>
    <H1>Spelling HTML5</H1>
    <P><TIME>September 10th, 2009</TIME> by Henri
    Sivonen</P>
    <P>What's the right way to spell “HTML5”? The short
answer is: "HTML5" (without a space).</P>
    </ARTICLE>
```

More than one ARTICLE can be added to a page. You should think of the ARTICLE element as a tool that logically breaks up content. Similar content separated by the ARTICLE element can be contained within a SECTION element.

Using the HEADER and FOOTER Elements

The top and bottom of a page created with Microsoft Word or any other word processing software is a place reserved for the header and footer information page. This includes page number, copyright notice, and other details. Web pages are no different. Header and footer information is found on most web pages.

You can see the use of the header on the page in the following HTML example. It contains the Focal Press logo, the element line, high-level links, and a search box. HTML5 allows this area of content to be clearly identified as either a header or a footer using the new HEADER and FOOTER elements.

For instance, a HEADER for Focal Press would look like the following.

```
    <HEADER>
    <SECTION><a href="/"><img src="/images/fplogo.png"
border="none" alt="Focal Press logo" title="Focal Press
Home"/></a> learn | master | create</b>SECTION>
    <NAV>
    <ul><li><a title="Digital Imaging and Photography"
class="first" href="/photography.aspx">Photography</a>
</li><li><a title="Production, Postproduction, and Motion
Graphics" href="/film_video.aspx">Film &amp; Video</a></
li><li><a title="Animation, 3D, and Games" href=
"/animation_3d.aspx">Animation &amp; 3D</a></li><li>
<a title="Audio Engineering and Music Technology" href=
"/audio.aspx">Audio</a></li><li><a title="Broadcast
    and Digital Media" href="/broadcast.aspx">Broadcast
</a></li><li><a title="Theatre and Live Performance" href=
"/theatre.aspx">Theatre</a></li><li><a class="offsite last"
href="http://www.elsevierdirect.com/imprint.jsp?iid=100001"
    >Bookstore </a></li> </ul></NAV>
    </HEADER>
```

The FOOTER section to a page is also viewed on most web pages. An example FOOTER in HTML5 will look as follows:

```
    <FOOTER>
    <P>Copyright © 2011 Focal Press, Inc.</P>
    </FOOTER>
```

Unlike normal page layout, the HEADER and FOOTER are not exclusive to just the head and foot of a web page. You can have a header and footer placed around the ARTICLE or SECTION element if those pieces require specific header and footer content.

Using the ASIDE Element

The role of the ASIDE element is to describe content that is related to but is not part of the main content on the web page. You can think of the ASIDE element as fitting the role of a sidebar reference or an aside found in books and articles. The following example shows how the ASIDE element can be used with the ARTICLE element.

```
    <ARTICLE>
    <P>Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Vivamus sed eros at metus pulvinar convallis id quis
purus. Sed lacinia condimentum viverra.</P>
    <ASIDE>
    <H1>What is Lorem Ipsum?</H1>
    <P>Lorem Ipsum is simply dummy text of the printing and
typesetting industry.</P>
    </ASIDE>
    </ARTICLE>
```

The main content of the page and a support aside can be clearly separated using the ASIDE element.

Apply formatting, using CSS, to visually show where the ASIDE is on the screen.

Using the FIGURE Element

Inserting images into a web page is common practice. Identifying the image and supporting text as a figure is much more difficult. The FIGURE element clearly identifies an image and supporting description as being part of a set. This set is called a figure group. As with many of the previous new HTML5 elements, the FIGURE element is a method of blocking related content with itself.

```
    <FIGURE>
    <LEGEND>Figure 12. Using the FIGURE element
    </LEGEND>
    <IMG alt="The FIGURE element is another example
of block semantics in HTML5" src="figure_element.jpg"
border="0" height="140" width="240" />
    </FIGURE>
```

The FIGURE element has an additional element you can use within it. The LEGEND element identifies the text that is to be associated with the image. The FIGURE element can be used multiple times on a page. The Border attribute is deprecated but it is still used by most browsers.

Using the NAV Element

The final HTML5 blocking element is NAV. Navigation is important to any website. The role of the NAV element is to clearly identify groups of links that when grouped together form navigation.

Navigation can take many different roles on a single web page. The different types of content that can be grouped together as navigation include, but are not limited to, the following:

- Top-level links typically found in the top-right corner of a web page
- Links that move you through data such as "Next" and "Previous"
- Links found in the footer of a web page

The following is an example of navigation grouped using the NAV element.

```
<NAV>
<a href="/home.html">Home</a> | <a href="aboutUs.
html">About Us</a> | <a href="contactUs.html">Contact Us</a>
</NAV>
```

Of all the blocking elements in HTML5, the NAV element is one of the easiest to understand: the NAV element is used to define a section of HTML for navigation on the page.

Using CSS3

Tags are used in HTML5 to place and organize content at a level that is descriptive. This does not mean that the page will look good. Presentation of content on the page is controlled using Cascading Style Sheets Level 3, or CSS3, in HTML5.

Figure 1.11 jQuery Mobile leverages CSS3 to manage the presentation of content.

Using CSS to describe how your page should look, however, is not new. The technology was first introduced in 1997 and is now, in HTML5, in its third major release, named CSS3. The good news is that all CSS1 and CSS2 standards are fully supported by popular web browsers.

For mobile web design you will use CSS to format your web pages. There are good reasons why you want to do this. The first, and most important, is that CSS is a tool that allows you to easily apply page styling techniques to a whole website from one or more shared documents. This means you can quickly change the visual layout of a page, selection of pages, or your entire site.

The second is that Apple has GPU accelerated support for CSS. What this means is that CSS simply runs faster. Animations, rounded corners, embedded fonts, and transforms all look great on the iPhone. The powerful new Nvidia and Qualcomm chipsets appearing in most smart phones really give presentation in your web pages an edge. The result is that you can use CSS to design native app-like solutions without having to write native code. Just CSS.

In a later article you will see how jQuery Mobile enables you to build stunning solutions, with CSS3 playing a major role in the presentation. Figure 1.11 shows a website that

uses CSS3 in jQuery Mobile to build a website that looks like a native application on the iPhone.

This section will not go into detail about CSS creation and development. For a more detailed analysis of CSS3 in HTML5, check out the book *HTML5: Designing Rich Internet Applications* (David, 2010).

Designing Your Web Page with CSS

CSS is much easier to master than more complex parts of HTML5 such as Local Data Storage, Geolocation, and JavaScript. The basic premise for all CSS is that you have a definition that requires a value. For instance, if you want to define the size of a particular font, you write the correct CSS definition (font-size) and place a value. Here is the code:

```
font-size: 60px;
```

There are four rules you must follow:

1. Use a valid CSS definition.
2. Place a colon after the definition.
3. Add a valid value for the definition.
4. Complete the statement with a semicolon.

Follow these four rules and you are golden.

For basic CSS manipulation there are some great tools you can use. Adobe's Dreamweaver and Microsoft's Expression Web both support CSS2 design definition. Both of these tools are offer visual editors you can easily use to write CSS. Unfortunately your choices drop significantly when you start to look for more advanced CSS3 tools. This is in part due to the rapid development of CSS3. Check out visualizetheweb.com for the latest information on CSS3 tools.

When CSS was first released in 1997 there were about a dozen or so definitions you could use to control visual aspects such as font size, color, and background color. Now you have hundreds of different definitions that can be used extensively with any element on the screen.

Controlling Display with CSS

One of the easiest places to start learning how to use CSS definitions is through font control. CSS1 and CSS2 support nine different definitions within the font family. They are:

- Font-family
- Font-size
- Color
- Text-shadow
- Font-weight
- Font-style
- Font-variant
- Text-transform
- Text-decoration

The font-family definition allows you to select a font for your design. Here is how you write the definition:

```
font-family: Arial;
```

The challenge you have in using the font-family definition is that the number of fonts you can select from is limited to the fonts installed on the device viewing your web page. Web browsers and operating systems install a core set of fonts that you can use in your designs. The list of fonts you have available that are "web safe" includes the following:

- Arial/Helvetica
- Times New Roman/Times
- Courier New/Courier
- Verdana
- Georgia
- Comic Sans MS
- Trebuchet MS
- Arial Black
- Impact
- Palatino
- Garamond
- Bookman
- Avant Garde

This list is not very exhaustive and you run into issues where the fonts will not match. For instance, you may select the font Tahoma and it will look great on Windows Phone 7 but will not look the same on the iPhone. Often you will find that there are similar fonts on devices, but they simply have different names. For instance, you can select the following font family:

