shortcut

Your Short Cut to Knowledge

Optimizing Your Website for Mobile Safari

Ensuring your website works on the iPhone and iPod touch

August Trometer

Addison-Wesley
Pearson Education

AWPROFESSIONAL.COM

Introduction....2
iPhone Basics....4
Getting Standard....11
Getting Compatible....20
Getting Optimized....34
Using Rich Media....46
How-To's and Other Quick Tips....54
Resources....62

Introduction

By all accounts, the iPhone is one of the most successful products in consumer electronics history. In a little over two months following its release, Apple and AT&T sold more than one million iPhones in the United States alone. Then, as if to sweeten the pot, Apple launched the iPod touch, a new iPod that shares the same touch-screen technology as the iPhone.

In addition to the touch screen, the iPhone and iPod touch have something else in common: Mobile Safari.

With most mobile devices, the browser is anemic and underpowered. Although passable in a pinch, they usually display only bare-bones, stripped-down versions of most web pages. Mobile Safari changes that. It's a full-fledged browser that displays pages just the way you would see them in a normal desktop browser.

And if Apple has its way, this is just the beginning. Mobile Safari's underlying technology, WebKit, is Open Source, and it won't be long before other manufacturers start to make devices with browsers very similar to Mobile Safari.

If you have a website, these things are important. As we all know, site visitors who have problems with a website usually don't return. As the number of Mobile Safari users grows, you are potentially talking about millions of visitors who may or may not come back to your site.

Getting Compatible

Now that your website is valid, well formed, and standard, you're ready to take the next step: making your site Mobile Safari compatible. By compatible, I mean that pages load on Mobile Safari just as they would on the desktop, with no errors and no visual defects.

In many respects, Mobile Safari looks and acts just like Safari for Mac and Windows, but it has some significant limitations, too. Understanding these limitations and adjusting your site to accommodate them will get you on your way to full Mobile Safari optimization.

Windows and Dialogs

One of the first things you'll notice when using Mobile Safari is that there is only one "window," and there isn't a tab bar. Because of the limited space on Mobile Safari's viewport, Apple wisely chose to do away with the notion of multiple windows and tabs. Instead, Mobile Safari has pages to keep multiple web pages open at the same time.

To access these pages, a user taps the **Pages** button at the bottom of the screen. The user can flick through tiny versions of the pages (shown in Figure 3.1). To bring a page into the viewport, all the user needs to do is tap the page icon to bring it forward into Mobile Safari's viewport.

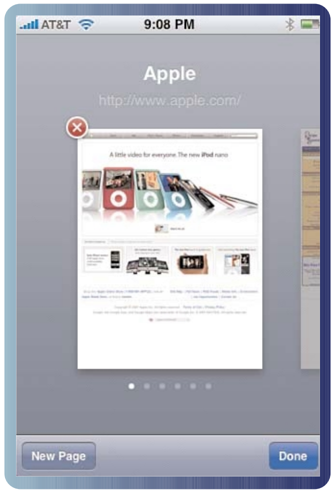


FIGURE 3.1
Selecting a page in Mobile Safari

Of course, this is time-consuming and, to be honest, a hassle. Worse yet is the fact that many times the tiny images look the same. If you tap the wrong page, you have to go through the process all over again.

In addition, because of memory constraints, Mobile Safari has an eight-page limit. This means that, no matter what, a user can only have eight pages open at a time. If a user goes beyond eight, Mobile Safari begins deleting pages to clear space for the next one. The pages that have been most recently used are kept while the pages the user hasn't looked at in a while go on the chopping block.

Because of this, it's prudent on your part to limit, if not eliminate, the use of pop-up and new windows through your website. Pop-ups are irritating in general, but for a Mobile Safari user, they can be torture, especially if a pop-up unexpectedly blows away a page the user wanted to keep open.

If you need to communicate with a user through a dialog box, that's fine. In fact, Mobile Safari supports the three main JavaScript dialog types:

- `alert()`
- `confirm()`
- `prompt()`

When you use these, a special type of dialog appears on the Mobile Safari screen (see Figure 3.2), and the user can then respond to it.

