Building Touch Interfaces with HTML5

Speed up your site and create amazing user experiences

DEVELOP AND **DESIGN**

Stephen Woods

Building Touch Interfaces with HTML5

Speed up your site and create amazing user experiences

DEVELOP AND **DESIGN**

Stephen Woods



PEACHPIT PRESS WWW.PEACHPIT.COM

Building Touch Interfaces with HTML5: Develop and Design

Stephen Woods

Peachpit Press

www.peachpit.com

To report errors, please send a note to errata@peachpit.com. Peachpit Press is a division of Pearson Education.

Copyright © 2013 by Stephen Woods

Project Editor: Nancy Peterson Production Editor: Rebecca Chapman-Winter Development Editor: Jeff Riley Compositor: Danielle Foster Technical Editor: Nicholas C. Zakas Copyeditor: Gretchen Dykstra Proofer: Darren Meiss Indexer: Jack Lewis Cover Design: Aren Straiger Interior Design: Aimi Heft

Notice of Rights

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. For information on getting permission for reprints and excerpts, contact permissions@peachpit.com.

Notice of Liability

The information in this book is distributed on an "As Is" basis, without warranty. While every precaution has been taken in the preparation of the book, neither the author nor the publisher shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

Trademarks

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Peachpit was aware of a trademark claim, the designations appear as requested by the owner of the trademark. All other product names and services identified throughout this book are used in editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this book.

```
ISBN 13: 978-0-321-88765-8
ISBN 10: 0-321-88765-4
```

Printed and bound in the United States of America

987654321

To Sashimi, the best cat ever.

ACKNOWLEDGEMENTS

Thanks to Jeff Riley, Nancy Peterson, Michael Nolan, and the staff at Peachpit for making the book possible and my words less incoherent. Thanks to Nicholas Zakas for his exceptionally detailed and thoughtful criticism about this book and for his mentorship at Yahoo!

Thanks as well to Stoyan Stefanov for his last-minute review and invaluable experience in the world of technical writing. Thanks also to Guy Podjarny for his time and research.

This book wouldn't have been possible without the support of my manager, Ross Harmes, as well as the rest of the front-end team at Flickr.

Thanks to Benjamin for showing me the many uses for mobile devices.

Finally, thank you to Elise for putting up with me while I spent hours every evening staring at my computer and a pile of cell phones.

ABOUT THE AUTHOR

Stephen Woods is a Senior Front-end Engineer at Flickr. He has been developing user interfaces for the web since the end of the last century. He has worked at Yahoo! since 2006. Before Flickr, he developed JavaScript platforms that supported the Yahoo! home page and worked on the UI team at Yahoo! Personals. He's an expert with the full web stack, but his primary interest is making responsive user interfaces with web technologies. Stephen has spoken at SXSW and HTML5DevConf about touch interfaces and he has been published in *.net* magazine.

CONTENTS

	Introduction	viii
	Welcome	X
CHAPTER 1	THE MOBILE LANDSCAPE	2
	The difference between a touch device and a desktop	4
	Devices in the wild	5
	HTML5	8
	The uncanny valley: What makes a touch interface responsive?	? 9
	Wrapping up	11
CHAPTER 2	CREATING A SIMPLE CONTENT SITE	12
	Choosing a philosophy: Mobile first or mobile last	15
	Creating the markup	15
	Laving the groundwork: the <head></head>	
	Understanding the viewport	18
	Responsive CSS	22
	Wrapping up	29
CHAPTER 3	SPEEDING UP THE FIRST LOAD	
	How the browser loads a page	
	Why are pages slow?	32
	Speeding up with YSlow and PageSpeed	36
	Solving common problems	39
	Pulling it all together	47
	Wrapping up	47
CHAPTER 4	SPEEDING UP THE NEXT VISIT	48
	Caching in HTTP	50
	Optimizing for mobile	51
	Using web storage	53
	The application cache	61
	Wrapping up	65
CHAPTER 5	USING PJAX TO IMPROVE THE TOUCH EXPERIENCE	66
-	The price of page loads	68
	The browser history API	70
	Adding PJAX	
	Wrapping up	87
CHAPTER 6	TAPS VS. CLICKS: BASIC EVENT HANDLING	88
	What makes a tap different?	
	Introducing touch events	91

	Handling taps	94
	Wrapping up	102
	Project	103
CHAPTER 7	CSS TRANSITIONS, ANIMATION, AND TRANSFORMS	104
	Animating elements	106
	CSS transforms	122
	Wrapping up	132
	Project	133
CHAPTER 8	MAXIMIZING JAVASCRIPT PERFORMANCE	134
	Performance testing and debugging	136
	The write-only DOM	137
	Prioritizing user feedback	139
	Putting it together: Infinite scroll	140
	Wrapping up	151
	Project	151
CHAPTER 9	THE BASICS OF GESTURES	. 152
	Why gestures?	154
	Creating a progressively enhanced touch control	156
	Building a touch lightbox	163
	Wrapping up	181
	Project	181
CHAPTER 10	SCROLLING AND SWIPING	182
	Scrolling	184
	When layouts fail	192
	Making the bird browser swipeable	195
	Wrapping up	214
	Project	215
CHAPTER 11	PINCHING AND OTHER COMPLEX GESTURES	. 216
	Understanding multi-touch limitations and support	218
	Handling multiple touches	219
	Handling pinches	224
	Wrapping up	237
	Project	237
	Index	238
APPENDIX A	DEBUGGING TOOLS	A:1
APPENDIX B	MOBILE FRAMEWORKS	B:1
		C.1
AFFENDIA	Bonus chapters mentioned in this eBook are available after the	index
	bonas enapters mentionea in this ebook are available after the	much.

INTRODUCTION

As of this writing, 11.42 percent of web visits are via a mobile device (according to Stat-Counter.com). One year ago that was 7 percent. Three years ago it was 1.77 percent. Desktops will be with us for a while, but the future of the web will be on mobile devices.

For web developers, supporting mobile devices is the biggest change since the web standards revolution of the early 2000s. Mobile devices all have HTML5-capable, thoroughly modern browsers. They have limited memory and slow CPUs. They often connect via highlatency connections. Most importantly, they all have touch interfaces.

Developing for mobile is developing for touch. Many of the skills you use for desktop web development carry over to the mobile web, but some things are quite different—and getting those things right can be difficult. I wrote this book to help you get the new things right.

WHO THIS BOOK IS FOR

This book was written for two types of readers:

- Experienced web developers who have never developed for mobile or touch interfaces and want to learn how.
- Developers who've been working in mobile but have struggled to make their mobile websites feel right.

This book is not for absolute beginners. You'll need to have a working knowledge of the web front-end: HTML, CSS, and JavaScript. Prior experience with the new APIs and features of HTML5 and CSS3 won't hurt either.

Most importantly, this book is for people who aren't content with a mobile site that's just good enough. If you want to build a site that feels fast and smooth, this book is for you.

WHAT YOU WILL LEARN

This book is focused on making touch interfaces that feel fast. It's structured in roughly the same way I approach optimizing a website. The first half covers what I consider the basics— concepts that make any website faster, but mobile sites in particular. Chapters 2 and 3 show you how to build a simple site and make it load faster. Chapter 4 helps you speed up users' next visit to the site with caching. Chapter 5 is all about removing page loads all together and structuring applications to maximize real and perceived performance.

The second half of the book is specifically about touch interfaces, in particular making them feel as smooth and fast as possible. The book gets more complex as it goes on. If you feel like the later chapters are over your head, try applying what you've learned so far in your work and then coming back to some of the ideas I'll present toward the end. A website doesn't need pinch to zoom to be useful.

WHAT YOU'LL NEED

To get the most out of this book you'll need at least one touch-enabled device in addition to your computer. If you're only going to have one, I recommend an iOS 6 or Android 4 device. Having both is ideal if you can afford it.

