



Beginning

PhoneGap

Thomas Myer

www.it-ebooks.info

BEGINNING PHONEGAP

INTRODUCTION	xix
CHAPTER 1	Introducing PhoneGap	1
CHAPTER 2	Installing and Configuring PhoneGap	17
CHAPTER 3	Basic Walkthrough	31
CHAPTER 4	Events	45
CHAPTER 5	Working with the Device, the Network, and Notifications	59
CHAPTER 6	Accelerometer	73
CHAPTER 7	Compass	85
CHAPTER 8	Geolocation	99
CHAPTER 9	Media	113
CHAPTER 10	Camera	129
CHAPTER 11	Storage	143
CHAPTER 12	Files	157
CHAPTER 13	Contacts	179
CHAPTER 14	Capture	189
CHAPTER 15	Creating a Note-Taking Application	197
APPENDIX A	Answers to Exercises	213
APPENDIX B	Tools for PhoneGap	235
APPENDIX C	PhoneGap .js	247
APPENDIX D	PhoneGap Plug-ins	333
INDEX	349

BEGINNING **PhoneGap**

Thomas Myer



John Wiley & Sons, Inc.

Beginning PhoneGap

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2012 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-118-15665-0
ISBN: 978-1-118-22600-1 (ebk)
ISBN: 978-1-118-23932-2 (ebk)
ISBN: 978-1-118-25399-1 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Limit of Liability/Disclaimer of Warranty: The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (877) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2011939646

Trademarks: Wiley, the Wiley logo, Wrox, the Wrox logo, Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. Android is a trademark of Google. Blackberry is a registered trademark of Research In Motion. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

To my wife, Hope, for loving me anyway.

ABOUT THE AUTHOR

THOMAS MYER works and lives in Austin, Texas. He's been a PHP web developer for about ten years, and a mobile apps developer for the past three. Until recently, he was the Top Dog of Triple Dog Dare Media, but now works for Deal Management, LLC, out of Minnesota, where he builds web and mobile apps. You can follow him at @myerman on Twitter.

ABOUT THE TECHNICAL EDITORS

BRIAN LEROUX is the lead software architect at Nitobi Inc. where he focuses on delivering web and mobile apps and helping developers all over the world write their apps. He is a contributor to the popular PhoneGap Open Source framework, and is the creator of XUI and Lawnchair.

DANNY LOWE was born in Huntington Beach, and raised in Southern California, but now calls Austin, Texas, his home. He is a computer programming superhero who has been skillfully hiding his secret identity since 1996. When he is not writing code and saving the careers of project managers by leaping tall deadlines in a single bound, Lowe is playing music, writing fiction and poetry, or hanging out with his wife, Stephanie, and two sons. Follow him on Twitter at @DannyInAustin.

CREDITS

ACQUISITIONS EDITOR

Mary James

PROJECT EDITOR

Kevin Shafer

TECHNICAL EDITORS

Brian LeRoux

Danny Lowe

SENIOR PRODUCTION EDITOR

Debra Banninger

COPY EDITOR

Kim Cofer

EDITORIAL MANAGER

Mary Beth Wakefield

FREELANCER EDITORIAL MANAGER

Rosemarie Graham

ASSOCIATE DIRECTOR OF MARKETING

David Mayhew

MARKETING MANAGER

Ashley Zurcher

BUSINESS MANAGER

Amy Knies

PRODUCTION MANAGER

Tim Tate

**VICE PRESIDENT AND EXECUTIVE GROUP
PUBLISHER**

Richard Swadley

**VICE PRESIDENT AND EXECUTIVE
PUBLISHER**

Neil Edde

ASSOCIATE PUBLISHER

Jim Minatel

PROJECT COORDINATOR, COVER

Katie Crocker

PROOFREADER

Nancy Carrasco

INDEXER

Robert Swanson

COVER DESIGNER

Ryan Sneed

COVER IMAGE

© tc397 / iStockPhoto

ACKNOWLEDGMENTS

MANY THANKS TO Mary James of Wiley, who called out of the blue one day and wondered if I would like to work on a PhoneGap book. The same thanks go to Kevin Shafer and the fine editorial team at Wiley for taking my garbled gobblydegook and making it into an actual book.

This book would not be complete without my technical reviewers: Brian Leroux, who took a lot of time out of his busy schedule to give me his thoughts on the manuscript; and Danny Lowe, who proved invaluable as he tested every line of code in the examples. I know I was a pain in the rear to work with, so thanks for putting up with me.

CONTENTS

<i>INTRODUCTION</i>	<i>xix</i>
CHAPTER 1: INTRODUCING PHONEGAP	1
Using PhoneGap	1
Looking at a Quick Showcase	2
Taking a Basic Run-Through	5
History of PhoneGap	8
Getting to Know the Origins of PhoneGap	8
Assessing PhoneGap's Current Status	8
Understanding What PhoneGap Is Good/Bad At	9
Understanding the Basics of a PhoneGap Application	10
What You Can Build with PhoneGap	10
Quick Overview of the API	12
Mobile Design Issues	13
Summary	15
CHAPTER 2: INSTALLING AND CONFIGURING PHONEGAP	17
Downloading PhoneGap	17
Downloading PhoneGap for iOS	18
Downloading PhoneGap for Android	20
Downloading PhoneGap for BlackBerry	22
Downloading PhoneGap for webOS	23
Downloading PhoneGap for Symbian	24
Using PhoneGap Build	25
Choosing Your Environment	27
Summary	28
CHAPTER 3: BASIC WALKTHROUGH	31
Using PhoneGap with Xcode	31
Using PhoneGap with an Android Device	36
Using PhoneGap with a BlackBerry Device	41
Using PhoneGap with Other Devices	42
Summary	42

CHAPTER 4: EVENTS	45
Understanding Events	45
Using the Events Listener	46
Understanding Event Types	47
backbutton	47
deviceready	48
menubutton	49
pause	50
resume	50
searchbutton	51
online	52
offline	53
Summary	56
CHAPTER 5: WORKING WITH THE DEVICE, THE NETWORK, AND NOTIFICATIONS	59
Getting Information from the Device	59
Getting the Device Name	60
Getting the PhoneGap Version	60
Getting the Device Platform	61
Getting the Device UUID	61
Getting the Operating System Version	61
Checking for a Network	63
Determining the Connection Type	63
Using Notifications	66
Using Alerts	66
Using Confirmation Dialogs	67
Using Beeps	67
Using Vibrations	68
Summary	70
CHAPTER 6: ACCELEROMETER	73
Getting to Know the Accelerometer	73
What Is the Accelerometer?	73
Using the Accelerometer	74
Showcase of Accelerometer Applications	74
Using the Acceleration Object	78
Using Accelerometer Methods	78
getCurrentAcceleration	78
watchAcceleration	79

clearWatch	80
Accelerometer Option	80
Summary	82
CHAPTER 7: COMPASS	85
Getting to Know the Compass	85
What Is the Compass?	85
Differences among the Different Devices	86
Using the Compass	88
Showcase of Compass Applications	88
Using Compass Methods	90
getCurrentHeading	90
watchHeading	91
clearWatch	92
Using a Compass Option	92
Improving the Look and Feel	94
Summary	97
CHAPTER 8: GEOLOCATION	99
Getting to Know Geolocation	99
What Is Geolocation?	99
Using Geolocation	100
Showcase of Geolocation Applications	100
The Position, PositionError, and Coordinates Objects	103
Position Object	103
PositionError Object	104
Coordinates Object	104
Using Geolocation Methods	105
getCurrentPosition	105
watchPosition	105
clearWatch	106
Using Geolocation Options	107
Improving the Look and Feel	108
Summary	110
CHAPTER 9: MEDIA	113
Learning about Media Files	113
What Are Media Files?	113
Using Media Files	114
Showcase of Media Applications	114

The Media Object	117
Using Media Methods	117
getCurrentPosition	118
getDuration	118
play	119
pause	119
release	120
startRecord	120
stop	121
Handling Errors	121
Improving the Look and Feel	125
Summary	127
 CHAPTER 10: CAMERA	 129
Getting to Know the Camera	129
What Is the Camera?	129
What Is the Photo Gallery?	130
Using the Camera	131
Showcase of Camera Applications	131
Using the Camera Object	133
Using the getPicture Method	133
Using Camera Options	135
Improving the Look and Feel	138
Summary	141
 CHAPTER 11: STORAGE	 143
Learning about Storage Options	143
Using the Database Object	146
Opening a Database	146
Running a SQL Query	147
Viewing a Result Set	148
Handling Errors	149
Using the localStorage Object	152
Summary	155
 CHAPTER 12: FILES	 157
Learning about Filesystems	157
Learning about Directories and Files	158
Using the DirectoryEntry Object	158
Using the FileEntry Object	164

Using Flags	168
Using LocalFileSystem	168
Reading Files	169
abort	170
readAsDataURL	170
readAsText	170
Writing Files	171
Learning about Transferring Files	172
FileUploadOptions	173
FileUploadResults	173
Handling Errors	173
FileError	174
FileTransferError	174
Summary	176
CHAPTER 13: CONTACTS	179
<hr/>	
Learning about Creating Contacts	179
Saving a Contact	181
Cloning a Contact	182
Removing a Contact	182
Finding a Contact	182
Understanding Some Quirks	183
Contacts	183
ContactName	184
ContactOrganization	185
Handling Errors	186
Summary	187
CHAPTER 14: CAPTURE	189
<hr/>	
Learning about Capturing Video	189
Using the options Argument	190
Recognizing Quirks when Capturing Videos	190
Learning about Capturing Audio	191
Using the options Argument	191
Recognizing Quirks when Capturing Audio	192
Learning about Capturing Images	192
Using the options Argument	193
Recognizing Quirks when Capturing Images	193
Handling Errors	193
Summary	195

CHAPTER 15: CREATING A NOTE-TAKING APPLICATION	197
Designing the Application	197
Building the Application	198
Creating the Capture Options	198
Adding Metadata	202
Saving and Synching	203
Adding Geolocation	206
The Final Code	207
Cleaning Up the App	210
Summary	210
APPENDIX A: ANSWERS TO EXERCISES	213
APPENDIX B: TOOLS FOR PHONEGAP	235
APPENDIX C: PHONEGAP.JS	247
APPENDIX D: PHONEGAP PLUG-INS	333
INDEX	349

INTRODUCTION

THE WORLD OF MOBILE APP DEVELOPMENT is changing rapidly, and frameworks like PhoneGap represent an important step in broadening that world to a very large audience of experienced web developers. With PhoneGap, you can now transfer some (if not most) of your knowledge and skills from the web world to the mobile app world.

This book introduces you to PhoneGap concepts, and helps you transition to where you're building functional apps — all without having to learn Objective-C or another similar language for native app development.

WHO THIS BOOK IS FOR

If you're reading this book, you probably fit one of the following descriptions:

- You're a web developer with intermediate knowledge of cascading style sheets (CSS), HyperText Markup Language (HTML), and JavaScript.
- You've been building working web apps for a while now (several years, more likely), and are thus conversant with web databases like MySQL, or at least have been involved in projects that work with MySQL.
- You've been asked (either at your job or by a client) to build a mobile app that works on more than one platform (Android and iOS, for example), and you must get something done fast.
- You have access to all the tools of the trade — a text editor, an image editor, and so on.

If you're the impatient sort, and want to jump right into things, skip to Chapter 2 that discusses installing PhoneGap, and then pick and choose which chapters to read as needed. For example, if you need to learn about geolocation (discussed in Chapter 8) or the compass (discussed in Chapter 7), feel free to start there — the chapters are written so that they can stand alone.

On the other hand, if you need a more comprehensive look at PhoneGap, start at the beginning (Chapter 1) and progress to the end of the book. The chapters have been arranged in order of complexity. The first few chapters after the installation procedures deal with easy things to master (such as getting device information, discussed in Chapter 5) and proceed to more complex subjects (such as working with databases in Chapter 11, and filesystems, discussed in Chapter 12).

WHAT THIS BOOK COVERS

This book serves as an introduction to PhoneGap. When the writing for this book began, PhoneGap 0.9.6 was the active version, but somewhere in the middle of the project, PhoneGap 1.0 came out. The author revisited written material and amended it as needed, so the information should be current. Most of the discussions in this book focus on the “Big Two” mobile device operating

systems (for Android and iOS devices), but also include information on BlackBerry, Palm/webOS, and other platforms as needed.