```
font-family: "Courier New", Courier, monospace;
```

This collection of fonts will allow the text to be presented correctly no matter the system viewing the page. In this instance, "Courier New" is the Windows Phone name for "Courier" on iOS. Monospace is a Unix/Linux equivalent that you will find on Android.

Here is a collection of safe font-family names you can use:

- Arial, Arial, Helvetica, sans serif
- Arial Black, Arial Black, Gadget, sans serif
- Comic Sans MS, Comic Sans MS, cursive
- Courier New, Courier New, Courier, monospace
- Georgia, Georgia, serif
- Impact, Impact, Charcoal, sans serif
- Lucida Console, Monaco, monospace
- Lucida Sans Unicode, Lucida Grande, sans serif
- Palatino Linotype, Book Antiqua, Palatino, serif
- Tahoma, Geneva, sans serif
- Times New Roman, Times, serif

- Trebuchet MS, Helvetica, sans serif
- Verdana, Verdana, Geneva, sans serif
- Wingdings, Zapf Dingbats (Wingdings, Zapf Dingbats)

Embedding Fonts Using CSS3

A way to get around the problems of creating font-family lists is to embed the font directly into your CSS. CSS3 finally allows you to do this across your web browsers. The technology for font embedding, however, is not new. Netscape Navigator 4 was the first web browser that allowed you to support font embedding using a plug-in technology called TrueDoc by Bitstream. To compete with Navigator 4, Microsoft released a "me too" technology called Embedded Open Type in the Windows version of Internet Explorer 4.

As you might expect, HTML5 has driven new technologies to enable true font embedding. Three standards are now recommended to embed fonts. They are:

- TrueType
- Scalable Vector Graphic Fonts
- WOFF

Embedding a font into your CSS document is now very easy. Figure 1.12 shows Google's Web Font directory of free HTML5 web fonts you can use now.

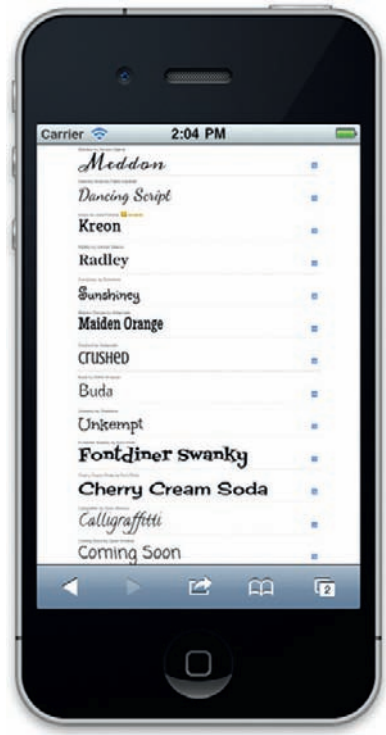


Figure 1.12 Free fonts from Google you can use on your website.

To embed a font into a web page you need only two things: the font file and definition in CSS linking to the font.

The font myCustomFont.ttf is being used in the CSS code below.

You need to create a new font-family in your CSS document that links to the TrueType font. The following CSS code shows, in line 2, where you can create a new font-family called "myCustomFont" and, in line 3, you are linking to the font and identifying the type of font.

```
@font-face{
font-family: 'myCustomFont';
src: url('MYCUSTOMFONT.ttf') format('truetype');
}
```

You now have a new font-family that you can reference in your normal CSS. Here, a P element is being formatted using the new font-family:

```
p {
text-align: center;
font-family: 'myCustomFont';
font-size:3cm;
}
```

Now your web pages will display the embedded font correctly no matter what web browser is viewing your design. Font freedom has finally come to the web!

Sizing Your Fonts with CSS Units of Measurement

After selecting a font-family for your text you will also want to select the size of the font. By default, all web browsers have a preinstalled definition for a standard font size. This font size is usually 12 pt. You can use this as a size for your fonts as they appear on the screen using the following CSS font-size definition:

```
Font-size:medium;
```

If you want your font to appear smaller or larger on the screen you can use the following sizes for your fonts:

- Xx-small (approximately 7.5 pt)
- X-small (approximately 9 pt)
- Small (approximately 10 pt)
- Medium (approximately 12 pt)
- Large (approximately 14 pt)
- X-large (approximately 18 pt)
- Xx-large (approximately 24 pt)
- Smaller
- Larger

Each of these font sizes are relative to the core browser defaulted font size. If the person who owns the web browser has changed that default then the sizes will dynamically change.

As a designer you are limited by the default font size list. The good news is that CSS allows you to leverage units of measurement to add precise size to your font. The following are all valid CSS units of measurement you can use.

- Cm: Centimeter
- In: Inch
- Mm: Millimeter
- Pc: Pica (1 p = 12 pts)
- Pt: Point (1 pt = 1/72 inch)
- Px: Pixels
- Rem: Font size of the root element

Using these different font sizes, the following styles are all valid:

```
.default {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: medium;
}
.px {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: 15px;
}
```

```
.cm {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: .5cm;
}
.mm {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: 2mm;
}
.inch {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: .25in;
}
.pica {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: 2pc;
}
.point {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: 10pt;
}
.rem {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: 1rem;
}
```

These font styles are applied to the following HTML:

```
    <p class=“default”>In hac habitasse platea dictumst.</p>
    <p class=“px”>Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Etiam accumsan convallis odio, vitae
semper mi pretium laoreet. </p>
    <p class=“cm”>In vestibulum, ipsum consectetur cursus
porttitor, mi tellus euismod purus, ac egestas nisl
risus ac risus. Suspendisse a nisi mi, nec rutrum nisi.
Suspendisse pretium aliquet convallis. </p>
    <p class=“mm”>Aliquam sollicitudin elementum est,
commodo gravida lorem imperdiet ac. </p>
    <p class=“inch”>In hac habitasse platea dictumst
. </p>
    <p class=“pica”>Donec rhoncus turpis vitae risus
commodo ac mollis ligula aliquam. Donec in mi arcu, id
vulputate turpis. </p>
    <p class=“point”>Nullam nunc dui, euismod vel lobortis
nec, suscipit non velit. </p>
    <p class=“rem”>Aliquam ornare, nibh eget facilisis
lobortis, ligula velit suscipit sem, id condimentum est
turpis ut magna. </p>
```

Figure 1.13 The @ font-face is used to embed fonts in the above web page.

Figure 1.13 shows you how these fonts are presented in your mobile browser.

CSS3 Color Control

As with size, color has many different units of measurement. The default for web design is hexadecimal, a combination of six letters and numbers. CSS3 provides you a much broader palette of colors to choose from that include the following:

- Color Name: You can use a name for color such as Brown, Black, Red, or even Cyan.
- Full Hexadecimal: A hexadecimal value comprised of six alphanumeric values.
- Short Hexadecimal: A hexadecimal value comprised of three alphanumeric values.
- RGB: A combination of red, green, and blue values.
- RGBA: A combination of red, green, and blue values with a transparency value (Alpha).
- HSL: A combination of hue, saturation, and lightness.
- HSLA: A combination of hue, saturation, and lightness with a transparency value (Alpha).

The following CSS uses these values to show how you can create the color red different ways:

```
.name {
color: red;
}
.fullHexVersion {
color: #FF0000;
}
.shortHexVersion {
color: #F00;
}
.rgb {
color: rgb(255,0,0);
}
.rgba {
color: rgba(255,0,0,100);
}
.hsl {
color: hsl(0%, 100%, 50%);
}
.hsla {
color: hsl(0%, 100%, 50%, 100%);
}
```

These different values are used in different places within the design community.

Adding Drop Shadow Text Effects

Love them or hate them, you cannot get away from the handy design technique of drop shadows. CSS3 now supports drop shadow effects, and they are very easy to add to your designs.

There are four elements that you can use to control the drop shadow definition. They are:

- Horizontal-offset (length, required)
- Vertical-offset (length, required)
- Blur-radius (length, optional)
- Shadow-color (color, optional)

The following CSS definition is an example of the use of the drop shadow, illustrated in Figure 1.14.

```
.dropShadow {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: 3cm;
        color: #CC3300;
        text-shadow: 0.25em 0.25em 2px #999;
}
```

The effect draws a light gray drop shadow with a slight blur.

Different colors and units of measurement can be used with the drop shadow effect. The following CSS definition uses pixels and RGBA for the measurement and color.

```
.transparentDropShadow {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: 25px;
        color: rgba(255,0,0,1);
        text-shadow: 5px 5px 5px rgba(0, 0, 0, 0.5);
}
```

Finally, you can use the drop shadow effect to force a “cut out” effect with your text. Apply the following CSS to text on the screen:

```
.cutout {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: 3cm;
        color: white;
        text-shadow: 0em 0em 2em black;
}
```

Figure 1.15 demonstrates the effect of the drop shadow as a cut out.

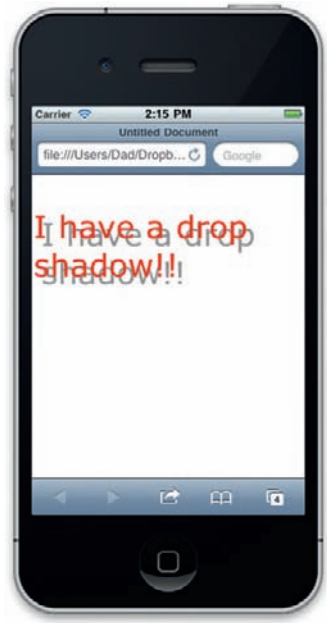


Figure 1.14 A CSS3 drop shadow.

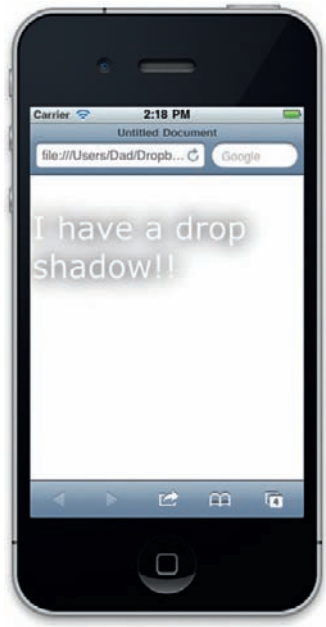


Figure 1.15 The CSS3 drop shadow effect can also be used as a cut-out effect.

Working with Columns in CSS3

A challenge for any web page is to create content that is split over two or more columns on the page. Creating columns often requires using complex tables structured together. Though not strictly part of the text family of CSS definitions, the new multicolumn layout is best when used with text on the screen.

The goal of the multicolumn definition is to allow your content to be spread evenly over two or more columns. There are three parts to a column layout:

- The number of columns
- The gap between the columns
- Column design (optional)

Figure 1.16 A simple two-column layout using CSS3.

The following CSS demonstrates how you can set up multicolumns to display in WebKit-enabled browsers.