When developing for the mobile web, try to get as many devices as possible. iOS and Android simulators are no substitute for real devices. When writing this book, I used a Samsung Galaxy S III with Android 4.0.4 (Ice Cream Sandwich), an iPhone 4, an iPhone 5, an iPad 1, and an HTC 8X (Windows 8). I supplemented these devices with the simulators.

At Flickr we have a similar set but we also have several Android tablets and a Kindle Fire.

FRAMEWORKS

This book doesn't use jQuery or any other JavaScript framework. You'll learn about a few specialized libraries, but we'll focus as much as possible on native DOM APIs. That's not to say you should avoid frameworks—far from it! But I want to make sure you understand how things really work. When you decide to build a site with jQuery mobile, Backbone.js, Zepto.js, or any other framework, you'll be much more comfortable understanding what's really going on.

The other huge benefit to understanding the native DOM APIs is that when you find a bug or a problem in a library you can patch it yourself and make a pull request with your fixes, benefiting the entire community. Appendix A lists some handy debugging tools.

Appendix B lists a few of the more common mobile-focused frameworks. When you build a new site, I recommend carefully evaluating your needs, including as little library code as you can, and adding only what you need.

The appendices are not printed in the book. They can be found at the book's companion website: touch-interfaces.com

THE WEBSITE

All the code samples in this book as well as late-breaking changes can be found at the companion website: touch-interfaces.com. The code samples are also mirrored on GitHub, where you can file issues with the samples and submit pull requests: https://github.com/saw/touch-interfaces.

WELCOME TO THE MOBILE WEB

Websites are built with HTML, CSS, and JavaScript. Mobile websites are no different. All you really need to get started is a web browser and a text editor, but to be really productive I recommend a few more tools.

THE TOOLCHAIN

The easiest process is to develop with a text editor and a desktop browser, then keep a touch device around for testing.



A TEXT EDITOR & A WEBKIT BROWSER

I use TextMate 2 (github. com/textmate/textmate) for MacOS X, but any editor will do.

Because the vast majority of mobile devices run a WebKit browser, you will find that Chrome or Safari is a an essential tool to being productive. It isn't the same as testing on the real device but it's a lot easier and essential.



A WEB SERVER

In order to test your site on an actual device you will need to serve pages on your local wireless network. On the Mac I find MAMP (www.mamp.info) to be a very convenient tool for this, but using the built-in Apache web server will work as well.



A TOUCH DEVICE

There is no substitute for a physical device. If you can afford it, I recommend having at least a recent Android phone and an iOS device. If you can only afford one phone it's helpful to find people who will let you borrow their phones for a moment to test on.

TESTING ACROSS DEVICES

You can't assume that all WebKit browsers are created equal. You should test your app in iOS 5, iOS 6, Android 2.3, Android 4.0, Android 4.1 (Chrome), and IE 10. Here is a guide to how to test on these devices, even if you don't have access to the device itself.



IOS SAFARI

Apple provides a quite capable simulator with XCode. The simulator can run as iOS 5 or 6 and as a tablet or phone. It also supports remote debugging with Safari. It really is a great tool and assuming you have a Mac this is a critical part of your toolset. XCode is available for free from the Mac App store.



ANDROID

Google provides emulators for just about every version of Android. These are available with the Android SDK (developer. android.com/sdk). Once you have the Android SDK, images for various Android versions are separate downloads. Keep in mind these are the official builds from Google; Android versions on actual devices can vary quite a bit.



WINDOWS 8

Microsoft does provide an emulator for Windows Phone 8; it's available with the SDK (dev. windowsphone.com/ en-us/downloadsdk). The emulator runs only on Windows. IE 10 for the desktop is the same browser, so most debugging can be down with the desktop browser rather than the emulator.



DEBUGGING

Debugging websites on phones can be a chore, but there are a lot of tools available to make it easier. I've provided a list of several on the website in Appendix A.

CHAPTER 4 Speeding Up the Next Visit



So much about computing performance depends on caching. Fundamentally, caching is putting data somewhere after you get it the first time so you can access it much more quickly the next time. On the web, we want to take advantage of caching as often as possible to speed up users' subsequent visits to the site, keeping in mind that their next visit is quite frequently within seconds of their first, when they ask for another page.

On mobile, as much as anywhere, we want to make the best possible use of caching. The main tools we have for caching on touch devices are the normal browser cache, localStorage, and the application cache. In this chapter we'll look at normal browser cache, which isn't as good as it should be; LocalStorage, a newish API for persistent storage that's an incredibly powerful tool for manual caching; and the application cache.

CACHING IN HTTP

HTTP was designed with caching in mind. The cache we're most familiar with is the browser cache, but additional caching *proxies* often exist as well, and they follow the same rules defined in the specification. There are three ways to control HTTP caches:

- Freshness
- Validation
- Invalidation

FRESHNESS

Freshness, sometimes called the TTL (Time To Live), is the simplest. Using headers, caching agents are told how long to hold on to a cached resource before it should be considered stale and refetched. The simplest way this is handled is with the Expires header. You might remember that YSlow and PageSpeed recommend setting far-future Expires headers for static content.

The goal here is that so-called static assets (like CSS and JavaScript) are never fetched again, if possible. YSlow advises that you set an expiration some time in the distant future:

Expires: Thu, 15 Apr 2025 20:00:00 GMT

The intent is that the browser (or a caching proxy) will keep this file around until it runs out of room in cache.

VALIDATION

Validation provides a way for a caching agent to determine if a stale cache is actually still good, without requesting the full resource. The browser can make a request with an If-Modified-Since header. The server then can send a 304 Not Modified response and the browser uses the file already in the cache, rather than refetching from the server.

Another validation feature is the ETag. ETags are unique identifiers, usually hashes, which allow cache validation without dates by comparing a short string. The requesting agent makes a conditional request as well, but this time with an If-None-Match header containing the ETag. If the current content matches the client's ETag, then the server can again return a 304 response.

Validating the cache does require a full round-trip to the server. That is better than redownloading a file, but avoiding a round trip altogether is preferable. That's the reason for the far-future expiration date. If the cached item hasn't expired, then the browser won't attempt to validate it.

INVALIDATION

Browsers invalidate cached items after some actions, the most common being any non-GET request to the same URL.

WHAT IS NORMAL CACHE BEHAVIOR?

So what is the normal behavior of the browser cache, if you don't mess with the headers or do anything else? Most browsers have a maximum cache size. When that size is reached they begin removing items from the cache that were *least recently used*. So a cached item that hasn't been used in a long time will be purged, keeping items used more frequently.

The result of this algorithm is that what is purged is completely based on user behavior and there's no reliable way to predict how it will work. It's safe to assume that if you don't think about cache headers, then some browser will cache something you don't want cached and won't cache something you do.

OPTIMIZING FOR MOBILE

The browser cache is very important on a desktop computer, but not so much on touch devices.

In iOS 5, the browser cache is limited to 100 MB and *does not* persist between app launches. That means that if the phone restarts or the browser is killed or crashes, the entire cache is emptied when the browser starts again. Android 2.x's stock browser (still the most widely installed version by far) has a cache limit of just 5.7 MB, and that isn't per domain—that's total (**Table 4.1**).

OS	BROWSER	MAX PERSISTENT SIZE
iOS 4.3	Mobile Safari	0
iOS 5.1.1	Mobile Safari	0
iOS 5.1.1	Chrome for iOS	200 MB +
Android 2.2	Android Browser	4 MB
Android 2.3	Android Browser	4 MB
Android 3.0	Android Browser	20 MB
Android 4.0-4.1	Chrome for Android	85 MB
Android 4.0-4.1	Android Browser	85 MB
Android 4.1	Firefox Beta	75 MB
BlackBerry OS 6	Browser	25 MB
BlackBerry OS 7	Browser	85 MB

TABLE 4.1 Persistent Cache Size by Browser

* Adapted from research by Guy Podjarny (www.guypo.com)

It's very important to optimize the cacheability of your site. But the very limited size of the browser caches means that users will very often come to your site with an empty cache, so optimizing for that state should not be neglected.