HOW THIS BOOK IS STRUCTURED

This book has been written by keeping in mind the way a beginner would approach the subject matter of PhoneGap. “Beginner” in this context not only encompasses those who might be beginning PhoneGap developers, but also those who may be extremely experienced and savvy web developers.

Chapters 1 through 3 provide a high-level overview of PhoneGap. Chapters 4 through 14 break out the different pieces of the PhoneGap API, and show you how each of them works, with examples.

Following is a breakdown of the organization for this book:

- **Chapter 1 (“Introducing PhoneGap”)** — This chapter describes what PhoneGap is, why it was created, and what it is good for.
- **Chapter 2 (“Installing and Configuring PhoneGap”)** — This chapter describes where you go to download PhoneGap, as well as what is necessary for configuration.
- **Chapter 3 (“Basic Walkthrough”)** — This chapter provides the basics of using PhoneGap.
- **Chapter 4 (“Events”)** — This chapter introduces you to the different device events that you can detect. Events are the foundation skill to learn for developing with PhoneGap.
- **Chapter 5 (“Working with the Device, the Network, and Notifications”)** — This chapter shows you how to get information from the device, how to detect the network, and how to send notifications to the user. Combined with Chapter 4, you now know the basics of creating the simplest mobile app possible.
- **Chapter 6 (“Accelerometer”), Chapter 7 (“Compass”), and Chapter 8 (“Geolocation”)** — These chapters cover the basic information you need to allow you to know where the device is in space (using the accelerometer), where it is headed (using the compass), and what its position is relative to the user (geolocation). By knowing these three things, you can understand how many social location services are added to various apps.
- **Chapter 9 (“Media”) and Chapter 10 (“Camera”)** — These chapters teach you how to play back audio and capture photos.
- **Chapter 11 (“Storage”) and Chapter 12 (“Files”)** — These chapters teach you how to store persistent data (either in a database or as a file).
- **Chapter 13 (“Contacts”)** — This chapter shows you how to interact with the device’s Contacts database.
- **Chapter 14 (“Capture”)** — This chapter shows you how to capture photographs, as well as audio and video recordings.
- **Chapter 15 (“Creating a Note-Taking Application”)** — This chapter uses what you’ve learned in the previous chapters and walks you through the building of a note-taking app that works with video, audio, photos, and text.

The appendixes provide further information on PhoneGap exercises presented in the chapters, tools you can use with PhoneGap development, the `PhoneGap.js` file, and PhoneGap plug-ins.

WHAT YOU NEED TO USE THIS BOOK

You will need a text editor of some kind to work with the code examples presented in this book. You will also need to occasionally edit images for your projects, although that's not required for any projects in this book. If you're developing an iOS application for iPhone, iPad, or iPod Touch, you will need Xcode and an Apple Developer License. Other platforms (such as BlackBerry or Android) require their own tools and procedures, and these are noted in the installation instructions provided in Chapter 2.

CONVENTIONS

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.

TRY IT OUT

The *Try It Out* is an exercise you should work through, following the text in the book.

1. It usually consists of a set of steps.
2. Each step has a number.
3. Follow the steps through.

How It Works

After some *Try It Out* exercises, the code you've typed will be explained in detail.



WARNING Boxes with a warning icon like this one hold important, not-to-be-forgotten information that is directly relevant to the surrounding text.



NOTE The pencil icon indicates notes, tips, hints, tricks, or asides to the current discussion.

As for styles in the text:

- We *highlight* new terms and important words when we introduce them.
- We show keyboard strokes like this: `Ctrl+A`.

- We show filenames, URLs, and code within the text like so: `persistence.properties`.
- We present code in two different ways:

We use a monofont type for most code examples.

We use bold to emphasize code that is particularly important in the present context or to show changes from a previous code snippet.

SOURCE CODE

As you work through the examples in this book, you may choose either to type in all the code manually, or to use the source code files that accompany the book. Some of the source code used in this book is available for download at www.wrox.com. When at the site, simply locate the book's title (use the Search box or one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book. Code that is included on the website is highlighted by the following icon:



If it is just a code snippet, you'll find the filename in a code note such as this:

Code snippet filename



NOTE Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-1-118-15665-0.

Once you download the code, just decompress it with your favorite compression tool. Alternatively, you can go to the main Wrox code download page at www.wrox.com/dynamic/books/download.asp to see the code available for this book and all other Wrox books.

ERRATA

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration, and at the same time, you will be helping us provide even higher quality information.

To find the errata page for this book, go to www.wrox.com and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page, you can view all errata that has been submitted for this book and posted by Wrox editors.



NOTE A complete book list, including links to each book's errata, is also available at www.wrox.com/misc-pages/booklist.shtml.

If you don't spot "your" error on the Book Errata page, go to www.wrox.com/contact/techsupport.shtml and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at p2p.wrox.com. The forums are a web-based system for you to post messages relating to Wrox books and related technologies, and to interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At <http://p2p.wrox.com>, you will find a number of different forums that will help you, not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to p2p.wrox.com and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join, as well as any optional information you wish to provide, and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.



NOTE You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works, as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

1

Introducing PhoneGap

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Using PhoneGap to build an application
- Exploring the history of PhoneGap
- Exploring what you can build with PhoneGap
- Understanding the basics of a PhoneGap application

Welcome to PhoneGap! You're reading this book because you've probably heard about PhoneGap, and want to learn more about how you can use it to build cross-platform mobile applications. This book delves into different parts of the PhoneGap API, and teaches you how to use the PhoneGap tools to build different applications.

Before getting into the nitty-gritty details, though, it's time to get a 10,000-foot overview of PhoneGap and what it can do. In this chapter, you learn about the basics of using PhoneGap, as well as a bit of the history of PhoneGap. By the end of this chapter, you will be familiar with the basics of building a PhoneGap application.

USING PHONEGAP

As you can see in Figure 1-1, according to PhoneGap's website (www.phonegap.com), PhoneGap is an "HTML5 app platform that allows you to author native applications with web technologies."



FIGURE 1-1: The homepage of the PhoneGap website

As a developer, you'll be using PhoneGap to develop working code for iPhones, Androids, BlackBerries, and webOS devices. In each case, you'll be using a PhoneGap wrapper that contains your web-based HTML, cascading style sheet (CSS), and JavaScript code. What this means for you is that you will be using web technologies you already understand, but still get access to many of the device's native features (such as the compass, the camera, the contacts list, and so on). It also means that you should be able to port your web code to multiple different devices with little or no changes.

Looking at a Quick Showcase

You might be thinking to yourself that PhoneGap sounds promising, but you're not quite sure what it can actually do. Without getting into a lot of technical detail quite yet, the best way to illustrate its capabilities is to offer a brief showcase of successful projects.

"Diary Mobile"

As shown in Figure 1-2, "Diary Mobile" is a mobile app that helps you to better organize your life. It features a Journal, a Task Manager, and a Planner. You can keep notes on your Android or iPhone, and then synchronize your information back to a centralized web repository. The app works offline, and automatically saves items online once you're back online.

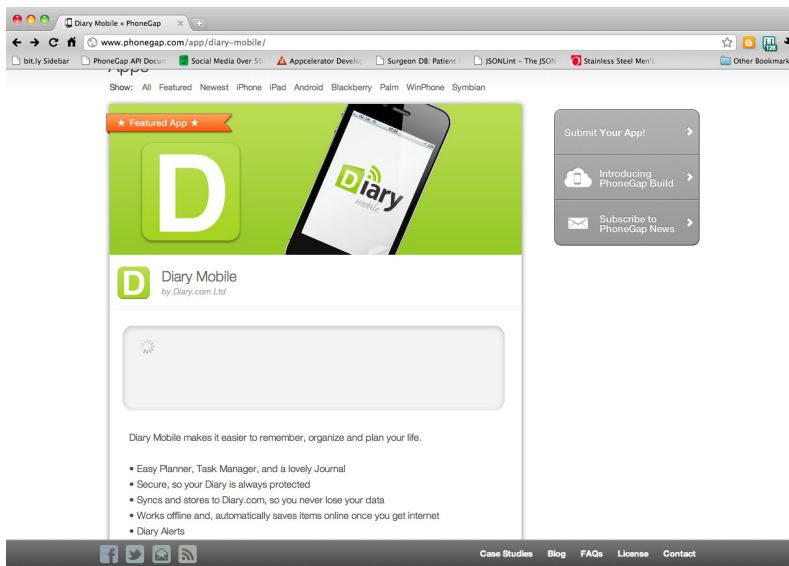


FIGURE 1-2: “Diary Mobile” application

“NFB Films”

As shown in Figure 1-3, “NFB Films” is an Android-only PhoneGap app that lets you watch more than 2,000 films for free. Users can search for content, browse films by channel or subject, and share their favorite documentaries, animated shorts, and trailers with other users.

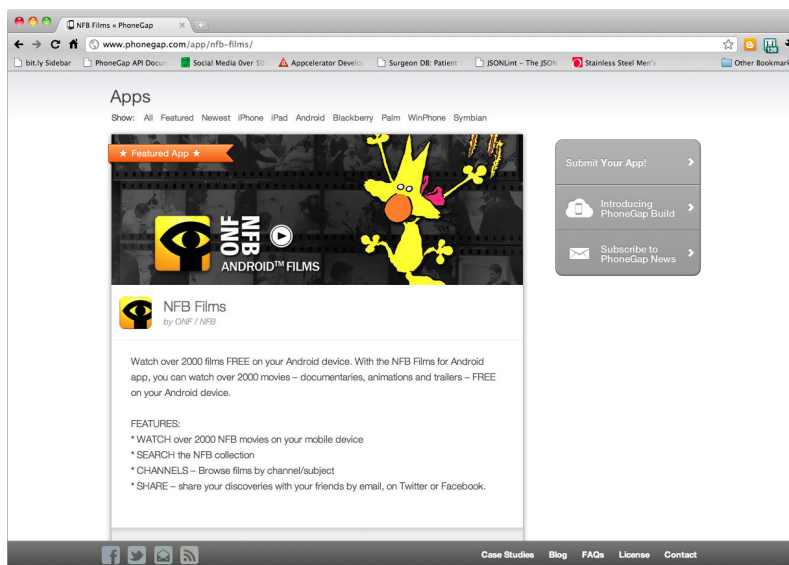


FIGURE 1-3: “NFB Films” application

“Just One More”

As shown in Figure 1-4, “Just One More” is an iPad/iPhone PhoneGap app that helps you find inspiring video content. It offers a simple, addictive interface for browsing and searching the Vimeo website (www.vimeo.com) — it’s a rich, immersive app, totally built with HTML, CSS, and JavaScript.

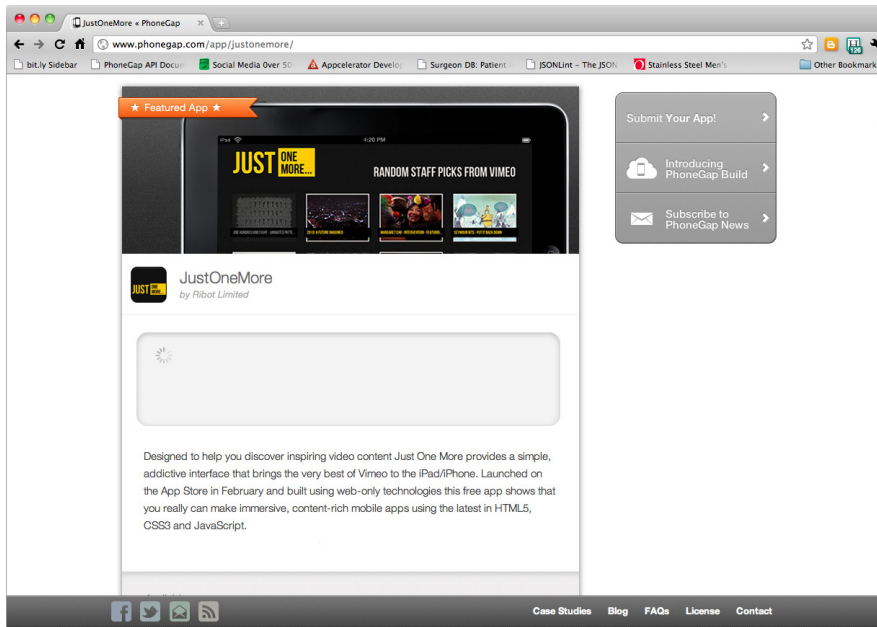


FIGURE 1-4: “Just One More” application

“Orbium”

As shown in Figure 1-5, “Orbium” is a fast-paced puzzle game available on iPhone, Android, and webOS. It features high-quality graphics and an intuitive touch-screen interface.