```
.simple {
        font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
        font-size: 12px;
        color:#444;
        text-align: justify;
        -moz-column-count: 2;
        -moz-column-gap: 1em;
        -webkit-column-count: 2;
        -webkit-column-gap: 1em;
        }
```

In this example, the column-count is two and the gap is 1 em. Figure 1.16 shows how this is displayed in your web browser.

You can add a column design between each column. The structure is:

```
-moz-column-rule: 1px solid #222;
-webkit-column-rule: 1px solid #222;
```

For each column design you can identify the width, border style, and color. You can use the standard measurement and color CSS formatting. You can choose from the following border styles:

- None
- Hidden
- Dotted
- Dashed
- Solid
- Double
- Groove
- Ridge
- Inset
- Outset

Additional elements, such as the IMG, can be used with text content in the column layout. Figure 1.17 illustrates a complex use of a multicolumn layout.

Figure 1.17 A complex three-column layout.

The CSS to create this layout is:

```
.complex {
       font-family: “Segoe UI”, Tahoma, Geneva, Verdana;
       font-size: 1.2pc;
       color:#444;
       text-align: left;
       -moz-column-count: 3;
       -moz-column-gap: 1em;
       -moz-column-rule: 2px dotted #999;
       -webkit-column-count: 3;
       -webkit-column-gap: 1em;
       -webkit-column-rule: 2px dotted #999;
       }
```

The style in this column layout is applied to P element, which contains both text and an IMG element. You should experiment with columns. They are certainly much easier to use than complex tables.

Increase Your Control over Gradient Colors

Control over your use of color has increased significantly with CSS3. You saw earlier that you can use long hexadecimal, short hexadecimal, RGB, RGBA, HSL, and HSLA to have access to millions of colors. In addition to solid colors, CSS3 gives you the ability to add gradients.

You can currently create two different types of gradient: linear and radial. Figure 1.18 illustrates the two different gradient types you can create.

The gradient definition is comprised of several key elements. They are:

- Type: Either radial or linear
- Point: Two space-separated values that explain where the gradient starts (this can be achieved with a number, percentage, or by using the keywords top, bottom, left, and right)
- Radius: The radius is a number that you only need to specify when you use the radial type
- Stop: The function of the Stop value is to identify the blend strength as a percentage or number between 0 and 1 (such as .75 or 75%) and a color; you can use any CSS3 supported color

Putting all of these together will give you a gradient. Gradients can be used with the following definitions:

```
background-image
border-image
list-style-image
content property
```

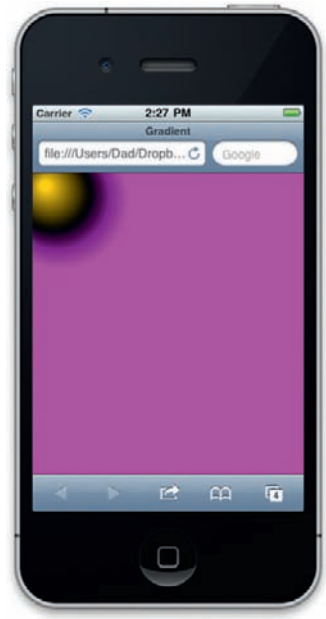


Figure 1.18 CSS3 allows you to add gradient colors.

The following example adds a gradient that goes from red-to-orange-to-orange-to-yellow:

```
body {
background-image: -webkit-gradient(linear, left top,
left bottom, from(red), to(yellow), color-stop(0.5, orange),
color-stop(0.5, orange));}
```

As you can see, the gradient is substituting an image in the background-image definition. The first definition identifies the gradient as linear. The next definition explains that the gradient is going to go from top to bottom. The two elected colors are red and yellow. The stop function has the colors blending halfway through to orange.

A radial gradient is completed in a similar way. The following adds a radial gradient that moves from red-to-orange-to-yellow:

```
body {
background-image: -webkit-gradient(radial, 45 45, 15, 100
100, 250, from(red), to(yellow), color-stop(50%, orange));}
```

In this instance, the numbers following the radial declaration determine the shape of the radius. The first two numbers dictate the angle of the ellipse in degrees. The third number dictates the size of the inner circle. The fourth and fifth numbers dictate the position of the gradient (left and top). The final number dictates the final size of the radius.

Figure 1.19 Multiple backgrounds.

Multiple Background Objects

You quickly run into limitations when you can use only one background image. With CSS3 you can now run multiple background images. Any element that supports the background-image definition now supports multiple background images. Using background images is very easy. You can start by listing the images you want to use. Take for instance the following:

```
background-image:
url(http://upload.wikimedia.org/wikipedia/commons/3/36/
Team_Singapore_fireworks_display_from_Singapore_Fireworks_
Festival_2006.jpg), url(http://upload.wikimedia.org/
wikipedia/commons/b/b2/OperaSydney-Fuegos2006-342289398.
jpg);
```

You can specify where you want each background to appear on the screen using the background-position definition. The definition is paired for the position of the background.

```
background-position: bottom left, top right;
```

Figure 1.19 shows the end result.

As you might expect, you can mix gradients and multiple background images together. The following CSS blends a radial gradient with two background images.

```
<html>
<head>
<title>Multiple Backgrounds</title>
  <style>
  body {
    background-image:
    url(http://upload.wikimedia.org/wikipedia/
commons/3/36/Team_Singapore_fireworks_display_from_
Singapore_Fireworks_Festival_2006.jpg),
      url(http://upload.wikimedia.org/wikipedia/
commons/b/b2/OperaSydney-Fuegos2006-342289398.jpg),
-webkit-gradient(radial, 45 45, 15, 100 100, 250, from(gold),
to(magenta), color-stop(50%, black));
   background-repeat: no-repeat;
   background-position: bottom left, top right;
      background-color:black;}
  </style>
</head>
<body>
</body>
</html>
```

Adding Rounded Corners to Layers

Adding rounded corners is not a new technique for the web. You see it all the time when you look at websites. The effect, however, is created through using images and tables to create the illusion of rounded corners. Adding images to the pages ensures that the page takes longer to load and makes modifying the page later more complex.

A simpler approach is to use the proposed Corner-Radius CSS definition that is currently supported in Mobile Firefox, Mobile Safari 3.0, and the Android web browser. The Corner-Radius definition is a line you can add to your CSS style. The following HTML code has a style embedded that changes the presentation of the block of text to have rounded corners with a heavy, black outline:

```
<p style="-moz-border-radius: 10px;-webkit-border-radius:
10px;border: 4px solid #FF0000;">Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Nam porta, lacus in cursus
cursus, justo purus fringilla nisi, quis cursus urna velit vel
felis. Nulla ac mi. Phasellus sodales dui vel tortor. Praesent
dignissim. Vestibulum vulputate nibh rutrum purus. Nulla ante.
Sed porta. Vestibulum commodo, mi nec tincidunt laoreet, urna
risus ornare libero, in imperdiet sapien enim vel nisi.</p>
```

Your content will now look like Figure 1.20 on your web page.

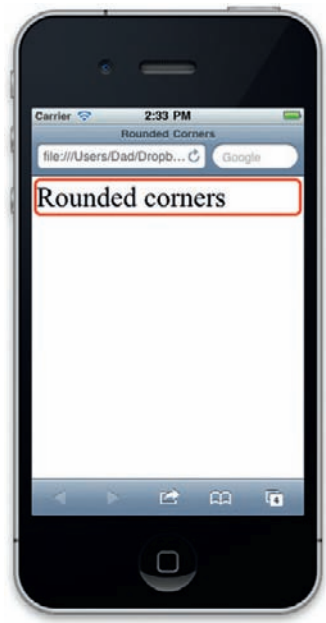


Figure 1.20 Rounded corners in your mobile site.

As you can see, the block of text now has a solid red line with rounded corners. It is this style description that controls the size of the radius, not an image. You can then easily modify the description as shown here:

```
-moz-border-radius: 10px
-webkit-border-radius: 10px
```

The standard is currently only in proposal stage and has not been adopted by all web browsers. For this reason, you need to add two border-radius style descriptions: one for Firefox (-moz-border-radius) and one for WebKit (-webkit-border-radius). Changing the value of the border-radius will change the size of the border. For instance:

```
Border-radius: 15 px
Border-radius: 25 px
Border-radius: 45 px
```

As you increase the border radius, you will also have to add additional styles, such as padding, to ensure that your border does not cut through your text as is shown in the example of border-radius: 45 px. Here is how you can add padding to manage your style.