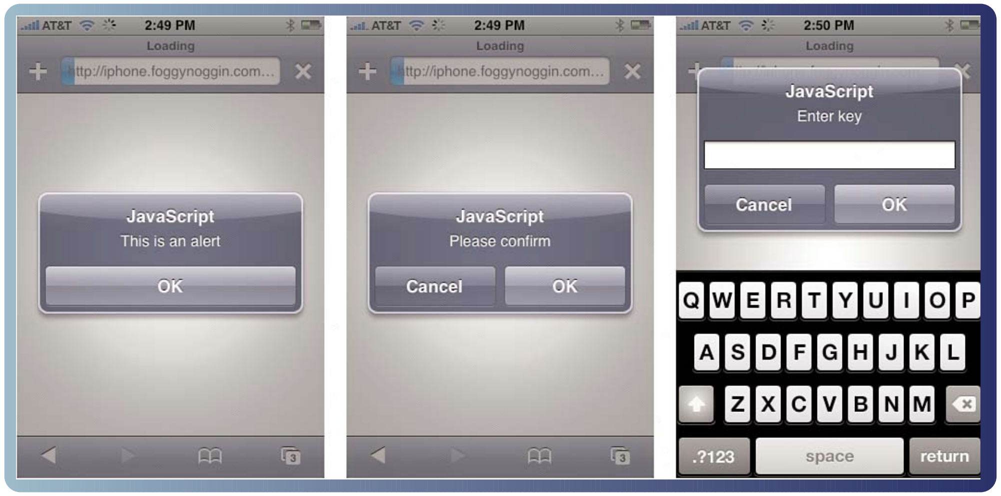


FIGURE 3.2
The JavaScript `alert()`, `confirm()`, and `prompt()` dialog boxes

iFrames and Framesets and `overflow: scroll;`

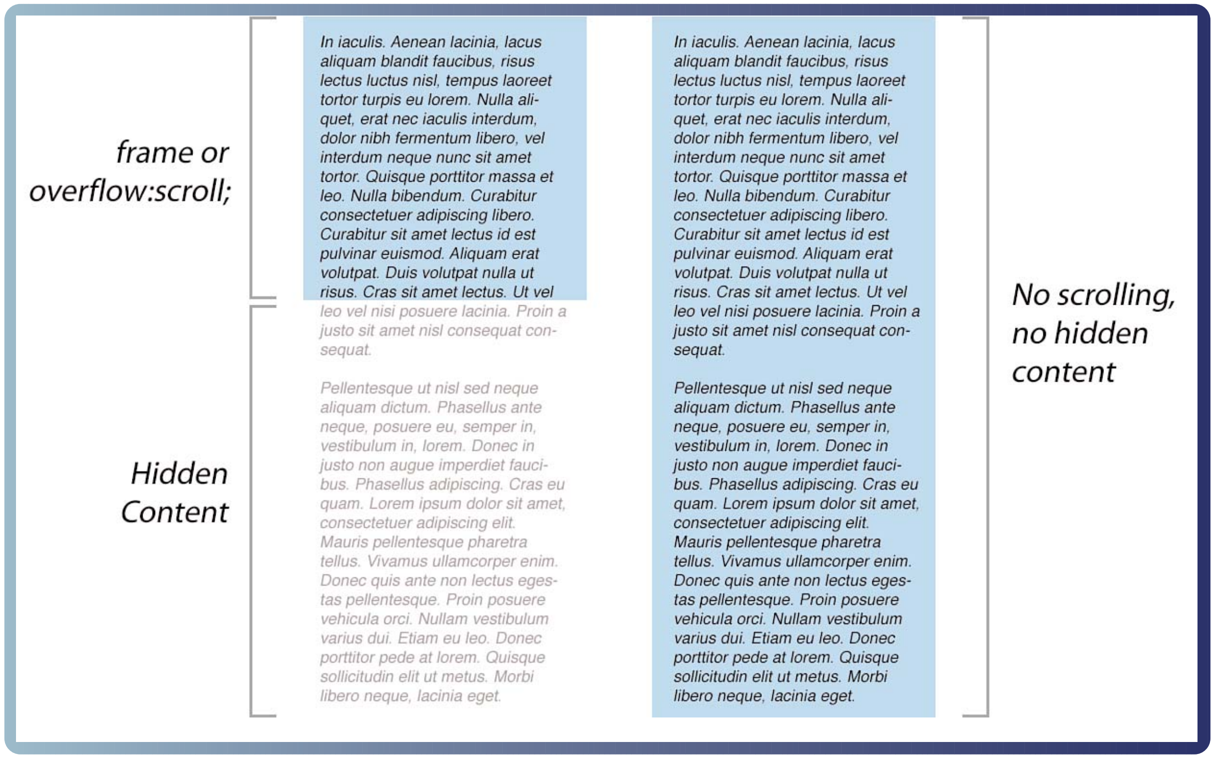
For Mobile Safari, framesets are bad, and iFrames aren't much better. Although, in most cases, frames will work in Mobile Safari, in some cases, the page simply won't render at all.

There's also the matter of scrolling. As mentioned previously, to conserve valuable screen real estate, Mobile Safari doesn't use scrollbars. On a desktop computer, scrollbars, in addition to allowing the user to scroll content, also serve as an indication that there is actual content to scroll. When viewing a long passage of text, the scrollbar appears to let them know that there is more content off the bottom of the screen. Shorter passages don't require scrolling, and the scrollbars vanish.

In Mobile Safari, however, there are no scrollbars, no matter how long the content. Because most web pages are longer than the viewport, it's easy for the user to assume there is more content below. But in a frame (essentially a box in the middle of the page), if there is scrollable content, the Mobile Safari user won't know that. No scrollbars appear to indicate extra content, and anything not plainly visible in that box will go unread.

In Figure 3.3, you can see the two different ways of handling content.