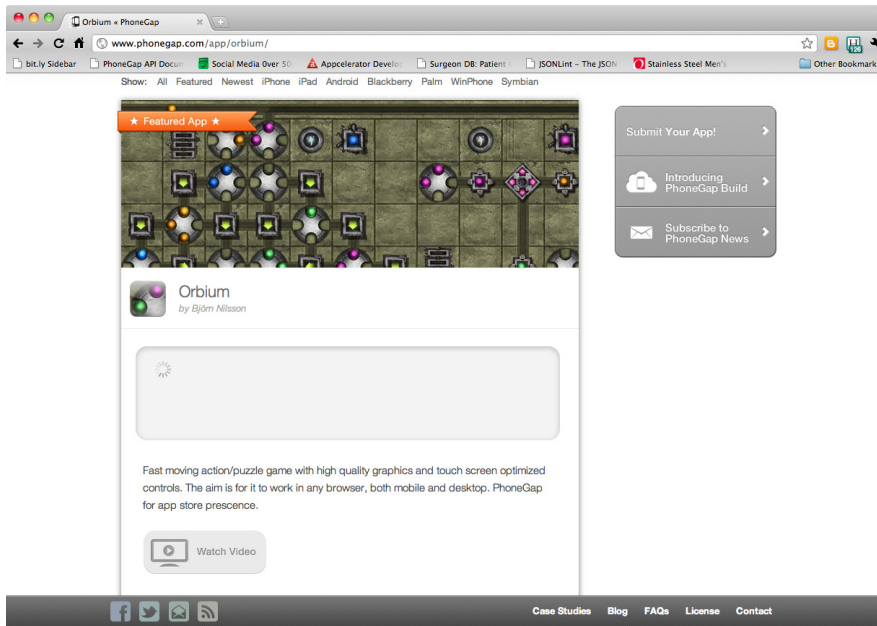


FIGURE 1-5: “Orbium” application

Taking a Basic Run-Through

PhoneGap doesn’t limit you to just creating HTML-based utilities. Just about anything you can dream up with modern HTML5, CSS3, and JavaScript, you can create with PhoneGap.

So, what’s involved, generally speaking, when you create a PhoneGap application? This entire book is devoted to drilling down into this topic with great detail, but first, it might be helpful to understand the overall process involved in working with PhoneGap.

Here is a typical workflow for building your first PhoneGap application:

1. Download the PhoneGap tools from www.PhoneGap.com. You’ll need the specific set of tools for the platform you’re working with (for example, Xcode for iPhone applications, and so on).
2. Follow the installation instructions to add the PhoneGap tools to your existing environment.
3. Once you have the tools installed, you typically need to start a new project using the extensions you just installed. For example, as shown in Figure 1-6, if you are using Xcode, you’ll have the option to start a PhoneGap project.

4. Once you've started a new project, you'll be working within the `www` folder, which, as shown in Figure 1-7, contains all your HTML, CSS, and JavaScript files.
5. Build your code and use the Emulator or an actual device to test what you're doing. Some of the supported functionality (such as vibrate notifications) probably won't work with the on-screen Emulator (shown in Figure 1-8), but your mileage will vary.
6. When you're ready to release your application to a marketplace, follow the specific instructions for the app store you're working with (Android market, Apple App Store, and so on) to release your software.
7. If you must support a different device, you can install the tools for that secondary device, port your web code over, and follow a testing a release path, mostly with minimal changes to your web code.

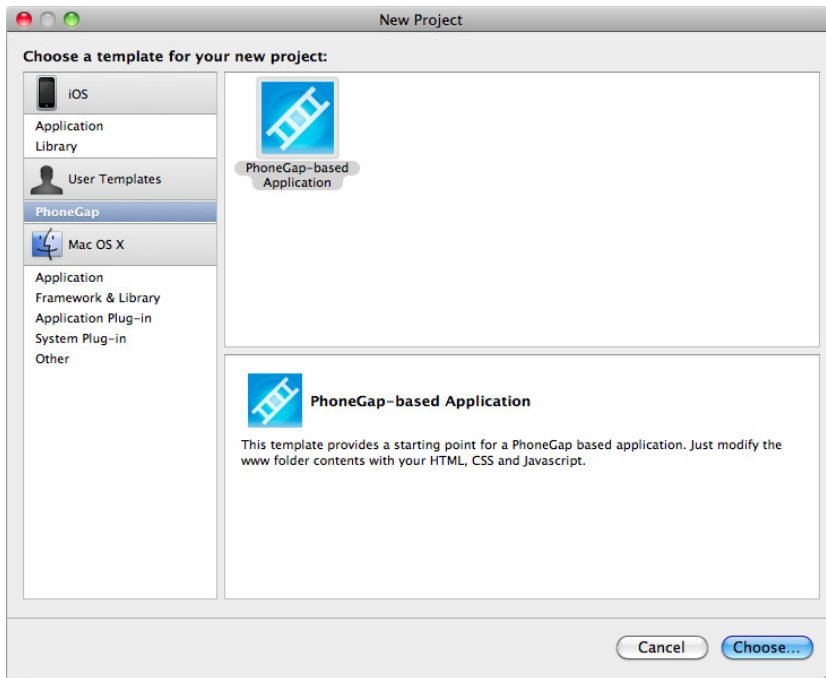


FIGURE 1-6: Starting a new project using Xcode

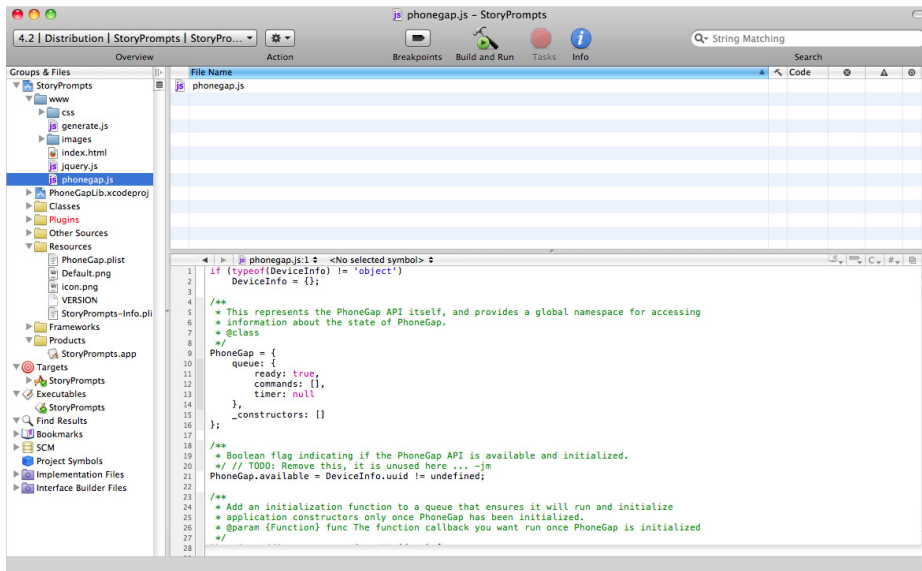


FIGURE 1-7: Working within the www folder

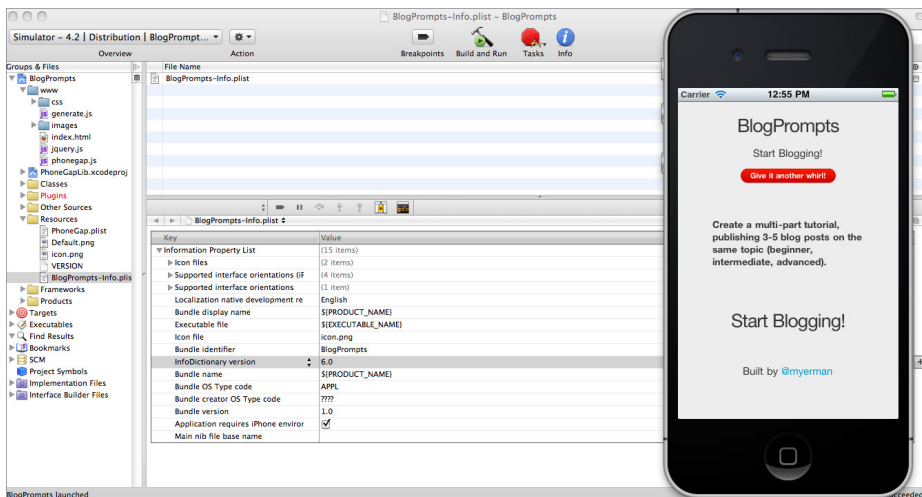


FIGURE 1-8: The on-screen Emulator

Obviously, there's a lot more to it than that, especially in what you can and can't do on specific devices. Before getting into all of that, however, let's take a quick look at the history of PhoneGap.

HISTORY OF PHONEGAP

PhoneGap solves two problems for developers:

- Can the developer (you!) use more familiar web languages like HTML and JavaScript to create a working mobile application?
- Can that code then be ported to another platform quickly and easily, thereby reducing the complexity of supporting multiple platforms?

Getting to Know the Origins of PhoneGap

The first PhoneGap code was authored at the San Francisco iPhoneDevCamp in August 2008. A contributing driver for its creation was a simple fact that almost every single newbie iPhone developer runs into: Objective-C can be a very unfamiliar environment for web developers, and there are lots more web developers out there than there are Objective-C ones.

The question was, could someone develop a framework that allowed web developers to leverage all their knowledge of HTML, CSS, and JavaScript that could also interact with the important native parts of an iPhone, such as the camera and compass?

Within a year, PhoneGap was winning awards and was starting to support the Android platform, making it even more useful to the growing family of mobile developers who need to support code on more than one platform.

Assessing PhoneGap's Current Status

Currently, PhoneGap is at Version 1.x (1.0.0.rc2 was released July, 2011). PhoneGap offers support for the following features across many of the major smartphone devices:

- Accelerometer
- Camera
- Compass
- Contacts
- Files
- Geolocation
- Media
- Network
- Notifications (alerts, sounds, and vibrations)
- Storage

If you're developing for iPhone or Android devices, chances are that all of that functionality is supported. If you're developing for Blackberry, webOS, Windows Phone 7, Symbian, or Bada devices, there may be a few things missing. (For example, there is no support for the camera, compass, or storage features on Windows Phone 7. The older Blackberry models support geolocation, notifications, and network reachability, but that's about it.)

A road map of future releases includes updates to the `Contact` API, bringing it up-to-date with the latest World Wide Web Consortium (W3C) specifications. Furthermore, plans are in the works for the following features (with this just being a taste, not the complete road map):

- Crypto
- Websockets
- Web notifications
- HTML media capture
- Calendar API
- Internationalization support
- Command-line builds
- Plug-in architecture (will help third-party developers extend PhoneGap)
- Network loss/regain events

Understanding What PhoneGap Is Good/Bad At

What is PhoneGap good at? Primarily, it helps you and your development team leverage years of experience at building interactive projects with web standards. If you're good with HTML, CSS, and JavaScript, you'll experience few problems while working with PhoneGap — all you need is a good introduction to the specifics of the API components, which this book will give you.

Another thing PhoneGap is good at is bridging the gap (hence the name) between your standard web technologies, and the unique capabilities inherent in your smartphone. By using the PhoneGap API components, you can quickly and easily access the onboard camera, pull up the contacts, or work with the compass.

If you need to connect your application with a remote web service (typically, a web service or a RESTful API), you can easily bring in tools like jQuery to create powerful Ajax handlers. Of course, you're also free to hand-roll your own `XmlHttpRequests`.

Given all this, a word of caution is in order. Just because you code an application using PhoneGap and it works on one device (iPhone, for example) doesn't automatically mean that it will work on others. You may have to test and tweak for Android, Blackberry, or Windows Phone 7 devices. So, don't imagine for a second that this is a “code once, deploy multiple times” scenario.