A good header for a static resource looks something like this:

HTTP/1.1 200 OK Content-Type: image/png Last-Modified: Thu, 29 Mar 2012 23:53:57 GMT Date: Tue, 11 Sep 2012 21:36:44 GMT Expires: Wed, 11 Sep 2013 21:36:44 GMT Cache-Control: public, max-age=31536000

Cache-Control: public makes sure that SSL resources can be cached by proxies. The max-age is one year (in seconds). The Expires date is also a year in the future.

In practice, it's a good idea to read up on how to configure your particular server so that the headers are correct. If you're working with separate back-end developers, gently remind them how important these values are.

For the actual content many major sites use cache-control: private to prevent any caching by proxies. For the Birds of California site, the content won't change that much, so on the server we can set up the cache headers to expire in one hour. We're using Nginx, so we can do that with the expires directive:

```
location / {
    expires 1h;
}
```

This results in a header that looks like this, assuming the site was accessed at 05:16:45 PST:

```
Last-Modified: Thu, 05 Jul 2012 17:15:35 PST
Connection: keep-alive
Vary: Accept-Encoding
Expires: Wed, 14 Nov 2012 06:16:46 PST
Cache-Control: max-age=3600
```

This prevents mobile users from refetching content too much during a browsing session, but ensures that the content is fresh, even for desktop users.

Another important thing to consider is web accelerators like Amazon Silk. Silk is the browser for the Kindle Fire tablets. Unlike a normal browser, Amazon Silk is a browser that lives both on the Kindle Fire and on Amazon servers. According to Amazon, much of the acceleration comes from pipelining and "predictive push," which means sending static resources to the browser before the browser even requests the resource. In this case Silk acts as a transparent HTTP proxy. A proxy may cache just like the browser, and it follows the same rules. So by sending the correct headers you're also improving performance for Kindle users.

USING WEB STORAGE

Browser makers, and Apple in particular, have left us with a less than ideal situation when it comes to the browser cache. But they and the W₃C have given us something else that almost makes up for it: the web storage API. Web storage provides a persistent data store for the browser, in addition to cookies. Unlike cookies, 5 MB is available per domain in a simple key-value store. On iOS, WebStorage stores the text as a UTF-16 string, which means that each character takes twice as many bytes. So on iOS the total is actually 2.5 MB.

USING THE WEB STORAGE API

Web storage is accessed from two global variables: localStorage and sessionStorage. session-Storage is a nonpersistent store; it's cleared between browsing sessions. It also isn't shared between tabs, so it's better suited to temporary storage of application data rather than caching. Other than that, localStorage and sessionStorage are the same.

Just like cookies, web storage access is limited by the same origin policy (a web page can only access web storage values set from the same domain) and if users reset their browsers all the data will be lost. One other small issue is that in iOS 5, web views in apps stored their web storage data in the same small space used for the browser cache, so they were hardly persistent. This has been fixed in iOS 6.

NOTE LocalStorage should not be treated as secure. Like everything, the user can read and modify what is in localStorage.

The web storage API is very simple. The primary methods are localStorage. getItem('key'); localStorage.setItem('key', 'value'). key and value are stored as strings. If you try to set the value of a key to a non-string value it will use the JavaScript default toString method, so an object will just be replaced with [object object].

Additionally you can treat localStorage as a regular object and use square bracket notation:

```
var bird = localStorage['birdname'];
```

```
localStorage['birdname'] = 'Gull';
```

Removing items is as simple as calling localStorage.removeItem('key'). If the key you specify doesn't exist, removeItem will conveniently do nothing.

In addition to storing specific information, localStorage is a great tool for caching. In this next section, we'll use the Flickr API to fetch a random photo for Birds of California, and use localStorage as a transparent caching layer to greatly improve the performance of the random image picker on future page loads.

USING WEB STORAGE AS A CACHING LAYER

For the Birds of California site, we can make things a little more exciting for users by incorporating a random image from Flickr, rather than a predefined image. This will sacrifice some of the gains we made in the last chapter in trimming images down to size, in exchange for developer convenience.

We'll use the Flickr search API to find Creative Commons–licensed photos of birds. Listing 4.1 is a simple JavaScript Flickr API module that uses JSONP to fetch data. For the sake of brevity the code is not included here, but it's available for download from the website. Let's use this module to grab some images related to the California Gull.

LISTING 4.1 Fetching the Flickr data

```
//a couple of convenience functions
var $ = function(selector) {
  return document.querySelector(selector);
};
var getEl = function(id) {
  return document.getElementById(id);
};
var flickr = new Flickr(apikey);
var photoList;
flickr.makeRequest(
   'flickr.photos.search',
  {
     text: 'Larus californicus',
     extras:'url z,owner name',
     license:5,
     per page:50
  },
  function(data) {
     photoList = data.photos.photo;
     updatePhoto();
  }
);
```

As you can see, the API takes a method (flickr.photos.search) and some parameters. This will hopefully give us back as many as 50 photos of *Larus Californicus*.

In **Listing 4.2**, the updatePhoto function takes the list, grabs a random photo from the list, and updates the image, the links, and the attribution.

```
LISTING 4.2 Updating the photo
  function updatePhoto() {
     var heroImg = document.querySelector('.hero-img');
     //shorthand for "random member of this array"
     var thisPhoto = photoList[Math.floor(Math.random() * photoList.length)];
     $('.hero-img').style.backgroundImage
             = 'url('+ thisPhoto.url z + ')';
     //update the link
     getEl('imglink').href =
        'http://www.flickr.com/photos/' +
        thisPhoto.owner +
        '/'+ thisPhoto.id;
     //update attribution
     var attr = getEl('attribution');
     attr.href = 'http://www.flickr.com/photos/'
          + thisPhoto.owner;
     attr.innerHTML = thisPhoto.ownername;
```

}

Add this script (with the Flickr API module) to the Birds of California page with a valid Flickr API key and the bird hero image will dynamically update to a random option from the search results list. With no changes to the HTML and CSS from before, however, the user will see the original gull photo, and then a moment later it will be replaced with the result from the API. On one hand, this provides a fallback in case of JavaScript failure for the image. But on the other hand, it doesn't look very nice, and we'll go ahead and say the image is an enhancement to the main content, which is the text.

With that in mind, let's create a null or "loading" state for the links and caption, as shown in **Listing 4.3**.

```
LISTING 4.3 Hero image null state
<div class="hero-shot">
<a id="imglink" href="#">
<span class="hero-img"></span></a>
Photo By <a id="attribution" href="#">...</a>
</div>
```

While the data is loading the user needs some indication that something's happening, just so she knows things aren't broken. Normally a spinner of some kind is called for, but in this case let's just add the text "loading" and make the image background gray until it's ready:

```
//show the user we are loading something....
var heroImgElement = $('.hero-img');
heroImgElement.style.background = '#ccc';
heroImgElement.innerHTML = 'Loading...';
```

```
//Then inside updatePhoto I'll remove the loading state:
heroImgElement.innerHTML = '';
```

So, now we have a pretty nice random image, with a loading state (**Figure 4.1**). However, we're making users wait every time they visit for a random image from a list that probably doesn't change that much. Not only that, but having a very up-to-date list of photos isn't all that important because we just want to add variety, not give up-to-the-minute accurate search results. This is a prime candidate for caching.

ntil Verizon 📀	16:27	_	-	
Birds	of Californ	ia		
stephenwoods.net	t/chapt 🖒	Google		
Birds of California!				
California Condor C	alifornia Qua	il Californ	nia Gull	
California Gull				
Photo By				
The California Gu a gull native to the The gull looks a b distinguishing cha black ring on a ye a rounder head.	II (Larus c e western it like the racteristic llow bill, y	alifornicu United S Herring (s include ellow leg	is) is tates. Gull. e a s and	
Adults have gray	and white	bodies w	/ith	
		<u> </u>	G	

FIGURE 4.1 The loading state.