```
    <p style="-moz-border-radius: 45px;-webkit-border-
radius: 45px;border: 4px solid #FF0000;padding: 12px;">
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam
porta, lacus in cursus cursus, justo purus fringilla nisi,
quis cursus urna velit vel felis. Nulla ac mi. Phasellus
sodales dui vel tortor. Praesent dignissim. Vestibulum
vulputate nibh rutrum purus. Nulla ante. Sed porta.
Vestibulum commodo, mi nec tincidunt laoreet, urna risus
ornare libero, in imperdiet sapien enim vel nisi.</p>
```

Without using complex images or tables, you have created a series of tabs that can be easily managed through your CSS and HTML.

Dazzling Your Audience with CSS3 Animation

CSS3 continues to expand what you can visually accomplish in your web pages. Animation is now also available to you as the design. Animation is split into two key parts: transitions and transforms.

Transitions control the change of state for an element, such as text fading in or changing color; transforms control the placement of an element.

The following two sections explain how you can control these two new animation techniques in your CSS designs.

Using Transitions in CSS

The transition effect is best used when you create a class and then a "hover" pseudo class to illustrate when the effect is to happen (i.e., when your cursor moves over the element). The transition itself is made of three parts:

- Property: The linked property between the two classes
- Duration: How long in seconds the transition will take
- Timing-function

The timing function keywords control different types of animation sequence:

- Linear: The linear function just returns as its output the input that it received.
- Ease: The default function, ease, is equivalent to cubic-bezier (0.25, 0.1, 0.25, 1.0).
- Ease-in: The ease-in function is equivalent to cubic-bezier (0.42, 0, 1.0, 1.0).
- Ease-out: The ease-out function is equivalent to cubic-bezier (0, 0, 0.58, 1.0).
- Ease-in-out: The ease-in-out function is equivalent to cubic-bezier (0.42, 0, 0.58, 1.0).
- Cubic-bezier: Specifies a cubic-bezier curve whose P0 and P3 points are (0,0) and (1,1), respectively. The four values specify points P1 and P2 of the curve as (x1, y1, x2, y2).

The following example applies a transition effect on the color definition in the PARAGRAPH element:

```
p {
        -webkit-transition: color 2s linear;
        font-size: medium;
        font-family: Arial, Helvetica, sans-serif;
        color: #FF0000;
}
p:active {
        font-family: Arial, Helvetica, sans-serif;
        color: #0000FF;
}
```

As you select any text using the PARAGRAPH element the text will slowly change from red to blue.

The top paragraph is red, the third has transitioned to blue, and the fourth is transitioning from one color to the next. You can elect to have all the properties be selected as part of the transition by changing the property value to "ALL" as in the following example.

```
p {
        -webkit-transition: all 2s linear;
        font-size: medium;
        font-family: Arial, Helvetica, sans-serif;
        color: #FF0000;
```

```
}
p:active {
        font-family: Arial, Helvetica, sans-serif;
        font-size: xx-large;
        color: #0000FF;
}
```

For quick, simple animation sequences, transitions are great.

Creating Animation with CSS3

For more complex animation you will want to use the new transform settings. The following HTML and CSS style allows you to add a bouncing text block to the screen:

```
<html>
 <head>
  <title>Bouncing Box example</title>
  <style type="text/css" media="screen">
   @-webkit-keyframes bounce {
    from {
    left: 0px;
    }
    to {
     left: 400px;
    }
   }
    .animation {
    -webkit-animation-name: bounce;
    -webkit-animation-duration: 2s;
    -webkit-animation-iteration-count: 4;
    -webkit-animation-direction: alternate;
    position: relative;
    left: 0px;
    }
   </style>
  </head>
 <body>
  <p class="animation">
   The text bounces back and forth
  </p>
 </body>
</html>
```

The animation is controlled through the use of the style sheet. There are two parts you need to control. The first sets up the type of animation you want to use. Here the setting is for an animation sequence named "bounce." The animation and the movement will be from 0 px to the left 400 px:

```
@-webkit-keyframes bounce {
 from {
  left: 0px;
```

```
 }
 to {
  left: 400px;
 }
}
```

The next step is to define what gets animated. In this example you have a CSS class associated with the "bounce" animation described earlier. There are a couple of additional settings. The duration setting controls how long each animation sequence takes to play in seconds and the count setting specifies how many times the animation plays. Together, it looks like this:

```
.animation {
 -webkit-animation-name: bounce;
 -webkit-animation-duration: 2s;
 -webkit-animation-iteration-count: 4;
 -webkit-animation-direction: alternate;
 position: relative;
 left: 0px;
```

All mobile browsers support these new animation techniques.

Using Class and Pseudo Styles

A pseudo class is a special extension to the element style definition. The most common use for pseudo classes is with the ANCHOR element. The way an ANCHOR element (the element that identifies links on a web page) is defined in CSS is as follows:

```
a {
        text-decoration: none;
        color: #0000FF;
}
```

The ANCHOR element, however, completes several different activities. The ANCHOR element has the default style, but also can have different styles when the link is being selected, when the link has been visited, and when you move your cursor over the link. Each of these different activities can be identified with pseudo classes. The following shows the pseudo class for a link that has been visited:

```
a:visited {
         color: #FF0000;
}
```

The ANCHOR element is listed first in your style document and is followed by a colon with the special pseudo class name called "visited." In your web page, the visited link will now have a different color.

The ANCHOR element has four pseudo classes: link, active, hover, and visited. The following style shows how you can define these four pseudo classes.

```
a{
        color: #0000FF;
}
a:link {
        text-decoration: none;
}
a:active {
        text-decoration: line-through;
}
a:visited {
        color: #FF0000;
}
```

The result is that you can now control the different actions of the ANCHOR tag.

CSS3 introduces additional pseudo class styles you can use. The complete list is:

- Active: The active element
- Focus: The element with focus
- Visited: A visited link
- Hover: The state when your cursor is over a link (this feature of CSS will *not* work on mobile devices)
- Link: An unvisited link
- Disabled: The state of an element when it has been disabled
- Enabled: The state of an element when it has been enabled
- Checked: A form element that has been checked
- Selection: When a user selects a range of content on the page
- Lang: The designer can choose which language is used for the style
- Nth-child(n): An element that is a specified child of the first sibling
- Nth-last-child(n): An element that is a specified child of the last sibling
- First-child: The first use of an element on the page
- Last-child: The last use of an element on the page
- Only-child: The only use of a element on the page

Media Definition Control

As we discussed earlier, different devices have different screen sizes. To help you, CSS3 has a final trick up its sleeve.

The media definition in CSS allows you to identify different styles for different media types. Originally defined in CSS2, the CSS3 expands the functionality of the CSS2 version to allow you to specify any type of device.

The easiest place to use the media definition is right when you link to a CSS document in the head of the web page. Typically you will write the following code to link to a CSS document:

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

The media definition now allows you to specify a style to be associated with a device. Take for instance the following CSS link reference to two styles documents.

```
<link rel="stylesheet" type="text/css" media="screen"
href="screen.css">
<link rel="stylesheet" type="text/css" media="print"
href="print.css">
```

The first link uses the media definition to target a CSS document from the computer screen. The second CSS document targets how data is presented when it is printed. Using this technique you can create two different presentation styles using the same content. One style is used for screen presentation and the other for print. Following is a list of the media names you can use:

- All: Suitable for all devices
- Braille: Intended for Braille tactile feedback devices
- Embossed: Intended for paged Braille printers
- Handheld: Intended for handheld devices (typically small screen, limited bandwidth)
- Print: Intended for paged material and for documents viewed on-screen in print preview mode
- Projection: Intended for projected presentations; for example, projectors
- Screen: Intended primarily for color computer screens
- Speech: Intended for speech synthesizers
- TTY: Intended for media using a fixed-pitch character grid (such as teletypes, terminals, or portable devices with limited display capabilities)
- TV: Intended for television-type devices (low resolution, color, limited-scrollability screens, sound available)

Having the names is great, but it does not help when there are so many different devices coming on to the market with different screen resolutions. To help with this you can modify the media type to look for screen resolutions and deliver the appropriate style sheet. Using the property device-width you can specify a style sheet for a specific width.