If you are working with multiple devices, you will need separate environments for each wrapper. For example, you won't be able to maintain your Android PhoneGap wrapper with Xcode. Another way of framing this is that you are potentially simplifying how much your web code must change to run on different devices, but you still must maintain separate wrappers.

If you create a PhoneGap application that relies heavily on animation and graphics, you might tax the device you are on — this is true whether you are working with PhoneGap or native code. Also, the same is true if you are creating huge dependencies with a remote API — your application should work well in offline mode, because you will never know if the person using the app has lost connectivity.

UNDERSTANDING THE BASICS OF A PHONEGAP APPLICATION

Now it's time to dig a little deeper into PhoneGap. Before jumping into an overview of the API specifics, it might be useful to go over some examples of what you can do with the major components — just to get your creative juices flowing for possible application ideas.

What You Can Build with PhoneGap

Earlier, you saw a quick showcase of different PhoneGap applications. Let's take a brief look at the possibilities with some of the major components of the API.

Working with Contacts

The contacts list is a fairly ubiquitous feature, available on most smartphones. With PhoneGap, you can easily do the following with the contacts feature:

- Create a contact using the `create()` method.
- Save a contact using the `save()` method.
- Find a contact using the `find()` method.
- Clone a contact using the `clone()` method.
- Remove a contact using the `remove()` method.

To create a contact, you pass a JavaScript Object Notation (JSON) object to `contacts.create()`, which creates a contact only within the application memory. To save the new contact to the Contacts database, you would use `save()`.

The PhoneGap API supports various property fields for a contact (such as a display name, a nickname, phone numbers, e-mails, addresses, birthday, gender, photos, time zone, and so on), most of which you can run through `contacts.find()` to get a list of matching users.

The `clone()` method allows you to quickly copy a contact that's in memory, and then change just the properties that are different. For example, you might have a group of people who all share the same physical address. Cloning the original contact would allow you to work through the list of property changes very quickly.

The `remove()` method works as expected — use it to remove a contact from the device's contacts database.

From this basic list of methods and functions, you could easily incorporate various views that allow users to search through their contacts, create new contacts, make changes to existing contacts, or even delete contacts. For example, your application might help users make batch operations on all of their contacts based on search criteria (such as a ZIP code or existence of a phone number).

Working with the Camera

Most smartphones have built-in cameras. The PhoneGap API provides two ways to capture images, and one is giving access to the camera via the `camera` object. The second is by using the `Media Capture` API (which you will learn a little about later in this chapter) Specifically, the

`camera.getPicture()` method takes a photo using the camera, or retrieves a photo from the device's photo album, depending on what source type you have passed in (either `CAMERA` or `PHOTOLIBRARY`).

You can also choose to have the camera provide you with a Base64-encoded photo image (this is the default setting), or the image file location. Once you have this information, you can use it to render an image, post the data to a remote server, or save the data locally.



WARNING *Be aware, however, that because the latest and greatest smartphones use some very high-quality cameras with high-resolution graphics, you could run into memory-management issues if you choose to work with Base64-encoded data.*

An interesting option is to capture an editable photo. This introduces all kinds of possibilities with respect to being able to crop an image (for example) after the photo has been captured by the device.



NOTE *Android phones, however, seem to ignore the `allowEdit` parameter.*

What are some of the applications you can build using the `Camera` API? A good one might be a photo-sharing application that allows you to take a picture, perform some basic edits, and then publish that photo to a remote web server.



NOTE *Those of you who are familiar with social photography apps will note that this, in fact, describes “Instagram.”*

Other application possibilities include a note-taking tool that lets a user take a photo of something, and then add notes to that image. Or, you could even have an application that lets shoppers take a picture of items they find in stores, and add them to a wish list they can then share with friends.

Working with Geolocation

Another hallmark of modern smartphones is their geolocation capabilities. Most smartphones will be able to use GPS or some other technology to tell you what latitude and longitude you're currently at with some degree of precision. Of course, if the device is using cell phone towers to triangulate your position, your location won't be accurate at all. And, if you have no network connectivity, you're completely out of luck.

The PhoneGap Geolocation API lets you get a device's current position (in longitude and latitude, but other details like altitude might also be included) and to watch a position in case it changes. This would be very useful if you were trying to track the device's movement.

You have many different ways to approach geolocation in your applications. You can use geolocation data to enrich other data — for example, you could add a latitude and longitude to any pictures taken

by your application. Or, you could add geolocation data to any notes created by the user. Or, you could send geolocation data to a remote web server so that dispatchers can keep tabs on field workers.

Working with Media Files

In PhoneGap, the `Media Capture` API isn't simply a good means for capturing photos. You can also use it to capture audio and video data as well. As expected, the `Media Capture` API allows you to start and stop recording, play, pause, and stop media files, and even display an audio file's duration.

From an application point of view, you can use the `Media Capture` API to create different voice- and video-recording apps — think of how useful it would be to have a tool that lets you record audio and video, and add written notes or photos in one package (for those of who use Evernote, you'll recognize that this describes it perfectly). You could also build an app that lets you sample different audio files. This could be fairly useful if you're building a mobile application that showcases a catalog of music or spoken-word recordings — PhoneGap will play locally saved files, as well as those on a remote web server.

Working with Storage Options

Those who have been using HTML5's storage options will be glad to know that PhoneGap also supports Web SQL databases. As with HTML5, you'll be working with SQLite locally, which is normally more than sufficient to create all kinds of rich data back-ends.

For example, you could build an application that allows users to synchronize data between the device and a remote web database. Your application could use a simple Ajax call to request specific data from a remote database, use JSON to transport that data to the phone, and then transform and store that JSON object into SQLite. Then, any changes made by the user locally could be retransmitted to the remote database.

Application possibilities here are immense — such as a company intranet-based knowledge base with a mobile app that lets users pull down information, make necessary changes, and publish it back to the intranet. Or, you could just enhance device-only applications with richer data — for example, for the previously mentioned note-taking application that lets users take pictures and record audio, you could use SQLite to store any notes and metadata for each of the pictures taken.

Quick Overview of the API

Because this entire book is devoted to the particulars of working with the PhoneGap API, there's no need to delve into a huge amount of detail as to what's in the API. However, it would be good for you to know the rough outlines of what's currently available in the PhoneGap API. So, here are the API components in alphabetical order:

- `Accelerometer` — Tap into the device's motion sensor.
- `Camera` — Capture a photo using the device's camera.
- `Capture` — Capture media files using the device's media-capture applications.
- `Compass` — Obtain the direction the device is pointing to.
- `Connection` — Quickly check the network state (either WiFi or cellular network).
- `Contacts` — Work with the device's contact database.

- **Device** — Gather device-specific information.
- **Events** — Hook into native events through JavaScript.
- **File** — Hook into the native filesystem through JavaScript.
- **Geolocation** — Make your application location-aware.
- **Media** — Record and play back audio files.
- **Network** — Quickly check the network state.
- **Notification** — Visual, audible, and tactile device notifications.
- **Storage** — Hook into the device's native storage options.

In the next few chapters, you learn how to install PhoneGap and then you'll get to know the `Events` API first, which will give you the basics of how to start any application. (In other words, if you can't detect `deviceready`, which tells you that the device is ready, you can't really do much else.) You'll then proceed with checking the network state, getting device information, and firing off custom notifications. Once you've mastered those basics, you can start really digging into the API.

Mobile Design Issues

So far, you've learned a lot about the PhoneGap API, at least in general terms, and are about to learn a great deal more as you progress through the rest of the chapters in this book.

However, one thing that can't be overemphasized is this: A PhoneGap application isn't just about the PhoneGap API. You will be building an HTML5 mobile application, and you must understand not only the basics of HTML in order to succeed, but also have a grasp of CSS and JavaScript. Furthermore, if you will be using other frameworks (such as jQuery, which is very popular), you must have a solid understanding of how those work as well.

Specifically, you will be building a dynamic web application on a device, and not a static HTML page or series of pages — it's not a good idea to simply port an entire website onto PhoneGap and present that to a user.

In other words, you have to think about the scale of the device's viewscreen. It does you no good to just dump a bunch of scrolling text and gigantic design elements (such as background images) into a 320-pixel by 480-pixel screen on the iPhone, for example.

A better approach is to re-chunk any content, or provide a more cut-down version of user interface (UI) elements (particularly navigation). This is not a book about the mobile device user experience, nor will this book provide any advice on layout or usability on a small screen. However, be aware that users will notice and not appreciate having to scroll or pinch/zoom in order to get the information they need.

A good approach is to use HTML elements that bring semantic structure to your content — specifically, headers, lists, and buttons, for starters — and that you can easily style using CSS.

Another thing to consider is that mobile devices have different latency and initialization characteristics than your average web browser running on a desktop or laptop machine. In other words, when a user loads your PhoneGap application, that app must go through a series of initialization procedures to be ready for use. Having large, crufty, or extremely convoluted layouts will just make for slower going and slower parsing.

In this regard, various frameworks have appeared to make your life easier, such as xuijs (shown in Figure 1-9), jQTouch (shown in Figure 1-10) and jQuery Mobile (shown in Figure 1-11).