If something is cacheable, it's generally best to abstract away the caching, otherwise the main logic of the application will be cluttered with references to the cache, validation checks, and other logic not relevant to the task at hand. For this call we'll create a new object to serve as a data access layer, so that rather than calling the Flickr API object directly, we'll call the data layer, like so:

```
birdData.fetchPhotos('Larus californicus', function(photos) {
  photoList = photos;
  updatePhoto();
```

});

Because all we ever want to do is search for photos and get back a list, we can hide the Flickr-specific stuff inside this new API. Not only that, but by creating a clean API we can, in theory, change the data source later. If we decide that a different API produces better photo results, we can change the data layer without making changes to any consumers. In this case the key feature is caching. We want to cache API results locally for one day, so that the next time the user visits she'll still get a random photo, but she won't have to wait for a response from the Flickr API.

CREATING THE CACHING LAYER

The fetchPhotos method will first check if this search is cached, and whether the cached data is still valid. If the cache is available and valid, it will return the cached data, otherwise it will make a request to the API and then populate the cache after firing the callback.

First, we'll set up a few variables, as shown in Listing 4.4.

```
LISTING 4.4 A caching layer
```

```
window.birdData = {};
var memoryCache = {};
var CACHE TTL = 86400000; //one day in seconds
var CACHE PREFIX = 'ti';
```

The memoryCache object is a place to cache things fetched from localStorage, so if those items are requested again in the same session they can be returned even faster; fetching data from localStorage is much slower than simply getting data from memory, without including the added cost of decoding a JSON string (remember, localStorage can only store strings). We'll talk more about CACHE PREFIX and CACHE TTL shortly.

The first thing we need is a method to write values into cache. We'll cache the response from the Flickr search, but wrap the cached value inside a different object so we can store a timestamp for a cache so that it can be expired.

```
function setCache(mykey, data) {
  var stamp, obj;
  stamp = Date.now();
  obj = {
    date: stamp,
    data: data
  };
  localStorage.setItem(CACHE_PREFIX + mykey, JSON.stringify(obj));
  memoryCache[mykey] = obj;
}
```

We're using CACHE_PREFIX for each of the keys to eliminate the already small chance of collisions. It's possible that another developer on the Birds of California site might decide to use localStorage, so just to be on the safe side we'll prefix our keys. The date value contains a timestamp in seconds, which we can use later to check if the cache has expired. We'll also add the value to the memory cache for quicker access to it if it's fetched again during the same session. We'll use the "setItem" notation for localStorage; this is much clearer than bracket notation—another developer will see right away what is happening, rather than thinking that this is a regular object.

The next function is getCached, which returns the cached data if it's available and valid, or false if the cache is not present or expired (the caller really doesn't need to know which):

```
//fetch cached date if available,
//returns false if not (stale date is treated as unavailable)
function getCached(mykey) {
  var key, obj;
  //prefixed keys to prevent
  //collisions in localStorage, not likely, but
  //a good practice
  key = CACHE_PREFIX + mykey;
  if(memoryCache[key]) {
    if(memoryCache[key].date - Date.now() > CACHE_TTL) {
      return false;
    }
}
```

```
return memoryCache[key].data;
}
obj = localStorage.getItem(key);
if(obj) {
    obj = JSON.parse(obj);
    if (Date.now() - obj.date > CACHE_TTL) {
        //cache is expired! let us purge that item
        localStorage.removeItem(key);
        delete(memoryCache[key]);
        return false;
    }
    memoryCache[key] = obj;
    return obj.data;
}
```

This function checks the cache in layers. It starts with the memory cache, because this is the fastest. Then it falls back to localStorage. If it finds the value in localStorage, then it makes sure to also put that value into the memoryCache before returning the data. If no cached value is found, or one of the cached values has expired, then the function returns false.

Next up is the actual fetchPhotos function that encapsulates the caching. All it has to do now is fetch the cached value for the query. If that value is false, then it executes the API method and caches the response. If it is true, then the callback function is called immediately with the cached value.

```
// function to fetch CC flickr photos,
// given a search query. Results are cached for
// one day
function fetchPhotos(query, callback) {
  var flickr, cached;
  cached = getCached(query);
  if(cached) {
    callback(cached.photos.photo);
  } else {
    flickr = new Flickr(API KEY);
```

```
flickr.makeRequest(
        'flickr.photos.search',
        {text:query,
         extras:'url z,owner name',
         license:5,
         per page:50},
        function(data) {
          callback(data.photos.photo);
          //set the cache after the
          //callback, so that it happens after
             //any UI updates that may be needed
          setCache(query, data);
        }
     );
  }
}
```

window.birdData.fetchPhotos = fetchPhotos;

Now the data call is fully cacheable, with a simple API.

MANAGING LOCALSTORAGE

This is just the beginning for localStorage. Unlike the browser cache, localStorage gives you full manual control. You can decide what to put in, when to take it out, and when to expire it. Some websites (like Google) have actually used localStorage to cache JavaScript and CSS explicitly. It's a powerful tool, so powerful that 5 MB starts to feel a little small sometimes. What do you do when the cache is full? How do you know if the cache is full?

First of all, we can treat localStorage as a normal JavaScript object, so JSON.stringify (localStorage) will return a JSON representation of localStorage. Then we can apply an old trick to figure out how many bytes that uses, including UTF-8 multi-byte characters: unescape(encodeURIcomponent('string')).length, which gives us the size of string in bytes. We know that 5 MB is 1024 * 1024 * 5 bytes, so the available space can be found with this:

1024 * 1024 * 5 - unescape(encodeURIComponent(JSON.stringify(localStorage))). \rightarrow length If you want to know if you've run out of space, WebKit browsers, Opera mobile and Internet Explorer 10 for Windows Phone 8 will throw an exception if you've exceeded the available storage space; if you're worried, you can wrap your setItem call in a try/catch block. When you've run out of storage you can either clear all the values your app has written with localStorage.clear, or keep a separate list in localStorage of all the data you cache and intelligently clear out old values.

THE APPLICATION CACHE

The traditional browser cache, as mentioned previously in this chapter, isn't particularly reliable on mobile. On the other hand, the HTML5 application cache is very reliable on mobile—maybe even too reliable.

WHAT IS THE APPLICATION CACHE?

With features like localStorage, you can easily see how a web application could continue to be useful even when not connected to the network. The application cache is designed for that use case.

The idea is to provide a list of all the resources your app needs to function up front, so that the browser can download and cache them. This list is called the manifest. The manifest is identified with a parameter to the <html> tag:

```
<!DOCTYPE html>
<html manifest="birds.appcache">
<head>
```

This file *must* be served with the mime-type text/cache-manifest. If it's not, it will be ignored. If you can't configure a custom mime type on your server, you can't use the application cache.

The manifest contains four types of entries:

- MASTER
- CACHE
- NETWORK
- FALLBACK

MASTER

MASTER entries are the files that reference the manifest in their HTML. By including a manifest, these files are implicitly adding themselves to the list. The rest of the entries are included in the manifest file.

CACHE

The CACHE entries define what to cache. Anything in this list *will be* downloaded the first time a visitor comes to the page. The entries will then be cached forever, or until the manifest (not the resource in question) changes.

NETWORK

Because the application cache is designed for offline use, network access actually has to be whitelisted. That means that if a network resource is not listed under network it will be blocked, even if the user is online. For example, if the site includes a Facebook "like" widget inside an iframe, if http://www.facebook.com is not listed in the NETWORK entry, *that iframe will not load.* To allow all network requests you can use the '*' wildcard character.

FALLBACK

These entries allow you to specify fallback content if the user is not online. Entries here are listed as pairs of URLs: the first is the resource requested, the second is the fallback. You have to use relative paths, and everything listed here has to be on the same domain. For example, if you serve images from a CDN on a separate domain you can't define a fallback for that.