```
<link rel="stylesheet" type="text/css" media="(device-
width: 320px)" href="iphoneClassic.css">
```

Using CSS you can dynamically change the presentation of the content to best suit the device accessing the content.

Graphical Control with Bitmap, SVG, and CANVAS Elements

Tags are used in HTML5 to place and organize content at a level that is descriptive. This does not mean that the page will look good.

There are times, however, when you need to present graphics, too. Typically, HTML has provided support for pixel-based images only in JPG and GIF image format. With HTML5, you can now create mathematically generated images. The new formats are Scalable Vector Graphics, SVG, and CANVAS. The difference between the two is that SVG is an XML-based language that describes how an image should be displayed in 2D constructs. The CANVAS tag also describes 2D images, but it does so using JavaScript. The CANVAS tag also allows you to easily integrate interactivity within it using JavaScript.

Working with Bitmap Images on the Web

The web is not a friendly place for the designer. For many years you have been limited in the number of the file formats you can use. There are two predominant file formats used on the web for creating graphics: JPG and GIF.

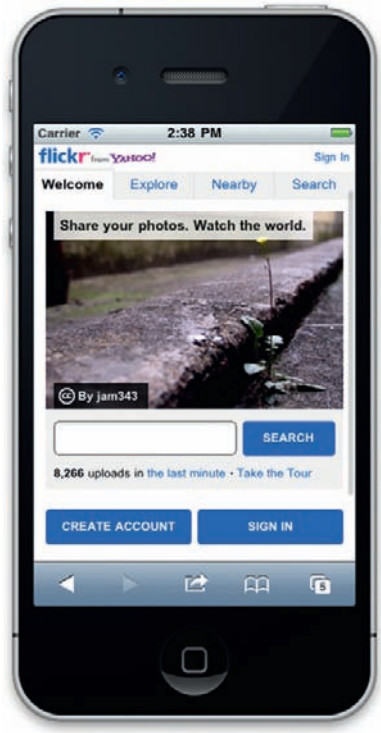


Figure 1.21 Images loaded to Flickr are in JPEG format.

JPEG, PNG, and GIF image formats are raster images created from pixels of individual color. Each has positives and negatives. JPEG images are an open standard managed by the Joint Photographers Expert Group. The JPEG file format allows you to create photo-realistic images. A great place to go to view millions of JPEG images is Yahoo's Flickr, as shown in Figure 1.21. A JPEG image is identified with either a JPEG or JPG extension .

The second file format used widely on the Internet is GIF, Graphics Interchange Format. Unlike JPEG, which supports millions of colors, the GIF file format only allows you to create images that support a color palette of 256 colors. On the face of it, the GIF format appears to be inferior to the JPEG format. However, the GIF format does have two features the JPEG format does not: setting transparency as a color and sequencing a series of images together to play back as a simple animation. It also handles solid colors more effectively.

Both JPEG and GIF image formats, however, are now being superseded by a more sophisticated image format: PNG.

Portable Network Graphics, PNG, are a raster-based file format that gives you the best of both JPEG and GIF and a little more. PNG image will support 32-byte images for photo-realistic presentation. Additionally, backgrounds in PNG images can be set to be transparent, the same as GIF images.

While PNG, GIF, and JPEG images are all great, it is difficult to programmatically change the graphical display of the images. For instance, you cannot create a bar chart using JPEG images that change as new data comes in. HTML5 introduces two solutions that address this problem: SVG and CANVAS.

Working with CANVAS and SVG Graphics

The CANVAS HTML5 element allows you to create bitmap images programmatically using JavaScript as the designer. Through this technique complex animations and interactive solutions can be created. Google has established ChromeExperiments.com (*www.chromeexperiments.com/*) to demonstrate powerful CANVAS and JavaScript experiments.

The second technology, SVG, Scalable Vector Graphics, is a vector-based technology that enables you to create images and animation using XML syntax similar to HTML. SVG started as an Open Standard in 1999. The support for SVG started out patchy in the mobile community. For a long time, only Mobile Safari on iOS devices supported SVG. However, the release of Android 3.0 and 2.4 have changed this.

SVG is a vector-based image format very similar to native Flash drawings. This gives you a great advantage when it comes to hi-res screen displays. No matter how detailed the screen is trying to be, the image will always be crisp without affecting the size of the file. In contrast, PNG and other raster images need larger files for crisper images at high resolutions. Figure 1.22 is an SVG illustration of the official SVG logo.

Unfortunately, although CANVAS drawings do render on all mobile browsers, the processing needed by the JavaScript engine is spotty. Apple does not use GPU enhancement for CANVAS drawing, relying on an already overburdened CPU. Some Android sellers running Android 2.2 can leverage the speedy V8 JavaScript engine to speed up CANVAS redrawing. This is a problem today, but will likely be mitigated by more powerful and smarter devices coming out in 2011 and 2012.

Figure 1.22 SVG graphics in iOS.

Figure 1.23 Video controlled by HTML.

Adding Video to Your Web Pages

Today, people will watch more than 2 billion movies on the Internet. That's right, two *billion*. Video is a big deal. Fortunately, HTML5 makes it easier for you to add video when you use a new HTML element called VIDEO, as shown in the screen shot in Figure 1.23.

Mobile devices were among the first widely used devices that supported the new VIDEO element. Over the last couple of years there has been a lot of controversy over how you can use the VIDEO element. Essentially, adding video with the VIDEO element is very simple: you can use a tag in your web page with a few attributes. Here is an example:

```
<video src="mobileVideo.mp4" ></video>
```

This example links to an MPEG4-encoded video. That's it. No fussing with plug-in OBJECT tags and parameters. Just use one line to add a simple, powerful HTML5 element.

This new VIDEO element has instigated a war of words about which video format the VIDEO element should play. There are three formats jostling for votes (MPEG4, Ogg, and WebM). The good news is, the top mobile web browsers now support VIDEO. For instance, Apple's Mobile Safari for iPhone and iPad, Google's Android and Chrome OS, and Mobile Opera and Mobile Firefox all support the new HTML5 VIDEO element.

In this section, I introduce you to:

- The HTML5 VIDEO element
- The attributes used to control content within the element
- How to encode HTML5 VIDEO
- Whether or not you should be using HTML5 VIDEO in your mobile website

By the time you reach the end of this article you will be comfortable working with the new HTML5 VIDEO element in your mobile web pages.

Controlling Video with VIDEO Tags

As we've seen, adding a VIDEO element requires only one line in your HTML. The following example adds opening and closing tags for the VIDEO element. The first tag includes an SRC attribute that points to a supported HTML5 video file (in this example we're pointing to an MPEG4 video):

```
<video src="mobileVideo.mp4" ></video>
```

That's it. Additional functionality can be added using the following attributes in the VIDEO element:

- Autoplay: The video will play immediately if already downloaded in your cache (this attribute does not work on iOS devices, but does on Android)
- Controls: A simple playback head will be added with VCR-like play and pause controls
- Height and Width
- Loop: You can loop the video
- Poster: Allows you to set a placeholder image for the video

To get the most out of your video playback you'll want to use some of these attributes. For instance, if you want your video to start playing when the web page has finished loading, you should use the Autoplay attribute as follows:

```
<video src="mobileVideo.mp4" autoplay></video>
```

The video won't automatically play if you don't include it.

A second useful attribute to add is the Controls attribute:

```
<video src="mobileVideo.mp4" autoplay controls></video>
```

Try viewing the Controls attribute in different mobile browsers such as Mobile Safari on the iPhone or iPad, and Android's browser—you'll notice it looks different in each browser. Each browser plays back the video differently, and each engine has its own default control style. This can make it difficult to present a video playback experience that's consistent from one browser to another.

You can override the default video playback features with some creative JavaScript and CSS.

Note
The *Autoplay* attribute doesn't work with Mobile Safari on the iPhone and iPad but will work for Android devices.

Using JavaScript to Control the VIDEO Element

JavaScript is able to control any elements in HTML. The VIDEO element is a valid, first-level element JavaScript can control. This means you can control media using your own custom controls. The following example will show you how to add a custom Play/Pause button to your video.

Start with a blank HTML5 page:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Adding Video to a Mobile App</title>
</head>
<body>
</body>
</html>
```

In the BODY section, add the VIDEO element and link to a video file:

```
<video autoplay >
<source src="mobileVideo.mp4">
</video>
```

You can see here that the video file doesn't have any attributes that control playback. You can add those controls programmatically with JavaScript. Let's start by adding the controls that play the movie:

```
<a href="#" >Play/Pause</a>
```

After the VIDEO element, add the following JavaScript:

```
<script>
  var video = document.getElementsByTagName('video')[0];
</script>
```

This script gives the VIDEO element a name you can reference. The final step is to add a script to the ANCHOR tag:

```
<a href="#" onclick="if (video.paused) video.play();
else video.pause()">Play/Pause</a>;
```

The ANCHOR element uses an on-click event to trigger an IF/ELSE JavaScript command. If the button is pressed and the video hasn't been played, then the video will start to play. Else, if the video is playing and the button is selected it will pause the video. Altogether your code will look like this:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Adding Video to a Mobile App</title>
</head>
<body>
<video autoplay >
<source src="mobileVideo.mp4">
</video>
<script>
var video = document.getElementsByTagName('video')[0];
</script>
<br />
<a href="#" onclick="if (video.paused) video.play();
else video.pause()">Play/Pause</a>
</p>
</body>
</html>
```

An additional benefit of using JavaScript to control the presentation of your controls is that you can use CSS to style them. Here is a basic style applied to our video controls:

```
<!DOCTYPE HTML>
<html>
<head>
<title>Video in HTML5</title>
<style type="text/css">
a {
        font-family: Arial, sans-serif;
        font-size: large;
        text-decoration: none;
        color: #C0C0C0;
}
h1 {
```