FIGURE 3.3
Selecting a page in Mobile Safari



In iaculis. Aenean lacinia, lacus aliquam blandit faucibus, risus lectus luctus nisl, tempus laoreet tortor turpis eu lorem. Nulla aliquet, erat nec iaculis interdum, dolor nibh fermentum libero, vel interdum neque nunc sit amet tortor. Quisque porttitor massa et leo. Nulla bibendum. Curabitur consectetuer adipiscing libero. Curabitur sit amet lectus id est pulvinar euismod. Aliquam erat volutpat. Duis volutpat nulla ut risus. Cras sit amet lectus. Ut vel leo vel nisi posuere lacinia. Proin a justo sit amet nisl consequat consequat.

Pellentesque ut nisl sed neque aliquam dictum. Phasellus ante neque, posuere eu, semper in, vestibulum in, lorem. Donec in justo non augue imperdiet faucibus. Phasellus adipiscing. Cras eu quam. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Mauris pellentesque pharetra tellus. Vivamus ullamcorper enim. Donec quis ante non lectus egestas pellentesque. Proin posuere vehicula orci. Nullam vestibulum varius dui. Etiam eu leo. Donec porttitor pede at lorem. Quisque sollicitudin elit ut metus. Morbi libero neque, lacinia eget.

In iaculis. Aenean lacinia, lacus aliquam blandit faucibus, risus lectus luctus nisl, tempus laoreet tortor turpis eu lorem. Nulla aliquet, erat nec iaculis interdum, dolor nibh fermentum libero, vel interdum neque nunc sit amet tortor. Quisque porttitor massa et leo. Nulla bibendum. Curabitur consectetuer adipiscing libero. Curabitur sit amet lectus id est pulvinar euismod. Aliquam erat volutpat. Duis volutpat nulla ut risus. Cras sit amet lectus. Ut vel leo vel nisi posuere lacinia. Proin a justo sit amet nisl consequat consequat.

Pellentesque ut nisl sed neque aliquam dictum. Phasellus ante neque, posuere eu, semper in, vestibulum in, lorem. Donec in justo non augue imperdiet faucibus. Phasellus adipiscing. Cras eu quam. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Mauris pellentesque pharetra tellus. Vivamus ullamcorper enim. Donec quis ante non lectus egestas pellentesque. Proin posuere vehicula orci. Nullam vestibulum varius dui. Etiam eu leo. Donec porttitor pede at lorem. Quisque sollicitudin elit ut metus. Morbi libero neque, lacinia eget.

On the left, you can see a frame, or a block with a style of `overflow: scroll;` (they both display the same). Much of the content is hidden to the user. On a desktop browser, scrollbars would indicate the extra content, and the viewer would know to scroll to see the hidden text. Mobile Safari, on the other hand, shows no scrollbars, and the user would have no idea that any content was hidden.

A better solution, shown on the right side of Figure 3.3, is to allow the block of text to flow to its natural end. This prevents any content from being hidden and prevents the user from missing important information they might otherwise miss.

The bottom line: If you're using frames or `overflow: scroll;`, a redesign might be in order.

As pointed out in Chapter 2, "Getting Standard," you should be designing your site in columns and boxes. With your content flowing in narrow columns, all users (even those with a desktop browser) will be able to see all the content on the page. Nothing will be hidden, and nothing will be missed.

Plug-ins, Java, and Flash

You've probably already heard the bad news, but I'll say it for the record: Mobile Safari does not support plug-ins, Java applets, or Flash.

Sure, that's inconvenient, but when you think about it, it makes sense. The iPhone and iPod touch are highly optimized devices. Memory and processor use are concerns with every single operation. A rogue line of code from a Java applet or plug-in could dramatically impact the performance of the phone. As for Flash, not only does it hamper both battery life and device performance, but in many cases, it simply won't work. Many Flash sites rely on mouseovers and drags, and Mobile Safari doesn't support this. Any Flash that relies on these events remains static and unusable with Mobile Safari.

In most cases, Flash is used for basic animation. If you need animation for your website, try using Apple's `<canvas>` tag. The `<canvas>` tag allows for very complex vector graphics and animations using JavaScript as the control language. After you get the hang of it, it's fairly easy to do.

For example, the following code randomly draws colored circles on a given canvas area:

```
<html>
     <head>

          <script type="text/javascript">

               function draw() {

                    // Get a reference to the canvas
                    var canvas = document.getElementById('myCanvas');

                     // Make sure this browser can use a canvas object
                    if (canvas.getContext){

                         // Get the canvas context
                         var ctx = canvas.getContext('2d');

                         // Set a drawing color
                         // We're using random numbers to generate the color
                         // There's also a random Alpha setting for transparency
                         ctx.fillStyle = "rgba(" + Math.floor(Math.random()*255) + "," +
                          Math.floor(Math.random()*255) + "," +
                          Math.floor(Math.random()*255) + "," + Math.random() + ")";

                         // Draw a circle
                         // Place the circle in a random spot on the canvas
                         // with a random radius
                         ctx.beginPath();

ctx.arc(Math.floor(Math.random()*200),Math.floor(Math.random()*200),Math.floor(Math.random()
*75),0,Math.PI*2,true);
                         ctx.fill();

                         // Set a timeout function to
```