CREATING THE CACHE MANIFEST

Here's a manifest for the Birds of California site from the previous chapter:

CACHE MANIFEST

```
# Timestamp:
# 2013-03-15r1
CACHE:
jquery-1.8.0.min.js
gull-360x112.jpg
gull-640x360.jpg
gull-720x225.jpg
```

FALLBACK:

NETWORK:

*

Notice that there are entries for all the different images. Because these are explicit, the browser will download and cache all of them on the first visit to the page, but will never again need to fetch them.

PITFALLS OF THE APPLICATION CACHE

The application cache is the nuclear option. That's because the files in here will *never* expire until the manifest file itself changes, the user clears the cache, or the cache is updated via JavaScript (more on that later). That's why we included a timestamp in the manifest so we can easily force a change to the file if we want to invalidate cached versions in the wild.

The application cache is also completely separate from the browser cache. For example, it is possible to create an application cache that will never revalidate. If you set a far-future Expires header on the manifest file, the browser will cache that file forever. When the application cache checks whether it has changed, it will get the version in the browser cache, see that it is unchanged, and then hold on to the cached files forever (or until the user explicitly clears her cache).

Once the page is cached, it's possible to visit Birds of California without network connectivity. On iOS, offline is guaranteed to work only if the user has bookmarked the page on her home screen. In iOS Safari the contents of the application cache may be evicted if the browser needs to reclaim the space for the browser cache. The cache will still be used.

One of the other pitfalls of the application cache is that once it expires it won't be updated until the next time the user visits. So if a user comes to your site with a stale cache, she'll still see the cached version, even though it's been updated. To make sure users get the latest and greatest bird info, we'll take advantage of the application cache JavaScript API to programmatically check for a stale cache.

AVOIDING A STALE CACHE WITH JAVASCRIPT

The API for the cache hangs off the window.applicationCache object. The most important property there is "status." As shown in **Table 4.2**, it has an integer value that represents the current state of the application cache.

CODE	NAME	DESCRIPTION
0	UNCACHED	The cache isn't being used.
1	IDLE	The application cache is not currently being updated.
3	CHECKING	The manifest is being downloaded and updates are being made, if available.
4	UPDATEREADY	The new cache is downloaded and ready to use.
5	OBSOLETE	The current cache is stale and cannot be used.

 TABLE 4.2
 Application Cache Status Codes

Thankfully, you don't have to remember these numbers; there are constants on the applicationCache object that keep track of the association:

```
> console.log(window.applicationCache.CHECKING)
```

2

On the Birds of California site, we'll add a short script to check the cache every time the page loads:

```
//alias for convenience
var appCache = window.applicationCache;
```

```
appCache.update();
```

This goes at the bottom of page and doesn't need to be ready for the window onload event to do its stuff. At this point we could start polling appCache.status to see if a new version is loaded. When it's calling the swapCache method, it will force the browser to update the changed files in the cache (it will not change what the user is seeing; a reload is still required). It's simpler to use the built-in events that the applicationCache object provides. We can add an event handler to automatically reload the page when the cache is refreshed:

```
var appCache = window.applicationCache;
```

```
appCache.addEventListener('updateready', function(e) {
```

```
//let's be defensive and double check the status
if (appCache.status == appCache.UPDATEREADY) {
```

```
//swap in the new cache!
appCache.swapCache();
```

```
//Reload the page
window.location.reload();
```

```
}
```

});

appCache.update();

In addition to the extremely useful "updateready" event, there's a bigger set of events available on the applicationCache object, one for each state we already saw in the status property.

Having the page automatically reload, particularly when the user is in the middle of looking at the site, is a terrible user experience. There are several ways to handle this. Using a confirm dialog box or whisper tip to ask the user to reload to fetch new content is better, but still not great. In the next chapter we'll explore a much better way to handle this, and other cases, by dynamically updating the content with AJAX.

THE 404 PROBLEM

If any of the resources in the CACHE entry can't be retrieved when the browser attempts to fetch them, the browser ignores the cache manifest. This means that if a user visits your site and for some reason one of the requests fails, it will be as if she were a completely new visitor the next time she visits—the cache will be useless. That means the cache is quite brittle: unless all the requests are successful, there's no caching at all—it's all or nothing.

THE APPLICATION CACHE: WORTH THE PAIN?

The application cache is obviously fraught with difficulties, not the least of which is how difficult it is to invalidate. It gives you a lot of power, but at the cost of flexibility and maintainability. Users love an app that launches instantly, but everyone hates strange errors. The stickiness of the application cache leads necessarily to strange bugs that are hard to chase down. When you use it, you'll eventually end up with a file that you just can't seem to get out of cache. It isn't that the application cache is buggy; it's that it's completely unforgiving. If you deploy a bad cache, it can be a real problem to undo the error.

Optimizing for browser cache and using the much more flexible web storage API is usually a better choice, but when you want the fastest possible launch time, and you're willing to accept the difficulties, the application cache is an incredible tool.

WRAPPING UP

Caching is one of the most powerful tools for optimizing performance. It's one of the basics that you really want to get right before you move on to more complex optimizations. In this chapter we covered the fundamentals of the browser cache and some simple optimization strategies. We discussed web storage and using it for caching data. Finally we talked about the application cache, which is powerful, if a bit finicky.

In the next chapter we'll look at how to work around the overhead of page loads entirely with PJAX.

FURTHER READING

The complete APIs for web storage and the application cache are well covered on the Mozilla Developer Network:

- https://developer.mozilla.org/en-US/docs/DOM/Storage
- https://developer.mozilla.org/en-US/docs/HTML/Using_the_application_cache

INDEX

NUMBERS

2D transforms matrices 130 overview of 125–128 3D transforms matrices 131 overview of 125–128

Α

Absolute positioning, in bird browser example 195-198 ActionScript 3.0 x addRoute method 79-80 Adobe Air x Adobe Flash Builder xi Adobe Flash Catalyst xi Adobe Flash Player х Adobe Flash Professional, in Creative Suite 5.5 x Adobe Flex xi AJAX 68-70 all keyword, CSS transitions and 107 Amazon Silk browser on Kindle Fire 6 role of web accelerators in optimizing browser caching 51 Android Browser multi-touch support 218 native scrolling in 184-185 optimizing caching for 51 overflow in Android 2 189-190 pushState support 73-74 WebKit browser engine in 8 zooming impacting fixed layouts 194-195 Android Chrome default browser on Android devices 6 frames timeline tool 120-121 high-density displays and 22 multi-touch support 218 optimizing caching for 51 pushState support 73-74 testing performance with console.time() 136 WebKit browser engine in 8 Android OS, operating systems for touch devices 5-6

Animation

compositing 122 creating hidden images 107 creating using cubic Bézier timing functions and vendor prefixes 114-116 CSS animation 112, 146-147 CSS transforms 122-125 CSS transitions 106-107 debugging 120-122 deferred loading 146-148 of elements 106 good and bad 105 hardware acceleration and 125-129 JavaScript animation 117 JavaScript transitions 108-112 keyframes 113 requestAnimationFrame 117-120 smoothness of 120 subproperties 113-114 swapping CSS styles 108 transformation matrices 130-132 wrap up 132 APIs, web storage 53 Apple high-density displays and 22 iOS, see iOS UI conventions 11 Apple Touch Interface Guidelines 10 Application cache avoiding stale cache 63-65 creating CACHE MANIFEST 62 overview of 61-62 pitfalls of 63 when to use 65 <aside> tags, in content site example 16

В

Bandwidth, of network connections 35 Bézier curves creating CSS animation using cubic Bézier timing functions and vendor prefixes 114-116 for ease-in/ease-out 110-112 Blackberry OS 5-6 Blocking rendering DOM operations and 138 load time and 35

<body> tags, in content site example 16-17 Bouncing ball markup animation using cubic Bézier timing functions and vendor prefixes 114-116 keyframes for 113 subproperties 113-114 transition example with JavaScript 108-110 transitions and 110-112 Breakpoints creating 25 CSS styles for 27-29 media queries 25-27 Browser history API hash-bang urls and 77 load avoidance and 70 - 71Browsers. see Web browsers build function 203 buildChrome function, for lightbox 171-172 Byte count load time and 35 reducing with Closure Compiler 68