```
            Arial, sans-serif;
            font-size: 24pt;
            color: #C0C0C0;
    }
    body {
            background-color: #000000;
    }
    </style>
    </head>
    <body>
    <h1 align="center">Video on your Mobile Device</h1>
<p align="center">
    <video autoplay >
     <source src="mobileVideo.mp4">
    </video>
     <script>
      var video = document.getElementsByTagName('video')[0];
     </script>
    <br />
     <a href="#" onclick="if (video.paused) video.play();
else video.pause()"> Play/Pause</a>
    </p>
    </body>
    </html>
```

There are other controls you can add to your VIDEO element. You can add a playback head to track where you are in the video, for example, as well as fast-forward and rewind.

Encoding Video and Audio for Web Delivery

The H.264 support, also known as MPEG4, is the video and audio format supported on your iPhone and used by many companies. Unfortunately, MPEG4 has patents that protect the technology. This has lead to confusion about whether or not you can freely use MPEG4 video in your web pages. The patent group managing MPEG4, the MPEG-LA group, has stated it will *not* charge royalties for the use of MPEG4 video embedded into web pages. Check out the comprehensive FAQ Microsoft put together here: *www.microsoft.com/windows/windowsmedia/licensing/mpeg4faq.aspx*.

The alternatives to H.264 are Theora and webM formats. Technically, H.264 is cleaner at higher resolutions (you'd have to be a videophile to see the difference), but overall the quality difference between H.264 and Theora/WebM is minimal. Ultimately, consumers of video/audio content will determine which CODEC will become the format of choice.

The challenge we face now is identifying which browser supports which video format. Table 1.1 provides a brief breakdown: As you can see, not all browsers support all video formats.

Table 1.1 Supported HTML5 Video CODECs

	MPEG4	Ogg Theora	WebM
Mobile Firefox	No	Yes	No
Chrome OS	No	Yes	Yes
Mobile Safari	Yes	No	No
Windows Phone 7	No	No	No
Mobile Opera	No	Yes	No

Creating Video in H.264 Format

Creating H.264-formatted video is relatively easy. There are dozens of products on the market that will take almost any video format and convert it to MPEG4, including Cucusoft, MP4 Convertor, and more. For Mac users, it's even easier; your copy of iMovie already supports MPEG4 format.

Creating Video in Ogg Theora and WebM Formats

There are a few tools that allow you to create Ogg Theora video. You can check out the latest solutions here: *www.theora.org/downloads/*.

The challenge to WebM as a new CODEC is support in creating WebM files. Fortunately, the group managing WebM as an Open Standard, the WebM Project, has a website with open-source tools you can use to encode video into WebM formats. Check it out at *www.webmproject.org/code*.

Ensuring Your Video Plays Back

Currently, not all mobile browsers support the same video playback. This leaves you with a thorny problem: How do you support video across all these different browsers?

Well, HTML5 has that covered. The VIDEO element allows you to add nested SOURCE elements:

```
<video autoplay controls>
<source src="sample.mp4">
<source src="sample.ogv">
<source src="sample.webm">
</video>
```

Using this technique guarantees your HTML5-compliant web browser will play back your video by selecting the first file that matches a CODEC installed on your device.

Streaming for Video Playback on Mobile Devices

In addition to controlling whether or not video will actually play on your mobile device, you also need to know whether or not the bandwidth streaming to the phone will allow the video to play back.

There are two ways you can control the playback of video: download or streaming.

Downloading files requires that the whole video is either first downloaded or, if the CODEC allows for it, will start playing back as the video is being downloaded, a feature known as progressive download. The MPEG4 format has progressive download built into the architecture. This allows for rented movies to start playing within a couple of minutes of selecting the title on your iPad. You do not need to wait for two to three hours for the whole movie to download.

Streaming is a method where the video is delivered in a series of small packets to the device. Unlike progressive download where the whole movie is downloaded to the browser cache, streaming does not leave any file on the device. The method allows for live video to be streamed to a device. The downside to streaming is that you must have a constant connection to the Internet for it to work.

Network connectivity is the largest challenge for delivering video to mobile devices. Today, four main data speeds are used to send data to a mobile device:

- WiFi
- 4G LTE
- 3G CDMA
- EDGE

The size of the screen, such as handheld or tablet, will determine what size video stream you are sending. For instance, for a phone, you can stream smaller video simply because the screen is smaller. However for a tablet device you will want to stream an HD quality video file to fill up the larger screen.

Larger video images will require faster Internet connections. Both 4G and WiFi are more than capable of streaming HD quality video. A 3G signal will be able to stream only a sub-DVD quality video, whereas EDGE, the slowest connection speed, will be able to stream only very small video images.

By default, HTML5 does not allow you to switch files as determined by network connection. Fortunately, you can use a feature built into modern web servers, such as Microsoft's Internet Information Server 7, called HTTP Live Streaming. With HTTP Live Streaming you can add a single file reference in the web page and the server can then dynamically select the appropriate video for the network bandwidth.

For HTTP Live Streaming to work, the web browser does need to understand and support the protocol. Fortunately, iOS, BlackBerry PlayBook, and Android do. It is also very likely the Windows Phone 7 will support HTTP Streaming when the browser adopts HTML5.

Applying New Web API Functionality to Your Mobile Web Pages

CSS, SVG, and Video are all great improvements to HTML5. The role for HTML5, however, is not simply to add eye-candy, but to enable developers to create applications in web browsers that are equal in performance to desktop applications. To accomplish this you need a powerful development language that gives developers the ability to create sophisticated solutions. The answer to this is JavaScript, the world's most popular programming language.

Currently, the belief is that web applications are simply not as powerful as desktop applications. The reason for this is not due to JavaScript, but the engines inside your web browser that process JavaScript. The faster a script can be processed, the more sophisticated your applications can become.

HTML5 is expanding to support application programming interfaces (APIs) that enable complex system integration inside your web page. The new APIs, such as geolocation and local data storage, are complex and require sophisticated use of JavaScript to make them work.

JavaScript is not a new technology. The roots of JavaScript go back to 1993 when Netscape Communications included a scripting technology called LiveScript with its web browser. Incorporating even a simple programming language that enables interactivity in the web browser became extremely popular.

The current release of JavaScript has dramatically matured the original LiveScript language. Unlike desktop applications that run code optimized for an operating system, JavaScript must be interpreted within a virtual machine translator running inside the web browser. This process inherently forces JavaScript solutions to run more slowly. To compensate for this, Google uses a technology called V8 that dramatically improves the processing of JavaScript code in Android 2.2. Competitors such as Apple and Microsoft have not brought their speedy JavaScript accelerators to the mobile platform (yet).

Today's JavaScript allows you to build desktop-like applications that run inside your web browser. Google's Wave solution is an excellent example of a massively complex application that is run using JavaScript.

JavaScript is the most popular development language in the world with millions of users. The technology is not too complex

to learn; indeed if you have any experience with C#, Java, or ActionScript, then you will likely pick up JavaScript quickly.

JavaScript is so popular that it has its own standard. ECMA International is an industry association founded in 1961 and dedicated to the standardization of Information and Communication Technology (ICT) and Consumer Electronics (CE). The standardized version of JavaScript is managed by ECMA. The full standard name is ECMA-262, but is often referred to as EcmaScript. ECMA-262 as a standard is well-supported by all web browsers.

Geolocation on Your Phone

There is no doubt that the tech world is going mobile. Devices now need to know where they are geographically. In preparation for this, HTML5 includes support for geolocation. The iPhone and Android phones are already geolocation enabled, as shown in Figure 1.24.

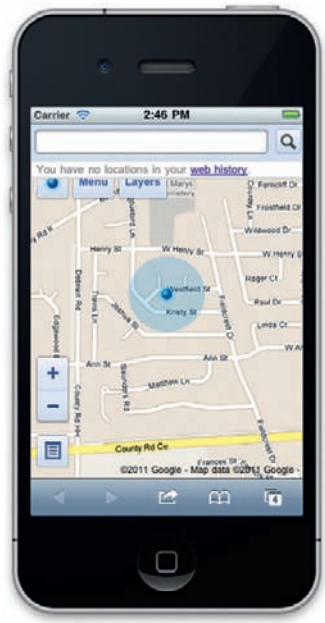


Figure 1.24 Google can use the GPS in your phone to detect where you are.

The following example uses Google Map's service and the browser's geolocation API to tell you where you are located. The first step is to load the Map services.

```
    <script src="http://maps.google.com/maps?file=api&amp;v=2&amp;sensor=false&amp;key=ABQIAAAAiUzO1s6QWHuyzxx-JVN7ABSUL8-Cfeleqd6F6deqY-Cw1iTxhxQkovZkaxsxgKCdn1OCYaq7Ubz3SQ" type="text/javascript"></script>
```

The Google Map services are publically accessible. Now you need to start writing JavaScript. The first step is to define a series of variables that you can use in your code:

```
    var map;
    var mapCenter
    var geocoder;
    var fakeLatitude;
    var fakeLongitude;
```

With your JavaScript variables defined, you can create the first function that initializes the geolocation services in your web browser:

```
    function initialize()
    {
           if (navigator.geolocation)
           {
                  navigator.geolocation.getCurrentPosition(function (position) {
           mapServiceProvider(position.coords.latitude,position.coords.longitude);
                  },
           }
           else
           {
```

Use of Google Maps on a website requires a unique key. You can get all the information for getting a key at the web address code.google.com/apis/maps/.