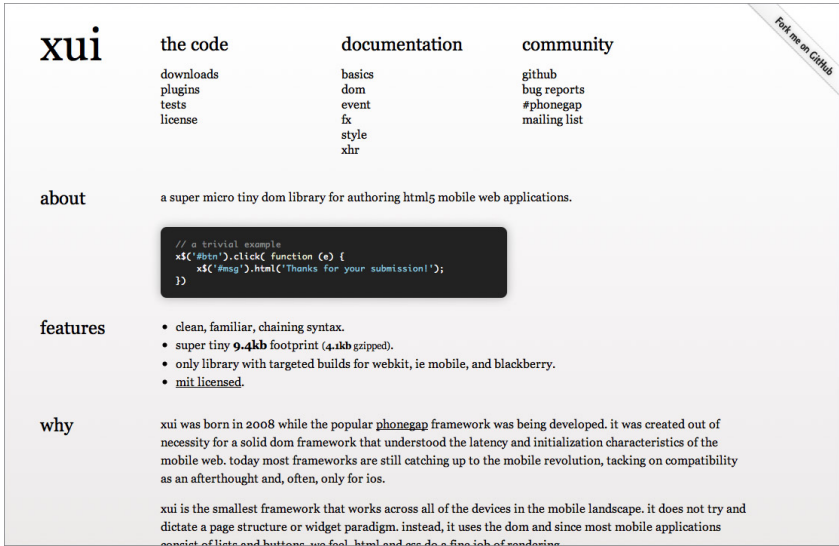


FIGURE 1-9: The xuijs framework

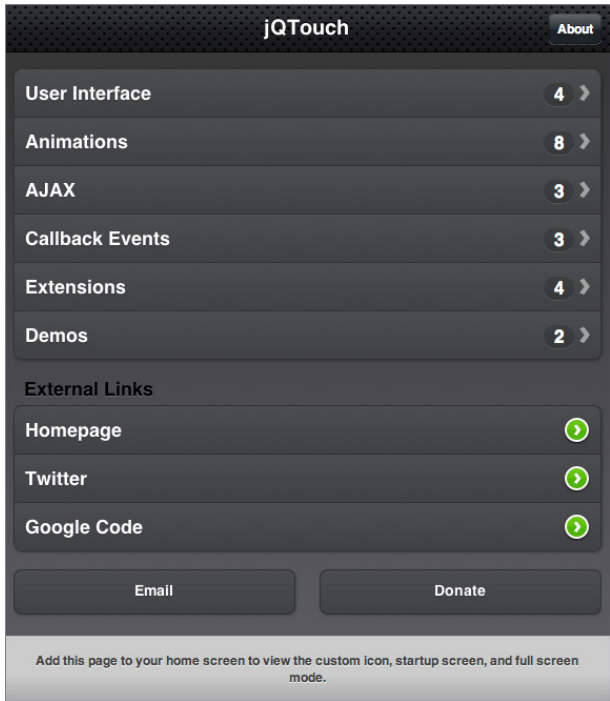


FIGURE 1-10: The jQTouch framework

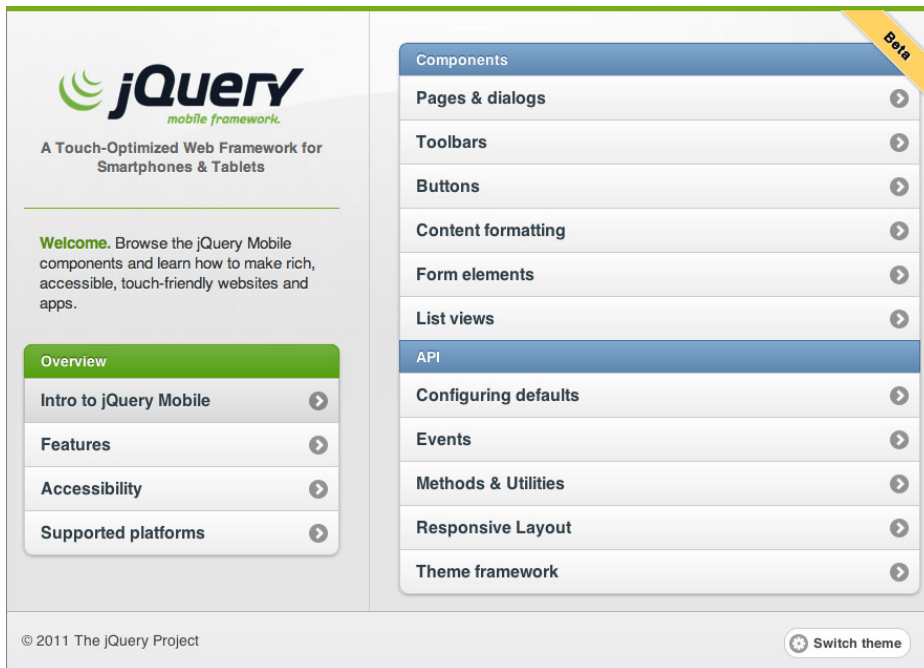


FIGURE 1-11: The jQuery Mobile framework

Different devices respond differently to the different frameworks (for example, the animations on jQTouch seem to make certain Android devices extremely unhappy, and xuijs seems to be the only framework that works on the older Blackberry devices). But don't let this kind of thing scare you off. Using a framework to create a usable UI will always be faster than putting something together yourself, and then testing on the different devices.

Last, but not least, on some phones, linking from one page to another might cause memory problems or slower interaction. One of the best practices is to try to keep as much functionality as you can on one page, and then use JavaScript functions to load information dynamically onto the page. You can do this with `document.getElementById()` or, if you prefer, by using jQuery's `$()` notation. If you're using a framework, a lot of this kind of behavior is handled for you — for example, on jQTouch, clicking a link is really a request to load a specific document object model (DOM) element, and not a separate page.

SUMMARY

In this chapter, you received a very quick introduction to PhoneGap, including some key API components (such as `Media Capture`, `Geolocation`, and `Storage`). You also learned about various frameworks and approaches that might help you in your future development of PhoneGap applications.

In Chapter 2, you'll learn how to install and configure PhoneGap.

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Typical development lifecycle	This entails installing PhoneGap tools; creating a new project; working with HTML technologies; and testing and deploying on different platforms.
Components of the PhoneGap API	This includes the Accelerometer, Camera, Capture, Compass, Connection, Contacts, Device, Events, File, Geolocation, Media, Network, Notification, and Storage APIs.

2

Installing and Configuring PhoneGap

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Downloading PhoneGap
- Installing PhoneGap
- Using PhoneGap Build

In this chapter, you learn how to find and install PhoneGap on your particular platform (OS X, Windows, or Linux). There's no need to read this entire chapter front to back — just find the bits that are relevant to your environment.

Note that this chapter also covers the brand-new PhoneGap Build feature, which enables you to submit code to a service and get app-store-ready apps in return. This service could potentially save you a lot of time and heartache, especially if you're trying to support numerous device options.

DOWNLOADING PHONEGAP

The following sections contain information about downloading PhoneGap for specific platforms. As of this writing, the current PhoneGap Software Development Kit (SDK) is Version 1.0.0, as shown in the upper-right corner of Figure 2-1.

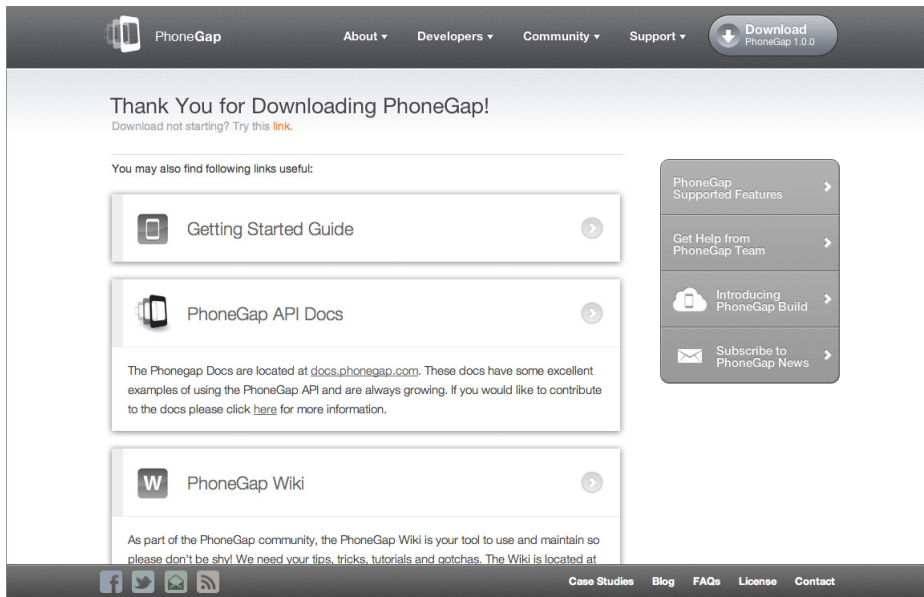


FIGURE 2-1: Current version of PhoneGap shown in upper-right corner

Downloading PhoneGap for iOS

Before you can get started, ensure that you meet the following requirements:

- You have an Intel-based computer running Mac OS X 10.6 (Snow Leopard).
- You have an Apple iOS device (iPhone, iPad, or iPod Touch).
- You have an Apple Developer Certification.
- You have Xcode already installed and configured. (You'll need a membership to the Apple Developer Portal for this.)

To install PhoneGap, follow these steps:

1. Open your web browser and point it at www.phonegap.com.
2. Click the Download button on PhoneGap's navigation bar. (It should be on the far-right margin.)
3. A ZIP file should download to your Downloads folder.
4. Double-click the ZIP file to uncompress it. You should see a folder that contains various files and directories, as shown in Figure 2-2.
5. Open the iOS folder and double-click the PhoneGapInstaller.pkg file.
6. When the installer screen opens, as shown in Figure 2-3, follow the prompts until the PhoneGap SDK is installed.

7. You can confirm that everything is installed correctly by opening Xcode and starting a new project. As shown in Figure 2-4, you should see a PhoneGap-based project as one of your options.

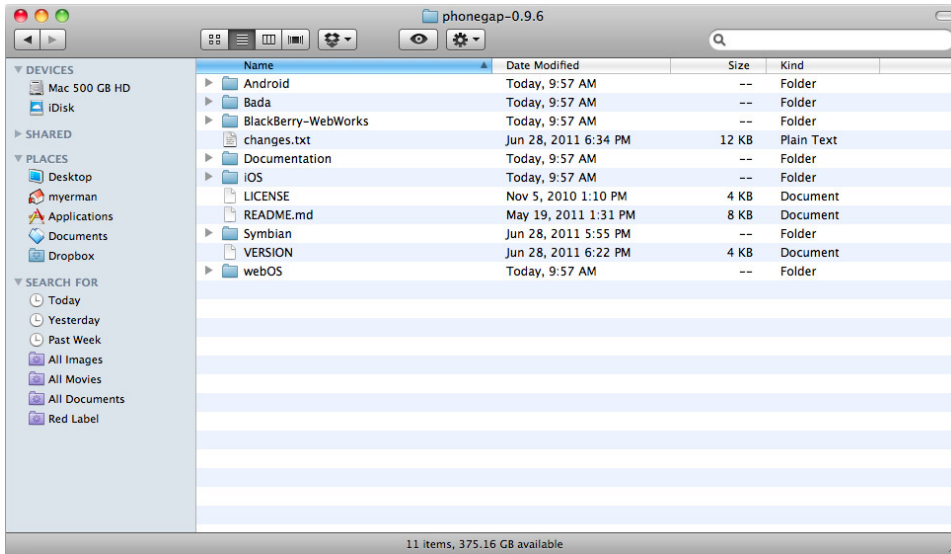


FIGURE 2-2: Folder resulting from uncompressing the ZIP file

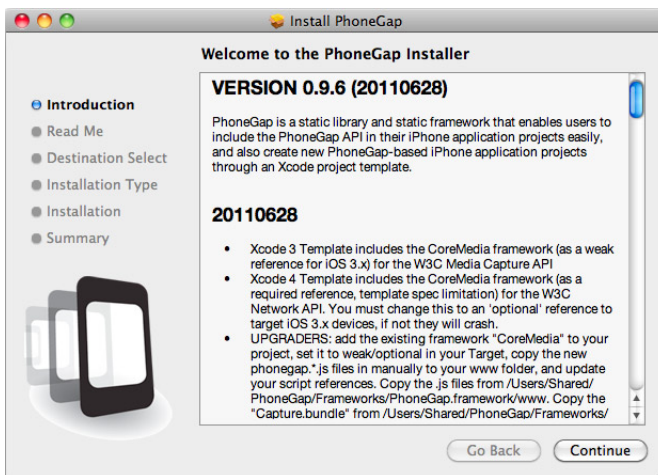


FIGURE 2-3: PhoneGap Installer screen

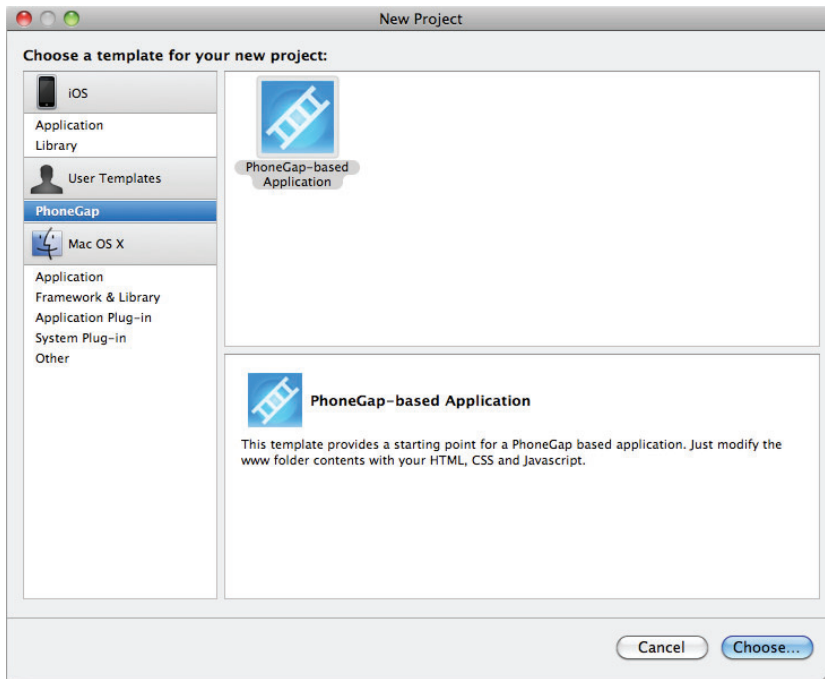


FIGURE 2-4: Starting a new PhoneGap project in Xcode

Downloading PhoneGap for Android

Before installing PhoneGap for Android, ensure that you meet the following requirements:

- You have an installed copy of Eclipse 3.4 or later.
- You have an installed copy of the Android SDK from <http://developer.android.com/sdk/index.html>.
- You have an installed copy of the ADT plug-in from <http://developer.android.com/sdk/eclipse-adt.html#installing>.



NOTE Note that if you don't have Eclipse, you learn about an alternative way to work with PhoneGap (via the command line) in Chapter 3.

To install PhoneGap, follow these steps:

1. Open your web browser and point it at www.phonegap.com.
2. Click the Download button on PhoneGap's navigation bar. (It should be on the far-right margin.)
3. A ZIP file should download to your Downloads folder.