С

CACHE entries, in manifest of application cache 62 Cache/caching application cache 61-65 CDN (content delivery network) and 37 creating wrapper for data calls to allow insertion of caching laver 201-202 the DOM 137-138 in HTTP 50-51 load time and 35 localStorage 60-61 mobile browsers and 40-41 optimizing 51-52 overview of 49 PageSpeed tool and 38 slide data 148 web storage API 53 Web storage as caching layer 54-60 "cargo cults," 136 CDN (content delivery network) 37

Center point, of layout 233 Charles proxy (charlesproxy.com), troubleshooting load time 39-40 Chrome. see Android Chrome Click events handling for lightbox 172 intercepting 81 slides and 173 taps compared with 89-90 Client-side templating languages 87 Closure Compiler, for reducing byte count 68 Compositing, in animation 122 console.time() fetch data function and 143 testing performance with 136 Constructor, for initializing touch lightbox 169-171 Content delivery network (CDN) 37 Content site breakpoints 25-27 CSS styles 22-25 CSS styles for breakpoints 27 - 29<head> tag 17-18 high-density displays 22 markup for 15-17 mobile first vs. mobile last philosophies 15 overview of 13-15 viewports 18 virtual pixels on viewport 19-22 wrap up 29 Conventions touch 154-155 UI 11 Cookies, impact on size of HTTP requests 35 Critical path, load time and 41 CSS (Cascading Style Sheets) absolute positioned styles 195-198 animation 112, 114-116, 146-147 applying styles to thumbnails 164-167 combining CSS files to reduce number of requests 40-41 creating content sites 14 creating styles for breakpoints 27-29 CSS3 in HTML5 suite 9 em and px units in CSS2 and CSS3 20 fixed position layout 186-189 media queries in CSS3 25 - 27mobile first and 15 for pinch to zoom 229 responsive CSS files 22 - 25

swapping CSS styles 108 swapping styles 108 transform-origin in 225 transforms 122-125 transitions 106-107 using for interfaces 140-142 cubic-bézier curve, for ease-in/ ease-out 110-112 currentSlide function 207

D

Data data model for bird browser 199-201 fetching 143-144 web storage 53 Debugging animation 120-122 Deferred loading of animated images 146-148 of images 143-145 Design, mobile first and 15 Desktops vs. mobile devices 15 vs. touch devices 4-5 Development, mobile first 15 device-height, viewports and 20-21 device-pixel-ratio, specifying with media queries 46 device-width, viewports and 20-22 Dilation, geometry of 226-227 "direct manipulation," touch interface and 10 <div> tags creating content site 16-17 creating slide template 173-174 creating thumbnail HTML 164 scrollable 185 DNS (Domain Name Service) 32 DOM (Document Object Model) caching 137-138 creating content sites 14 creating custom event interface with DOM Level 3 97-99 facades for event handling 99-102 optimizing DOM insertion 150-151 pruning DOM nodes when no longer useful 86 slowness of interaction with 137-138 Domain Name Service (DNS) 32 Domain names, resolving 32

Е

Ease-in/ease-out transitions 110-112 Edge cache, CDN (content delivery network) and 37 Elements animation of 106 labeling 186 scaling 235-236 visibility of 148-150 em units, in CSS2 and CSS3 20 endPos variable, handling touch events 178 ETags, validation feature for HTTP cache 50 Event handling creating synthetic tap event 97-102 facades around DOM for 99-102 IE10 (Internet Explorer 10) and 93-94 overview of 89 popstate event 72-73 properties of touch objects 92 for taps 94-97 taps vs. clicks 89-90 for touch lightbox example 172 types of touch events in mobile browsers 91 wrap up 102 Event listeners attaching to touchstart event 96-97 for intercepting click events 81 for MSPointer events on bird browser 212 - 214for touch events 160-161 for touch events on bird browser 208-211 Events attaching touch events to touch lightbox 177 click events. see Click events creating custom event interface with DOM Level 3 97-99 pointer events. see Pointer events, IE10 popstate event 72-73 replacing scroll event with timer 145-146 tap events. see Tap events touch events. see Touch events

F

Facades, around DOM for event handling 99-102 Fade in/fade out transitions, CSS 108 FALLBACK entries, in manifest of application cache 62 Feedback adding gesture support to lightbox 176 adding to light switch example 161-163 animation and 105 prioritizing 139 touch interface and 10-11, 90 types of gestures 153 Fetching data 143-144 Files byte count and load time 35 tips for downloading large 34 Firefox Beta 51 Firefox Mobile mobile browsers 8 multi-touch support 218 Fixed positioning CSS for fixed position layout 186-189 orientation changes and 192 pinching and 224 scrolling and 185-186 zooming impacting fixed layouts 194-195 Flags, for delaying a task in JavaScript 139 Flickr automatically adjusting image size 42 infinite scrolling of slides from 140 photo download speed 68 Form factors for touch devices 7 for touch devices vs. desktops 4 FPS (frames per second) frame rate in animation 117 smoothness of animation 120-121 and Frames per second (FPS) frame rate in animation 117 smoothness of animation and 120-121 Frames timeline tool, in Chrome 120 - 121Frameworks, single-page 87 Freshness, of HTTP cache 50

G

Geometry, of dilation 226-227 Gestures conventions for 154-155 creating progressively enhanced touch control 156-159 disabling native 219 listening to touch events 160-161 multiple. see Multi-touch overview of 153 pinching. see Pinching progressive enhancement of 155 reasons for 154 "snapping back," 162-163 user feedback and 11, 161–162 Gestures, in lightbox example adding chrome 171-172 adding gesture support 176–177 applying styles to thumbnails 164-167 attaching touch events 177 browser normalization and 168 building slides 174-175 constructor for initializing 169-171 creating slide template 173-174 disabling native 163 event handler 172 handling touch events 178-181 moving slides into position 175-176 thumbnail HTML for 164 utility functions 168-169 wrap up 181 goTo function, for moving slides into position 175-176, 207-208 GPUs (graphic processing units) hardware acceleration and 125-126 limitations of hardware acceleration 126-129 scrubbing transforms freeing up graphics memory 129 Graphic interface, in original Macintosh 89 Graphic processing units. see GPUs (graphic processing units) Graphics acceleration hardware in iOS devices 125–126 limitations of 126-129 overview of 7

Н

HandleDefer() caching slide data 148 deferring loading of animated images 146–147 deferring loading of images 143–145 handleRoute method 79-80 Hardware acceleration. see Graphics acceleration hardware Hash-bang urls 77 <head> tag, in creation of content site 17 - 18"Hiccups," animation-related issues 105 Hidden images, creating 107 Hide pages function 85 High-density displays 22,46-47 History, browser history API 70-71 HTML creating thumbnail HTML 164 for pinch to zoom 228 HTML fragments building bird browser with 195 creating 78 HTML5 application cache 61 overview of 8-0 HTTP cache/caching in 50-51 downloading multiple requests and 34 number of HTTP requests and load time 34-35 http-equiv meta property 17-18

I

Identity matrix, in CSS 130-131 IE10 (Internet Explorer 10) acting on touch movements 222-223 adding handlers for MSPointer events 212 - 214gestures 153 landscape orientation 194 managing start of a touch 221-222 multi-touch support 218 native scrolling 184–185 overview of 7-8 pointer events 93-94 pushState support 73-74 throwing exception when exceed storage space 61 Images dealing with load issues related to image size 42-47 deferred loading of animated images 146-148 deferring loading of 143-145 fetching data containing 143 Infinite scrolling creating 142-143 markup for 141-142 overview of 140

init function constructor for initializing lightbox 170-171 converting array into an object 200 Initialization constructor for initializing lightbox 169-171 of pinch to zoom layout 230-232 of swipeable bird browser interface 208-211 Internet Explorer 10. see IE10 (Internet Explorer 10) Invalidation, controlling HTTP cache 50 iOS multi-touch support 218 native scrolling and 184-185, 189 operating systems for touch devices 5-6 pushState support 73-74 Safari. see Safari zooming impacting fixed layouts 194-195 IP addresses 32 iPhone feedback in 00 as first modern touch device 89 multiple touch capacity 217 power and speed 4 smoothness of animation in 125 iScroll 4 191-192