```
          alert("I'm sorry, but Geolocation services are
not supported by your browser or you do not have a GPS
device in your computer. I will use a sample location to
produce the map instead.");
          fakeLatitude = 49.273677;
          fakeLongitude = -123.114420;
          mapServiceProvider(fakeLatitude,fakeLongitude);
         }
   }
```

The next function instructs the geolocation services to use the Google Map service.

```
   function mapServiceProvider(latitude,longitude)
   {
             mapThisGoogle(latitude,longitude);
   }
   function mapThisGoogle(latitude,longitude)
   {
          var mapCenter = new GLatLng(latitude,longitude);
          map = new GMap2(document.getElementById("map"));
          map.setCenter(mapCenter, 15);
          map.addOverlay(new GMarker(mapCenter));
          geocoder = new GClientGeocoder();
          geocoder.getLocations(latitude+','+longitude,
addAddressToMap);
   }
```

The final code completes the mapping:

```
   function addAddressToMap(response)
   {
          if (!response || response.Status.code != 200) {
                  alert("Sorry, we were unable to geocode
that address");
          } else {
                  place = response.Placemark[0];
                  $('#address').html('Your address: '+place.
address);
          }
   }
   window.location.querystring = (function() {
     var collection = {};
     var querystring = window.location.search;
     if (!querystring) {
       return { toString: function() { return ""; } };
     }
     querystring = decodeURI(querystring.substring(1));
     var pairs = querystring.split("&");
     for (var i = 0; i < pairs.length; i++) {
       if (!pairs[i]) {
         continue;
       }
     var separatorPosition = pairs[i].indexOf("=");
```

```
    if (separatorPosition == -1) {
        collection[pairs[i]] = "";
    }
    else {
      collection[pairs[i].substring(0, separatorPosition)]
= pairs[i].substr(separatorPosition + 1);
      }
    }
    collection.toString = function() {
      return "?" + querystring;
    };
    return collection;
})();
```

The final result is that you can use geolocation to determine where you are using just your web browser. This is very useful in mobile web browsers where you can link map services to geographically based tools.

Local Data Storage

Key to applications is the ability to store data. In the past you have been able to do this by using complex cookies or Ajax commands that leverage the ability to send data back to a database. The ability to store data locally in your web browser is dramatically improved with the implementation of LocalStorage.

LocalStorage is essentially the ability to have an SQL-like database running in your web browser. An example of LocalStorage being used is Google's version of Gmail for the iPhone/Android. Using LocalStorage, you can view and send e-mail with Gmail without having a web connection. The e-mail is resynchronized with the mail servers when a new network connection is established.

You access LocalStorage in your JavaScript by using the GlobalStorage object. The following example demonstrates LocalStorage being used.

The first step for the example in the previous image is to create an area where you can type some text. You will use standard form controls:

```
    <textarea id="text" class="freetext">
    </textarea> Item name <input id="item_name" type="text"
value="new item" />
```

An event will be added to the INPUT submit button to trigger the JavaScript to run:

```
    <input onclick="writeLocal();" type="button" value="Save" />
```

The LocalStorage posts the data stored in the browser to the web page. An area with the ID "items" is defined.

```
    <div id="items">
    </div>
```

The first function run in your JavaScript is to define that the content on the page is to be associated with the website on which your page is being hosted:

```
function $(id) { return document.getElementById(id); }
var host = location.hostname;
var myLocalStorage = globalStorage[host];
```

The second function allows you to store data using the LocalStorage API:

```
function writeLocal() {
var data = $('text').value;
var itemName = $('item_name').value;
myLocalStorage.setItem(itemName, data);
updateItemsList();
}
```

As with any SQL database you need to be able to delete entries. The following function allows you to delete items using the removeItem property:

```
function deleteLocal(itemName) {
myLocalStorage.removeItem(itemName);
updateItemsList();
}
```

The following sample shows you the whole program with some simple CSS styling for presentation:

```
    <html><head><title>HTML5 Web Storage / localStorage</
title></head>
    <style>
    .freetext {
            width: 100%;height: 40%;overflow: hidden;background:
#FFE;font-family: sans-serif;font-size: 14pt;-moz-border-radius:
10px;-webkit-border-radius: 10px;
    }
    li {
            padding: 4px;width: 400px;
    }
    input {
            margin: 2px;border-style: solid;-moz-border-radius:
10px;-webkit-border-radius: 10px;color: #666;padding: 2px;
    }
    body {
            font-family: “Lucida Sans”, “Lucida Sans Regular”,
“Lucida Grande”, “Lucida Sans Unicode”, Geneva, Verdana,
sans-serif;color: #FF0000;font-size: medium;
    }
    </style>
    <body><textarea id=“text” class=“freetext ”></textarea>
Item name <input id=“item_name” type=“text” value=“new item” />
<input onclick=“writeLocal();” type=“button” value=“Save” />
<div id=“items”></div>
    <script>
```