J

"Jank," animation-related issues 105 **JavaScript** animation 117 avoiding stale cache 63-65 blocking rendering while waiting for execution of 35 for lightbox example 168 transitions 108–112 JavaScript, maximizing performance of caching slide data 148 caching the DOM 137-138 checking element visibility 148-150 creating infinite scroll 142-143 deferring loading of animated images 146-148 deferring loading of images 143-145 function for fetching data 143-144 infinite scrolling 140 interacting with DOM and 137 making user feedback a priority 139 optimizing DOM insertion 150-151 overview of 135 replacing scroll event with timer 145-146

testing using console.time() 136 testing using JSPerf.com website 136-137 using templates and CSS for interface 140-142 wrap up 151 "Jitter," animation-related issues 105-106 Jobs, Steve 217 iQuery blocking rendering while waiting for execution of 41 creating custom event interface 97 facades around DOM for event handling 99-102 JSPerf.com website, for performance testing 136-137

Κ

Keyframes 113 @keyframes rule 113 Kindle Fire Android OS and 6 HTTP proxy improving performance 52 Konqueror 8

L

Landscape orientation, handling orientation changes 192-194 Latency load time and 35 speed of light and 37 Lavouts correcting pinch to zoom layout 230-232 creating for pinch to zoom 227-229 issues with mobile 192 orientation changes 192-194 zooming impacting fixed layouts 194-195 tags, in content site example 17 Libraries, reasons for avoiding use of 177 Life cycle functions, creating 221 Light switch example adding user feedback 161-163 building power switch control 156 CSS styles for 156-159 listening to touch events 160-161 Lightbox example adding gesture support 176-177 applying styles to thumbnails 164-167 attaching touch events 177 browser normalization and 168

building 168 building chrome for 171-172 building slides for 174-175 constructor for initializing 169-171 creating slide template 173-174 creating thumbnail HTML 164 creating utility functions for 168–169 event handler for 172 handling touch events 178-181 moving slides into position 175-176 overview of 163 Load time, speeding up first load combining CSS files to reduce number of requests 40-41 critical path and 11 how browsers load a page 32 image size and 42-47 overview of 31 PageSpeed tool 37-39 reasons for slow loading 32-35 solving common problems 39-40 speeding up 36 wrap up 47 YSlow tool 36-37 Load time, speeding up subsequent visits application cache 61-65 caching in HTTP 50-51 local storage 60-61 optimizing for mobile browsers 51-52 overview of 49 web storage 54-60 web storage API 53 wrap up 65 Loads, avoiding benefits of PJAX over AJAX 68-70 browser history API 70-71 creating HTML fragments 78 creating routers 79-81 enabling 78 hide pages function 85 implementing a route 81-83 intercepting click events 81 overview of 67 page handler function 85-86 popstate events 72-73 price of page loads and 68 pruning DOM nodes 86 pushState for changing browser history 71-72 pushState support in mobile browsers 73-74 replaceState for upgrading/ downgrading URLs 75-77 templates 83-85 localStorage, for managing cache 60 - 61localStorage variable, web storage API 53

Μ

Manifest application cache 61 creating CACHE MANIFEST 62 including timestamp to handle expiration of cache 63 MASTER entries, in manifest of application cache 61 Matrices, transformation matrices 130-132 Media queries CSS styles for 25 - 27dealing with load issues related to image size 43 handling orientation changes 193-194 specifying device-pixel-ratio 46 syntax 40 meta property, viewports 163, 185, 228 Mobile browsers. see also Web browsers cache/caching and 40-41 optimizing caching for 51-52 pushState support 73-74 touch events 91 types of 7-8 Mobile devices. see also Touch devices, introduction to power and speed 4 web browsers and touch interfaces 3 Mobile first approach, to creating content sites 15 Mobile First (Wroblewski) 15 Mobile last approach, to creating content sites 15 Model-View-Controller (MVC) 87 Mouse, touch interface compared with 89 mousedown events clicks and 90 touch devices and 91 mouseup events clicks and 90 touch devices and 91 Mozilla Development Network 26 MSPointer events. see Pointer events Multi-touch. see also Pinching acting on touch movements 222-223 browser support for 218 laying groundwork for 219-221 managing end of touch life cycle 223-224 managing start of a touch 221-222 overview of 218 wrap up 237 MVC (Model-View-Controller) 87

Ν

Native scrolling 184–185 <nav> tags, in content site example 16 Navigation, slide class for navigating between pages 202–205 NETWORK entries, in manifest of application cache 62 Network tab, Safari 33 nextSlide function, bird browser example 206–207

0

Objects converting array into 200 creating for circles 220-221 properties of touch objects 92 onclick events 91 Opera Mobile overview of 7-8 throwing exception when exceed storage space 61 Operating systems (OSs), for touch devices 5-7 Orientation handling changes in 192-194 handling for all browsers 211-212 OSs (operating systems), for touch devices 5-7 overflow:auto iScroll used for overflow 191-192 native scrolling and 185.189 overthrow used for overflow 190 overview of 189-190 Overthrow, for adding overflow 190

Ρ

Page handler function 85-86 Page loads avoiding. see Loads, avoiding speeding up first load. see Load time, speeding up first load speeding up subsequent loads. see Load time, speeding up subsequent visits PageSpeed tool expiration settings for HTTP cache 50 troubleshooting load time 37-39 Parallelism JavaScript and 139 parallel download support 34 Performance animation issues 105 debugging animation 120-122 HTTP proxy improving performance of Kindle Fire 52

JavaScript. see JavaScript, maximizing performance of testing 136-137 Web Inspector for diagnosing 33 Phones dealing with load issues related to image size 45-46 in iPhone. see iPhone smartphones 5 touch device form factors 4,7 Windows Phone 8 194 Photo viewers, support for 176 Pinch to zoom creating layout for 227-229 implementing 227 initializing and correcting layout for 230-232 Pinching creating layout for pinch to zoom 227-229 geometry of dilation and 226-227 handling touches 233-235 handling transform-origin in CSS 225 implementing pinch to zoom 227 initializing and correcting layout for pinch to zoom 230-232 next steps 236 overview of 224 preventing interference of scrolling with 232-233 reimplementing in JavaScript 163 scale and 224-225, 235-236 as type of gesture 153 wrap up 237 Pipelining, HTTP 34 Pixels, virtual on viewport 19-22 PJAX (pushState AJAX) adding HTML fragments 78 adding page handler function 85-86 adding templates to client 83-85 benefits over AJAX 68-70 for changing browser history 71-72 creating router 79-81 implementing a route 81-83 intercepting click events 81 single-page frameworks 87 support in mobile browsers 73-74 Pointer events, IE10 handling MSPointerDown events 221-222 handling MSPointerMove events 222-223 handling MSPointerUp events 224 listening to 212-214 overview of 93-94 Polyfill, overthrow as 190

popstate event, handling 72-73 Power switch, building control for 156-159 prevSlide function, bird browser example 206-207 Progressive enhancement building in layers and 155 of touch control 156-159 Properties animatable CSS 106-107 of pointer events 93 of touch objects 92 transform property 122 viewport 21 Pruning DOM nodes 86 pushState method. see also PIAX (pushState AJAX) browser history API 71 changing history with 71-72 cross-browser state change handler 76-77 mobile browsers support 73-74 px units, in CSS 20

R

replaceState method browser history API 71 for upgrading/downgrading URLs 75–77 requestAnimationFrame 117–120 Round-trip times, PageSpeed tool minimizing 38 Routers creating 79–81 implementing a route 81–83

S

Safari finding center point of layout 233 gestures 153 multi-touch support 218 optimizing caching 51 parallel download support 34 scale property 233 testing performance with console.time() 136 viewports and 18 Web Inspector for diagnosing performance issues 33 WebKit browser engine in 8 Scale determining a scale factor 233 geometry of dilation and 226-227 pinch effect and 224

transform-origin and 225. 235-236 Scientific method, applying to JavaScript performance 136 Scripts, blocking rendering while waiting for execution of 35, 41 Scrolling CSS for fixed position layout 186-189 fixed positioning approach in bird browser example 185-186 getting the right feel 183 infinite scrolling 140, 142–143 iScroll used for overflow 191-192 issues with scroll handler 145 native scrolling in iOS 5+, Android, and IE10 184-185 overflow:auto 189-190 overthrow used to add overflow 190 overview of 184 preventing interference with pinching 232-233 replacing scroll handler with timer 145-146 types of gestures 153 191–192 wrap up sessionStorage variable, web storage API 53 setTimeout animation 106, 117 Shared memory, for graphics 7 sidebars 16 Silk browser. see Amazon Silk browser Single-page frameworks 87 Slides, for bird browser goTo function for moving between 207-208 preparing for interaction 205-206 slide class for navigating between pages 202-205 Slides, for touch lightbox building 174-175 moving into position 175-176 template for 173-174 Smartphones 5 Smith, David 6 "snapping back," 162-163 Souders, Steve 34,37 SPDY, downloading multiple requests and 34 Speed of light, latency and 37 startPos variable, handling touch events 178 Subproperties, animation 113-114

Swipeable bird browser absolute positioning and 195–198 data model for 199–201 goTo function for moving to next slide 207–208

handling orientation in browsers 211-212 initializing interface and listening for touch events 208-211 listening to pointer events 212-214 overview of 195 preparing slides for interaction 205-206 slide class for navigating between pages 202-205 wrap up 214 wrapper for data calls to allow insertion of caching laver 201-202 Swiping adding gesture support to lightbox 176-177 bird browser example. see Swipeable bird browser conventions of touch interfaces 154-155 ease of 155 getting the right feel 183 light switch example 163 lightbox example. see Lightbox example slides and 173 types of gestures 153 user feedback and 11

Т

Tablets creating styles for breakpoints 27-29 dealing with load issues related to image size 45 touch device form factors 4, 7 touch interface on 5 Tap events creating synthetic tap event 97-102 gestures compared with 153 handling 94-97 what makes them different from click 89-90 events TCP connections, making requests when loading pages 32 Templates adding to client 83-85 creating slide template 173-174 using templates and CSS for interface 140-142 Testing performance with console.time() 136 with jSPerf.com website 136-137 Threads, JavaScript 139

Thumbnails creating thumbnail HTML 164 styling 164–167 "time to first byte," in measuring load time 31 Time To Live (TTL), controlling HTTP cache 50 Timer, replacing scroll handler with 145-146 TiVo 10-11 Touch control, progressive enhancement of 156-159 Touch conventions 11, 154-155 Touch devices, introduction to conventions and 11 desktops compared with 4-5 "direct manipulation." 10 form factors 7 HTML5 and 8-9 OSs (operating systems) 5-7 "the uncanny valley," 9-10 user feedback 10-11 web browsers 7-8 wrap up 11 Touch events attaching to touch lightbox 177 canceling. see touchcancel event ending. see touchend event event listener 160-161 handling 178-181 moving. see touchmove event starting. see touchstart event types of 91 Touch interface conventions 11, 154-155 "direct manipulation" and 10 feedback from 90 initializing interface and listening for touch events 208-211 mouse compared with 80 naturalness of 80 on tablets 5 "the uncanny valley" and 9-10 user feedback and 10-11 web browsers and 3 Touch objects, properties of 92 touchcancel event adding user feedback to light switch 163 listening to 160-161 WebKit and 91 touchend event adding user feedback to light switch 162 listening to 160-161

managing end of touch life cycle 223-224 WebKit and 91 touchmove event acting on touch movements 222-223 adding user feedback to light switch 161-162 handling 178-179 listening to 160-161 WebKit and 91 touchstart event attaching listener to 96-97 handling 178-179 listening to 160–161 managing start of a touch 221-222 toggling visibility of a photo 95 WebKit and 91 transform-origin in CSS 225 scaling elements and 235-236 Transforms 2D and 3D transforms 125-128 CSS transforms 122-125 matrices 130-132 scrubbing 128-129 Transitions CSS 106-107 JavaScript 108-112 translate transform function 122-123 TTL (Time To Live), controlling HTTP cache 50

U

UI (user interface). see also Touch interface conventions 11 creating custom event interface 97-99 CSS (Cascading Style Sheets) 140-142 graphic interface in original Macintosh 80 infinite scroll as UI pattern 140 initializing interface and listening for touch events 208-211 mouse compared with touch interface 89 "the uncanny valley," 9-10 Uniform resource locators. see URLs (uniform resource locators) URLs (uniform resource locators) breaking links 70 browser history API and 70-71 creating router for handling 79-81 cross-browser state change handler 76-77 handling popstate event 72-73 hash-bang urls 77

pushState method for changing browser history 71-72 upgrading incoming links 75-76 User feedback adding gesture support to lightbox 176 adding to light switch example 161-163 animation and 105 gestures and 153 prioritizing 139 touch interface and 10-11.90 UTF-8 character set specifying 18 using for arrows 186 Utility functions, for lightbox example 168-169

V

Validation, controlling HTTP cache 50 Vendor prefixes animation and 112 creating CSS animation using 114-116 transitions and 110-112 Viewports high-density displays 22 meta property 163, 185, 228 overview of 18 properties 21 virtual pixels on 19-22

W

W3C (World Wide Web Consortium) 9 WAI-ARIA (Web Accessibility Initiative-Accessible Rich Internet Applications) 44, 186 Waterfall graph, for animation frames 121 Web Accessibility Initiative-Accessible **Rich Internet Applications** (WAI-ARIA) 44, 186 Web browsers Android Browser. see Android Browser Android Chrome, see Android Chrome bird browser example. see Swipeable bird browser browser history API and 70-71,77 browser normalization 168 cross-browser state change handler 76-77 Firefox Mobile. see Firefox Mobile gesture support 155 handling orientation 211-212 how pages are loaded 32 Internet Explorer 10. see IE10 (Internet Explorer 10) iOS Safari. see Safari

for mobile devices. see Mobile browsers multi-touch support 218 native scrolling in 184–185 normal cache behavior 51 parallel download support 34 pushState for changing browser history 71-72 for touch devices 7-8 touch interface and 3 Web development, progressive enhancement approach 155 Web Hypertext Application Working Group (WHATWG) 8-9 Web pages rendering 32 slide class for navigating between pages of bird browser 202-205 Web storage. see also Cache/caching as caching laver 54-60 web storage API 53 WebKit CSS transitions and 107 developer tools 33

mobile browser 7-8 throwing exception when exceed storage space 61 touch events 91 vendor prefixes required for animation 112, 114-116 WHATWG (Web Hypertext Application Working Group) 8-9 Widgets iScroll 4 191–192 lightbox example. see Lightbox example window.pageYOffset 142 Windows 8 multi-touch support 218 operating systems for touch devices 5.7 zooming impacting fixed layouts 194-195 Windows Phone 8, landscape orientation in 194 World Wide Web Consortium (W₃C) 9 Wrappers, for data calls to allow insertion of caching layer 201-202

Υ

YSlow team, tips for downloading large files 34 YSlow tool expiration settings for HTTP cache 50 troubleshooting load time 36–37 YUI creating custom event interface 97 facades around DOM for event handling 99–102

Ζ

Zoom/zooming creating layout for pinch to zoom 227–229 double tap convention for 90 impacting fixed layouts 194–195 implementing pinch to zoom 227 initializing and correction layout for pinch to zoom 230–232