```
    function $(id) { return document.getElementById(id); }
    var host = location.hostname;
    var myLocalStorage = globalStorage[host];
    function writeLocal() {
     var data = $('text').value;
     var itemName = $('item_name').value;
     myLocalStorage.setItem(itemName, data);
     updateItemsList();
    }
    function deleteLocal(itemName) {
     myLocalStorage.removeItem(itemName);
     updateItemsList();
    }
    function readLocal(itemName) {
     $('item_name').value=itemName;
     $('text').value=myLocalStorage.getItem(itemName);
    }
    function updateItemsList() {
     var items = myLocalStorage.length
     // list items
     var s = '<h2>Items for '+host+'</h2>';
     s+= '<ul>';
     for (var i=0;i<items;i++) {
      var itemName = myLocalStorage.key(i);
      s+= '<li>'+
        '<div style="float:right;">'+
        '<input type="button" value="Load" onclick="readLocal
(\"+itemName+'\');"/'+'> '+
        '<input type="button" value="Delete" onclick="delete
Local(\"+itemName+'\');"/'+'> '+
        '</div>'+
        '<strong>'+itemName+'</strong>'+
        '</li>';
      }
      $('items').innerHTML = s+'</ul>';
    }
    window.onload = function() {
      updateItemsList();
      $('text').value=[
        'Quick and dirty Web Storage sample:',",
        '1) Write some text',
        '2) Give it some name',
        '3) Click Save button',",
        'Data is stored and retrieved using Web Storage (no
cookies and no server side).'].join('\n');
    }
    </script></body></html>
```

As you can see, the implementation of LocalStorage allows you to store data without using cookies or server side databases.

Developing for Specific Mobile Browsers

The standards covered up to this point will work on all HTML5 web browsers, whether they are on your phone, laptop, TV, or tablet. Mobile phones have many additional features you can support. The goal of this section is to highlight popular features specific to different phones and to direct you to where you can get additional information.

Apple's Mobile Safari

Figure 1.25 Mobile Safari logo.

At the time of this writing, there is a clear winner when it comes to advanced features in a mobile web browser: Apple. When iOS was first presented in January 2007, Steve Jobs went to great lengths to promise that Mobile Safari delivered "the whole web," not a broken presentation. Figure 1.25 is the first Mobile Safari logo.

On the whole, Apple has delivered on its promise. What you will find as you move through these articles, is that all the solutions will work on Apple's iOS. With that said, there is one major web feature supported by all mobile operating systems that Apple does not support: Adobe's Flash.

Apple and Adobe are now enjoying an infamous power play over standards. Apple is pushing HTML5 and Adobe is pushing Flash. While the battle is interesting, Flash-enabled websites, such as game sites, do not work on the iPhone. Time will tell how this battle will resolve itself.

With that said, Apple's standards support in Mobile Safari is amazing. Mobile Safari has full support of SVG, embeddable TrueType fonts, HTML5 Video (using MPEG4), web sockets, and can even use hardware features such as Gyroscope and the Accelerometer through a DeviceOrientation API.

There are some HTML5 features still missing in Mobile Safari, most notably Web Workers. Web Workers are features in JavaScript that enable two or more scripts to run simultaneously, a critical feature for enterprise scale applications. Today, iOS does not allow for this.

You can find out more about specific iOS features in Mobile Safari at Apple's developer website, as shown in Figure 1.26 (*http://developer.apple.com/ios*).

Google's Android Browser

Google's Android browser, like Apple's, is built out from WebKit. A big challenge with Android, however, is fragmentation. Google provides the operating system as a free solution that can be adopted by any hardware company. The problem with this approach is that mobile phone companies such as Motorola, Samsung, and HTC can choose what they want to add and remove

Figure 1.26 iOS developer site.

from the Android OS. For instance, some vendors include Google's V8 JavaScript accelerated engine but others do not. The result is that you do not have a consistent user experience. This does not mean you should not develop for Android; it just means you need to spend more time in your quality testing and controls.

Find out more about Android development at this site, as shown in Figure 1.27 (*http://developer.android.com/guide/*).

RIM's BlackBerry 6 and PlayBook

In many respects, RIM set the groundwork for today's smart phones with the BlackBerry phone. Within many corporations today, a BlackBerry phone is still very popular. But they are losing ground fast.

To compete against Apple and Google, RIM has done some solid soul searching and brought its core OS up to specification with competing technologies. The new BlackBerry 6 is sleek. And, guess what, the browser is based on WebKit. You know that means lots of HTML5.

Here are links to developer sections on BlackBerry 6 (Figure 1.28): *http://us.blackberry.com/apps-software/blackberry6/*.

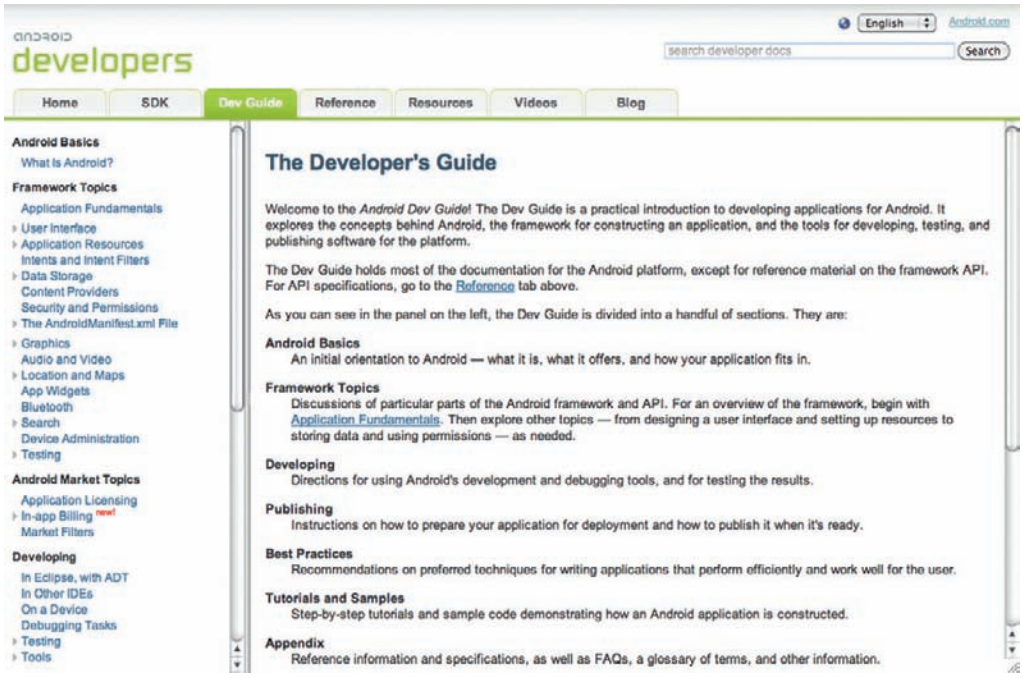


Figure 1.27 Android developer site.

Figure 1.28 RIM's developer site.

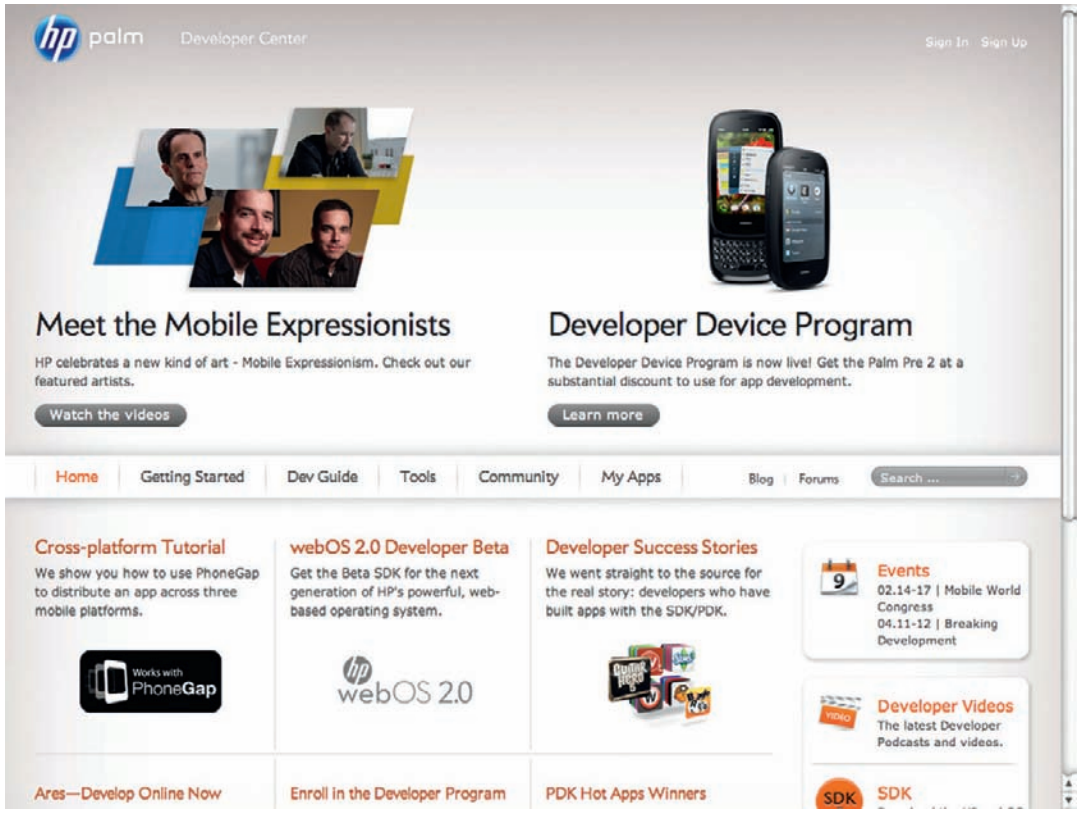
HP/Palm WebOS

HP acquired Palm and its WebOS platform in 2010. Since that time, HP has been very quiet about the acquisition. It is clear, however, that HP wants to enter the device market and that WebOS will be a competitive advantage for the company. The key to WebOS is that the whole OS is built using the web standards CSS, JavaScript, and HTML5. For more information check out *http://developer.palm.com/* (see Figure 1.29).

Developing Websites for the Rest

You may think that the term smart phone did not exist until the iPhone hit the scene. But the term itself has been around for many years, with many phones supporting web browsers since 2000. The problems they all share are poor support for standards and a terrible user experience. With that said, Nokia, the global leader in phone sales, has placed hundreds of millions of phones with web access into the hands of people around the world.

Figure 1.29 HP's WebOS developer site.



Nokia's MeeGo and Symbian

Nokia owns the mobile phone market. So why is it scared? The problem is that its two operating systems have not caught App fever and are not seen as platforms in the same light that Android and iOS are.

For this reason, Nokia is looking to change its core OS. Nokia has now agreed to support Windows Phone 7 OS.

Until then, you have MeeGo and Symbian. Both allow for apps to be downloaded, including web browsers. You can easily add Google Analytics to your website and see how many people are surfing to your website using Nokia phones. If your site does receive a lot of traffic from Nokia devices and you need to support them, keep this mantra close to your heart: keep it simple. Nokia phones are significantly underpowered compared to smart phones and even text-only web pages can take a long time to draw correctly on the screen.

Windows Phone 6.5 and Earlier

Windows Phone 6.5 was a big step forward when it came to the web browser even if those steps forward still placed it several steps behind Mobile Safari. Windows has improved a lot since the release of 6.5 with Windows Phone 7, but there are still a lot of 6.5 phones being used. When developing for these phones make sure you pay attention to the amount of content you place on a page. Too much will cause the page to take a very long time to load. Finally, keep to simple HTML and do not use too much CSS or JavaScript. The phone will choke.

Tablet Development

Much of this article places a focus on mobile development for smart phones. But, there is a second category of device that is gaining strong popularity—tablets. Tablet devices have been around for more than a decade, but it took Apple's iPad to reintroduce the category to the world. Apple was able to sell more than 17 million iPads from March through December of 2010. Following suit, Google introduced Android 3.0 Honeycomb as its competing tablet OS, RIM released the PlayBook, and HP is working on new tablets. The 2011 CES show presented more than 100 tablets. Figures 1.30 and 1.31 are images of the iPad and Honeycomb tablets, respectively.

Seems like we cannot get enough of them now!

As smart phones and laptops have different user experiences, so do tablets. Typically, tablet users are engaged with their content longer than smart phone users and the screen is much larger

Figure 1.30 Apple's iPad.

Figure 1.31 Google's Honeycomb tablet.

(though still smaller than a desktop). It still remains to be seen how to develop for tablets, but 2012 will be the year you start looking at tablets and considering how you need to ensure your website works for them as well as for smart phones and laptop computers.

Summary

The goal of this article was to introduce you to mobile web development. In many ways it is very similar to desktop website development—HTML5 is HTML5 no matter on which device you install it.

What is different is how you use and interface with the device. Smart phones are just very different devices to a laptop.

The next articles will introduce and review popular frameworks that allow you to quickly build out web sites that target smart phones. It is important to remember that each of these frameworks are built using the same HTML, CSS3, and JavaScript you have covered in detail in this article. What this means is that you can easily extend and enhance the frameworks you are about to use.

Let's roll up our sleeves and start creating web pages that target the device in your hand.