4. Double-click the ZIP file to uncompress it. You should see a folder that contains various files and directories.
5. Open the `Android` folder. You should see a `Sample` folder, a pair of JavaScript files, and a `.jar` file, as shown in Figure 2-5.
6. Launch Eclipse, and then, from the File menu, select `New ⇨ Android Project`.
7. In the root directory of the project, create two new directories:
 - `/libs`
 - `/assets/www`
8. Copy `phonegap.js` from your PhoneGap download earlier to `/assets/www`.
9. Copy `phonegap.jar` from your PhoneGap download earlier to `/libs`.
10. Now, you must make the following adjustments to the project's main Java file found in the `src` folder in Eclipse, as shown in Figure 2-6:
 - Change the class's extend from `Activity` to `DroidGap`.
 - Replace the `setContentView()` line with `super.loadUrl("file:///android_asset/www/index.html");`.
 - Add `import com.phonegap.*;`
11. Right-click the `AndroidManifest.xml` file and select `Open With ⇨ Text Editor`.
12. Paste the following permissions under `versionName`:

```
<supports-screens
android:largeScreens="true"
android:normalScreens="true"
android:smallScreens="true"
android:resizeable="true"
android:anyDensity="true"
/>
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name=
    "android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

13. You are now also required to have a `plugins.xml` file under `./res/xml/plugins.xml`. This file is included in the latest 1.0.0 release.

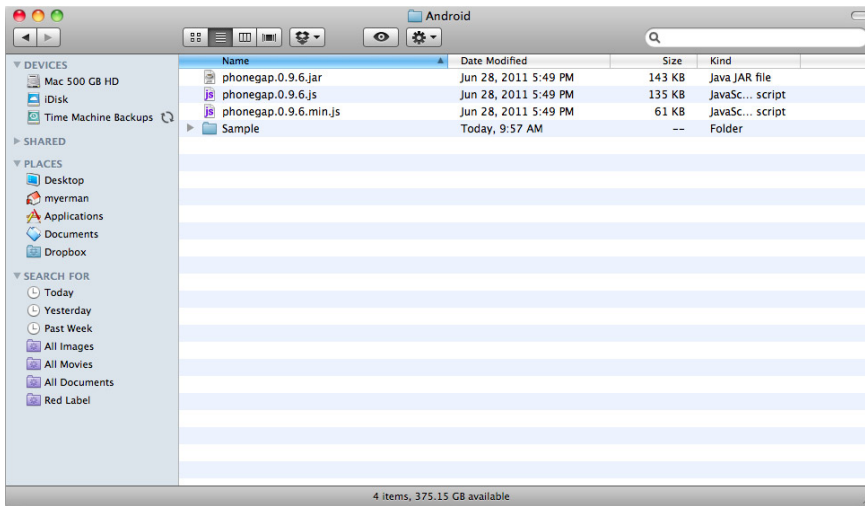


FIGURE 2-5: Contents of the Android folder

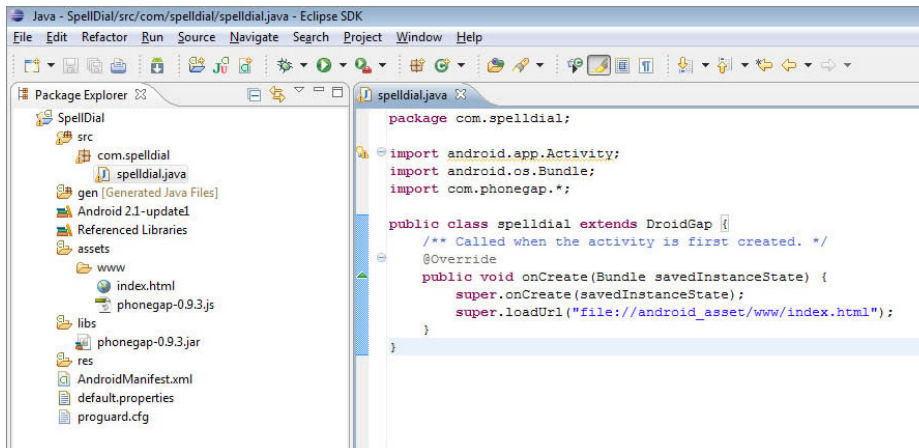


FIGURE 2-6: Main Java file in the src folder

Downloading PhoneGap for BlackBerry

Before downloading and installing the PhoneGap SDK, ensure you meet these requirements:

- You are running under the Windows XP (32-bit) or Windows 7 (32-bit or 64-bit) operating system.
- You have downloaded and installed the Sun Java Development Kit (JDK). Ensure that you add it to your PATH variable.



NOTE The Sun JDK is available at www.oracle.com/technetwork/java/javase/downloads/index.html#jdk.

- You have downloaded and extracted Apache Ant. Ensure that you add it to your PATH variable.



NOTE Apache Ant is available at <http://ant.apache.org/bindownload.cgi>.

- You have downloaded the BlackBerry WebWorks SDK and installed it to C:\BBWP.



NOTE The WebWorks SDK is available at <http://na.blackberry.com/eng/developers/browserdev/widget/sdk.jsp>.

To install PhoneGap, follow these steps:

1. Open your web browser and point it at www.phonegap.com.
2. Click the Download button on PhoneGap's navigation bar. (It should be on the far-right margin.)
3. A ZIP file should download to your Downloads folder.
4. Double-click the ZIP file to uncompress it. You should see a folder that contains various files and directories.
5. Copy the contents of the ZIP file to C:\Dev\phonegap.

Downloading PhoneGap for webOS

Before downloading and installing PhoneGap, ensure that you meet these requirements:

- You are running on a Windows, OS X, or Linux machine.
- You have downloaded and installed Virtual Box from www.virtualbox.org/.
- You have downloaded and installed the webOS SDK.



NOTE The webOS SDK is available at http://developer.palm.com/index.php?option=com_content&view=article&layout=page&id=1788&Itemid=321.

- If you're using Windows, ensure that you have downloaded and installed cygwin.



NOTE You can download cygwin at www.cygwin.com/setup.exe.

To install PhoneGap, follow these steps:

1. Open your web browser and point it at www.phonegap.com.
2. Click the Download button on PhoneGap's navigation bar. (It should be on the far-right margin.)
3. A ZIP file should download to your Downloads folder.
4. Double-click the ZIP file to uncompress it. You should see a folder that contains various files and directories.
5. Copy the contents of this folder to a more permanent location.

Downloading PhoneGap for Symbian

Before downloading and installing PhoneGap, ensure that you meet these requirements:

- You are using a Windows, OS X, or Linux machine.
- If you're on Windows, ensure that you have downloaded and installed cygwin.



NOTE You can download cygwin at www.cygwin.com/setup.exe.

To install PhoneGap, follow these steps:

1. Open your web browser and point it at www.phonegap.com.
2. Click the Download button on PhoneGap's navigation bar. (It should be on the far-right margin.)
3. A ZIP file should download to your Downloads folder.
4. Double-click the ZIP file to uncompress it. You should see a folder that contains various files and directories.
5. Copy the contents of this folder to a more permanent location.

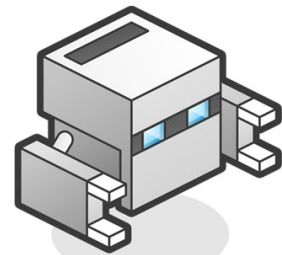
USING PHONEGAP BUILD

If you're in a bit of a hurry, or don't want to support numerous platform SDKs, configurations, and all the other stuff that goes along with trying to push code out to numerous devices, then you really need to take a look at PhoneGap Build service from build.phonegap.com.

This service is designed specifically for coders who want to create their web projects, submit this code, and receive back app-store-ready apps for iOS, Android, webOS, Symbian, BlackBerry, Windows Phone 7, and other devices.

To use PhoneGap Build, you must register for a beta invite, as shown in Figure 2-7.

Within the hour, you will receive an e-mail from PhoneGap with instructions on signing up for the service, as shown in Figure 2-8. This e-mail should contain a link and a signup code.



PhoneGap:Build

Write once. Compile in the cloud. Run anywhere.

Enter your email address

Register for the beta

Already received a beta code? [Create your account](#)

sign in »

Take the pain out of compiling mobile apps for multiple platforms

FIGURE 2-7: Registering for a beta invite at build.phonegap.com

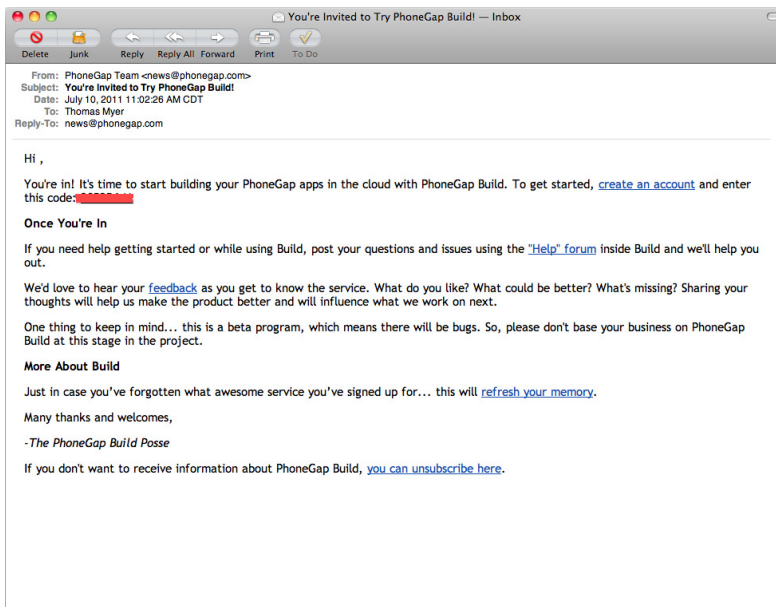
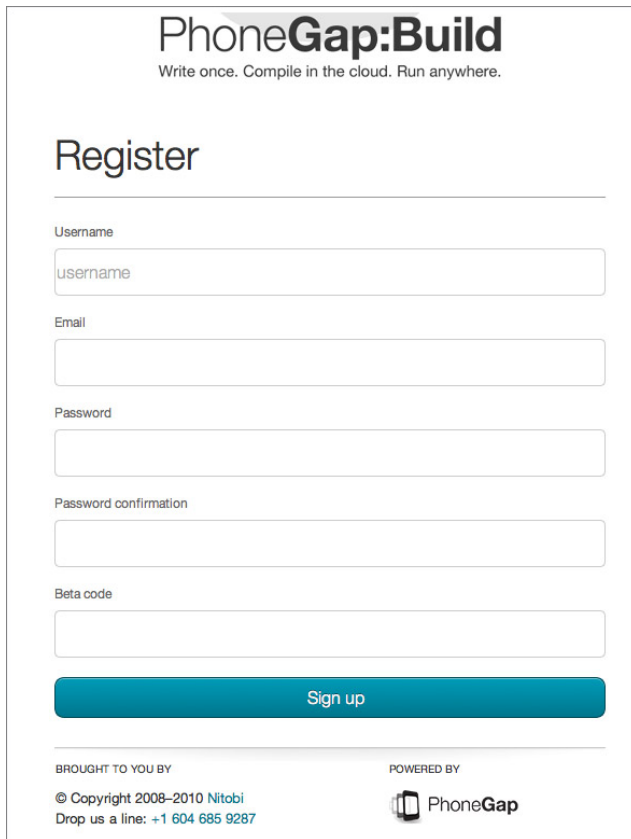


FIGURE 2-8: E-mail acknowledgment

Simply click the link, fill in the registration form shown in Figure 2-9, and use the code from the e-mail as your beta key. As simply as that, you are signed up!



The image shows a registration form for PhoneGap:Build. At the top, the logo "PhoneGap:Build" is displayed with the tagline "Write once. Compile in the cloud. Run anywhere." below it. The form is titled "Register" and contains several input fields: "Username" (with "username" as a placeholder), "Email", "Password", "Password confirmation", and "Beta code". A large teal "Sign up" button is positioned below the input fields. At the bottom, there is a section for "BROUGHT TO YOU BY" with copyright information for Nitobi (© Copyright 2008–2010) and contact details, and a "POWERED BY" section featuring the PhoneGap logo.

PhoneGap:Build
Write once. Compile in the cloud. Run anywhere.

Register

Username

Email

Password

Password confirmation

Beta code

Sign up

BROUGHT TO YOU BY
© Copyright 2008–2010 Nitobi
Drop us a line: +1 604 685 9287


POWERED BY
 PhoneGap

FIGURE 2-9: Registration form

As soon as your registration is complete, you'll see a Welcome screen and a form that lets you create a new application, as shown in Figure 2-10. You can upload your project archive, pull from a `git/` `svn` repo URL, or create a new `git` repository on PhoneGap.com.

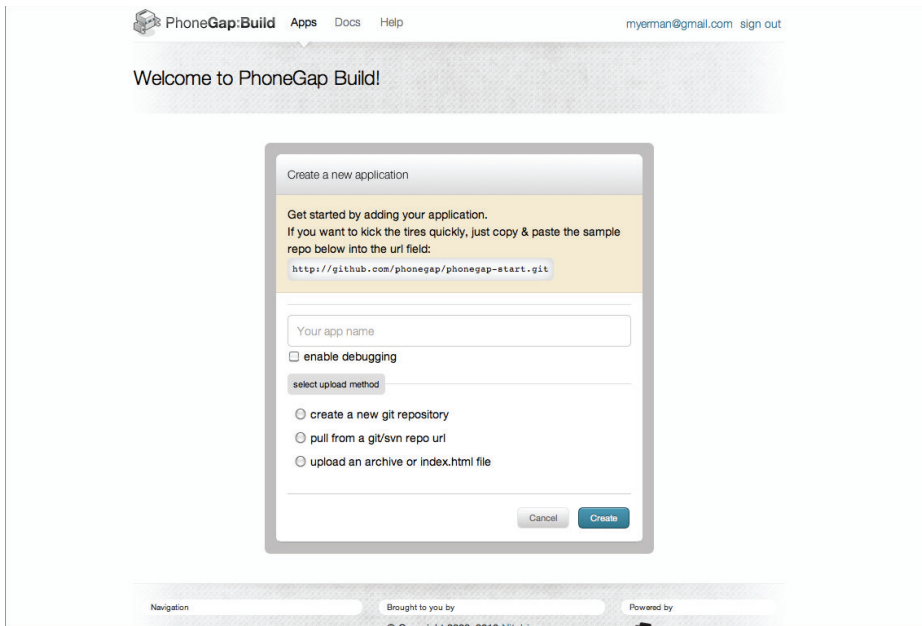


FIGURE 2-10: PhoneGap Build Welcome screen



NOTE Note that, for webOS and Symbian, you'll get back a binary that is ready for distribution. For Android, iOS, and BlackBerry devices, you'll need to provide proper certificates and/or signing keys. For now, the service is free, and will remain free for public open source projects.

CHOOSING YOUR ENVIRONMENT

Many of you won't be using a specific integrated development environment (IDE) to work with your web code. Instead, you'll be using whatever editor you feel most comfortable with. BBEdit (shown in Figure 2-11) is a popular choice. Other good choices include TextMate, Dreamweaver, and vi. Still others of you may choose to use a standard text editor like Notepad on Windows.

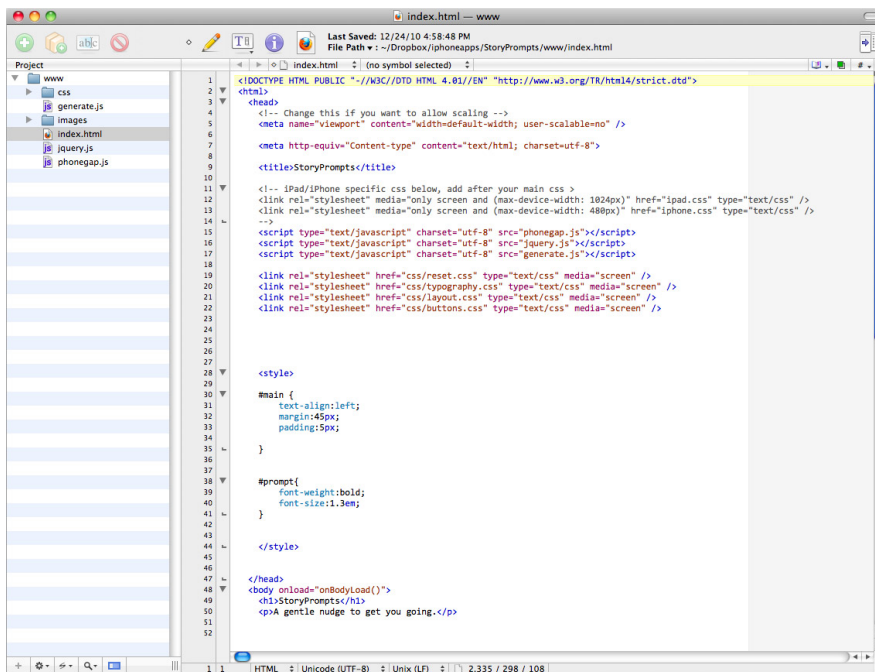


FIGURE 2-11: BBEdit

The point is that it doesn't matter. You can create your web projects in any environment you like. You can use a web browser like Chrome, Safari, Firefox, or Internet Explorer (IE) to view your code in action and debug. Then, when you're ready, copy your project files to Xcode (if you're releasing an iOS application) and do some testing there.

Of course, if you're already fairly used to working in Eclipse or Xcode, feel free to continue working in those environments. Just understand that your process will likely involve having a set of web project files that you then port from one environment to another, especially if you are supporting multiple devices.

SUMMARY

In this chapter, you learned how to install and configure PhoneGap on different platforms.

You are now ready to learn more about the environment you've just installed, and which is what you explore in Chapter 3.

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Installing PhoneGap for iOS devices	The PhoneGap SDK extension for Xcode gives you an extra option on the project-creation screen. Clicking it will load a project with the PhoneGap tools made available to you.
Installing PhoneGap for Android devices	On Android devices, you must install a series of tools and extensions for Eclipse, but you can also use the <code>droidgap</code> wizard (discussed in more detail in Chapter 3).
Installing PhoneGap for BlackBerry devices	You must install the Sun JDK before you install the PhoneGap tools.
Installing PhoneGap for webOS devices	Be sure to install <code>VirtualBox</code> and the webOS SDK before you install PhoneGap.
Installing PhoneGap for Symbian	When you download the PhoneGap SDK, copy the <code>Symbian</code> directory into a more permanent location.
Using <code>Build</code> <code>.PhoneGap.Com</code>	Use <code>Build.PhoneGap.com</code> if you don't want to hassle with all the different options when supporting multiple platforms.

3

Basic Walkthrough

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Using PhoneGap with Xcode
- Using PhoneGap with Android devices
- Using PhoneGap with BlackBerry devices
- Using PhoneGap with other devices

Now that you've performed an installation of PhoneGap, it's time to perform a basic walkthrough of working with PhoneGap in the different environments. The emphasis in this chapter is a quick overview — again, you're going to get more than enough familiarity with the API as you go along. What you need now is some idea as to the environment you've set up, and how to display your code in a simulator so that you can actually run some testing.

USING PHONEGAP WITH XCODE

If you've never worked with Xcode before, it can be a little bewildering, and that's an understatement. In Figure 3-1, you can see that Xcode is divided into five major sections:

- A top toolbar
- “Groups & Files” (left sidebar)
- A file browser that lists files
- A preview pane that shows the contents of a selected file
- A footer that contains messages about your builds, but only after you've run one.

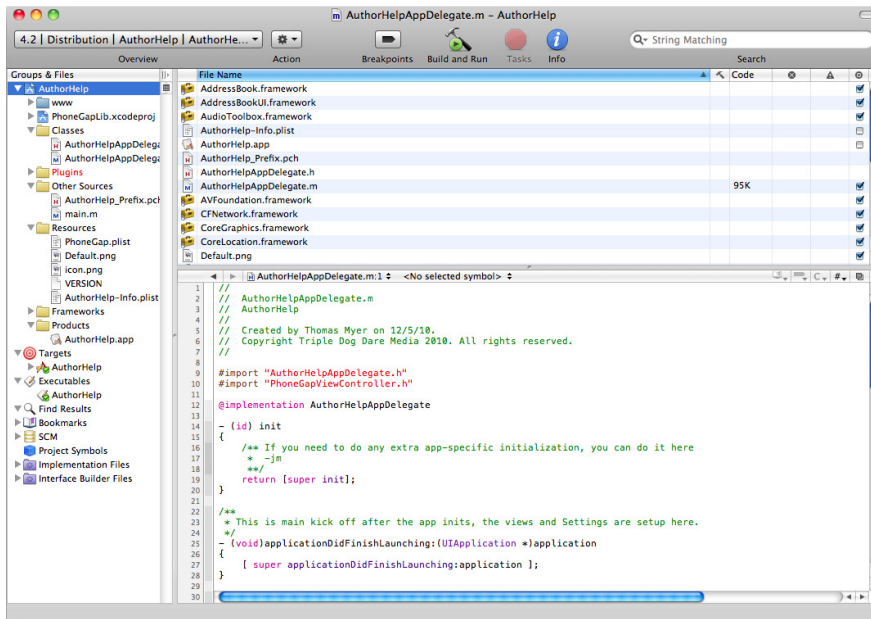


FIGURE 3-1: Major sections of Xcode

The most important part of the toolbar is the “Build and Run” button, which you’ll use to create a build, and then run it in either the simulator or the device. In fact, one thing you should do is to immediately click “Build and Run” when you first create a new PhoneGap project. Figure 3-2 shows what you should see in the simulator (usually a blank page).

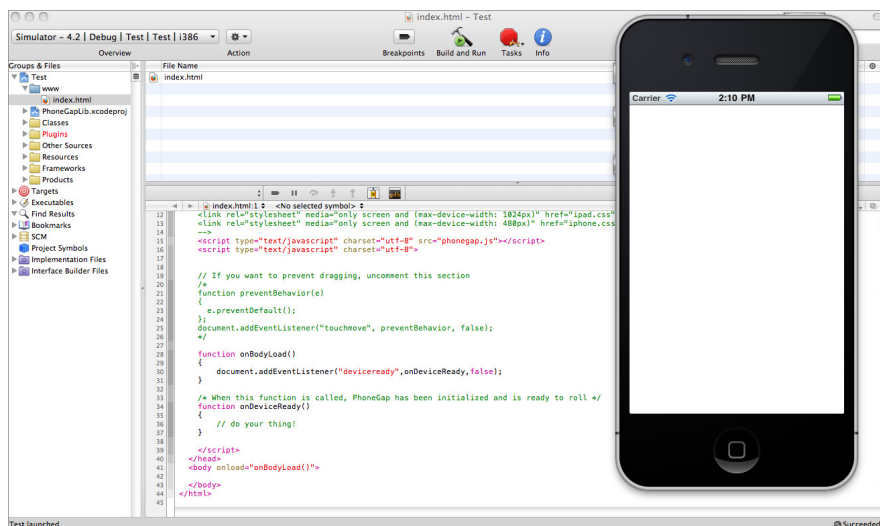


FIGURE 3-2: What appears in the Simulator

Notice that, in the Xcode footer, in the lower-right corner, it says “Succeeded.” If the build had not succeeded, you would see a red error message with the number of problems in the build.



NOTE Please note that in iPhone development, it’s called a simulator, but in Android it’s called an emulator. In either case, it means the same thing — a visual representation (that is, simulation/emulation) of your target device.

The problem with this approach is that you’re dealing with a blank page. How do you know if something actually happened? Well, one thing you could do is to open the `www` directory under “Groups & Files,” click the `index.html` page to load it in the preview pane, and then perform a very small edit.

On line 34 of the HTML file, you’ll notice a function named `onDeviceReady()`. This function runs when the `deviceready` PhoneGap event returns `true`. (You learn more about events in Chapter 4.) Instead of having an empty function, change it so that it fires an alert, as shown here:

```
function onDeviceReady()
{
    alert("I'm ready to get going!");
}
```

Save your work and click the “Build and Run” button again. After a few seconds, you’ll see the Simulator load your app and then, a second later, you’ll see an alert box pop up, as shown in Figure 3-3.

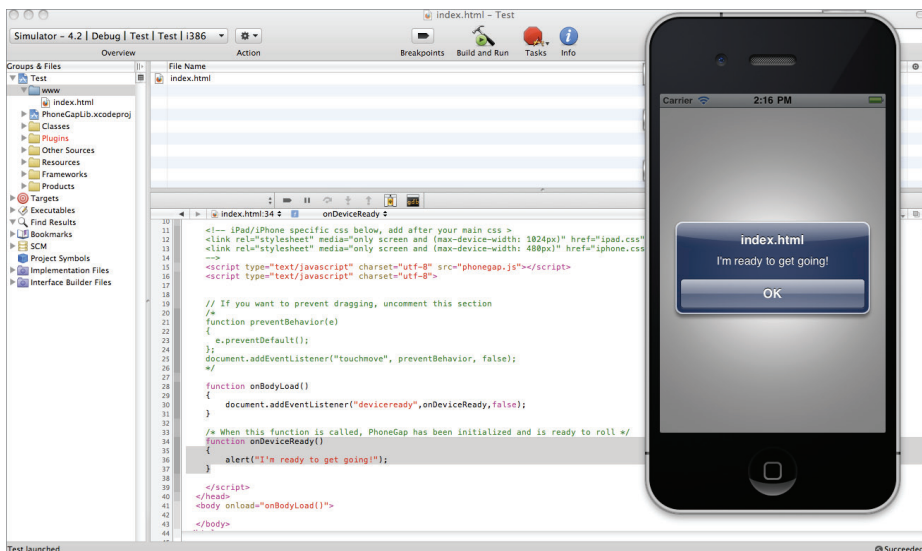


FIGURE 3-3: Alert box

There is an important thing to note here. If you have an error in your JavaScript, HTML, or other code (for example, if you were to remove the last brace on the previous function), Xcode still loads your project to the Simulator or device, and the only clue you'll have that there is a problem is a blank screen. There won't be any further diagnostics or messages from Xcode available to you. This is a fairly compelling reason to develop your code outside of Xcode (in other words, in your favorite editor) and then copy your code to where Xcode can see it.

The question becomes, “Where does that `www` folder reside?” The best way to find out is to right-click the `www` folder in “Groups & Files,” and then choose “Open With Finder” from the pop-up window, as shown in Figure 3-4.

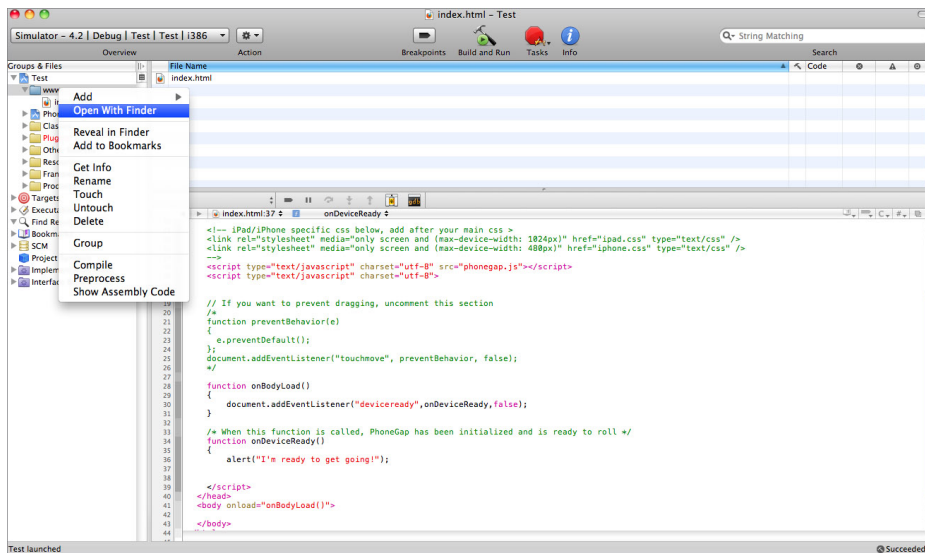


FIGURE 3-4: Choosing “Open With Finder”

This opens a new Finder window that lists all the files in that folder, as shown in Figure 3-5. You could then use your favorite text editor (BBEdit, TextMate, and so on) to work with the files in that folder directly, or you could copy your changes into that folder periodically when you're ready to deploy to a simulator or test device.

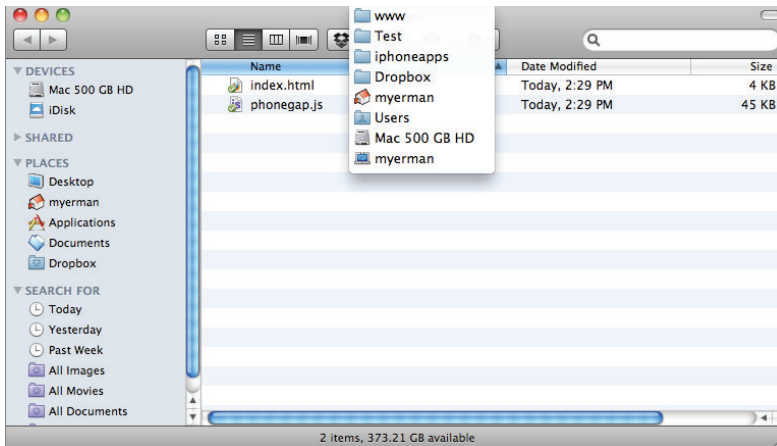


FIGURE 3-5: Viewing the files in a folder



NOTE By the way, you can discover the path of any folder by option-clicking (in Mac OS X) the name of the folder in the Finder. This displays a path to your target folder.

In terms of importance, the `plist` (which stands for property list) file for your project is fairly high on the list. This file lives in the `Resources` directory and usually has a name like `<name_of_your_project>-info.plist`. As shown in Figure 3-6, a `plist` file is basically an XML file that Xcode “prettifies” and displays in a much more user-friendly fashion.

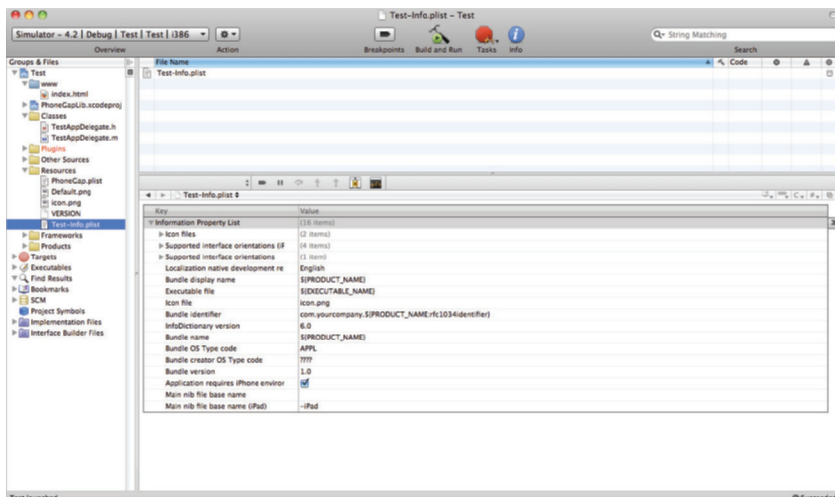


FIGURE 3-6: Viewing a plist file

From an app store perspective, the most important items in the `plist` file are the “Bundle identifier” (which you’ll want to personalize to your own company’s needs, and it *must* match the entry you make in iTunesConnect) and the “Bundle name.” For the most part, you can leave the “Bundle name” the same, but you will absolutely have to change the “Bundle identifier” if you want to publish your app to the store.

Other important items in the `Resources` folder are two images: `icon.png` (which is the icon that users tap to start your app on their iOS devices) and `Default.png` (which is a kind of splash screen that shows up right before your application loads). Both of these `.png` files are shown in Figure 3-7.

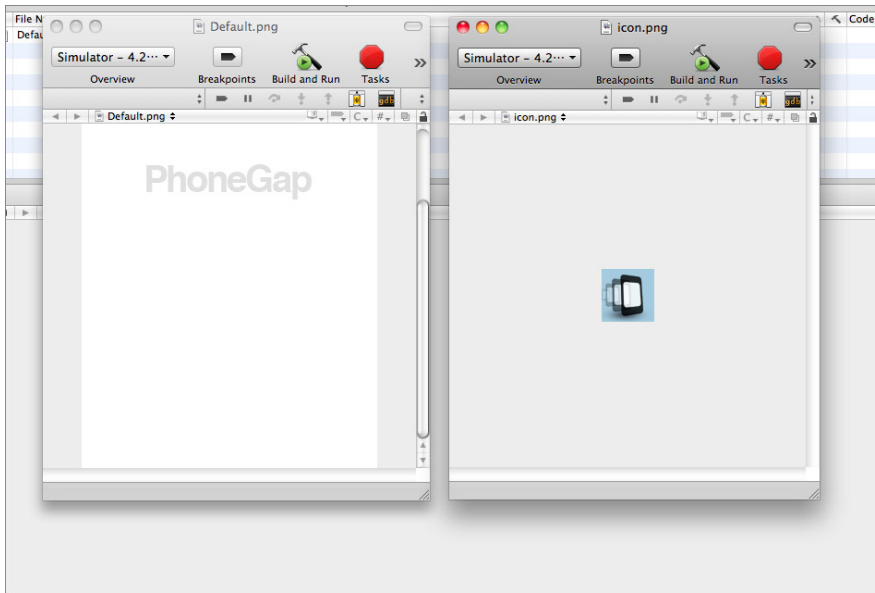


FIGURE 3-7: Images appearing in the `Resources` folder

It’s a good idea to create new versions of these and place them in the `Resources` folder. You can overwrite the existing files. However, be sure that you follow the sizing rules — for example, the icon is 57×57 pixels, and the splash screen is 320×480 , for iPhone/iPod Touch. But if you don’t, be sure that you update the names of the files in the `plist` file.

Last, but not least, are the Objective-C classes that exist in the `Classes`, `Plugins`, and `Frameworks` folders. For the most part, you’ll seldom interact with these files. In Appendix D, you will find a whole bunch of PhoneGap plug-ins that you can add to your projects, and these consist of both Objective-C files (that go in the `Plugins` folder) and JavaScript files (that go in your `www` folder). Other than that, you really shouldn’t mess with the PhoneGap core files.

USING PHONEGAP WITH AN ANDROID DEVICE

Although, in Chapter 2, you learned that you could use Eclipse to launch and test your Android-based PhoneGap apps, it’s much easier to use various command-line tools to create your apps. Remember that, with PhoneGap, you’ll be creating a project containing your HTML, JavaScript, and CSS files — with the right kinds of command-line tools, you could easily prep them for Android.

In Android, you must know to which targets you're going to publish. A *target* is a version of Android. To find out which targets are available, you can run the following command from a Terminal window (if you are using Mac OS X) or a similar shell:

```
android list targets
```

When you run that command on your machine, you should see output similar to the following:

```
Available Android targets:
id: 1 or "android-3"
    Name: Android 1.5
    Type: Platform
    API level: 3
    Revision: 4
    Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
id: 2 or "android-4"
    Name: Android 1.6
    Type: Platform
    API level: 4
    Revision: 3
    Skins: HVGA (default), QVGA, WVGA800, WVGA854
id: 3 or "android-7"
    Name: Android 2.1-update1
    Type: Platform
    API level: 7
    Revision: 2
    Skins: HVGA (default), QVGA, WQVGA400, WQVGA432, WVGA800, WVGA854
id: 4 or "android-8"
    Name: Android 2.2
    Type: Platform
    API level: 8
    Revision: 2
    Skins: HVGA (default), QVGA, WQVGA400, WQVGA432, WVGA800, WVGA854
id: 5 or "android-9"
    Name: Android 2.3
    Type: Platform
    API level: 9
    Revision: 1
    Skins: HVGA (default), QVGA, WQVGA400, WQVGA432, WVGA800, WVGA854
```

You can then use any of these different targets to create an emulator for testing your code. To create an emulator, use the `avd` (“android virtual device”) command, as shown here:

```
android create avd -n <nameOfSimulator> -t <targetName>
```

In this command, `<nameOfSimulator>` is a unique name that you provide (such as `MySimulator`) and `<targetName>` is one of the names from the list of targets (such as `android-7`). A complete command might be as follows:

```
android create avd -n mySimulator -t android-7
```

When you press Enter, you'll see the following come up on the screen. (Just press Enter when asked about the custom hardware profile. This will default to “No.”)

Android 2.1-update1 is a basic Android platform.
 Do you wish to create a custom hardware profile [no]
 Created AVD 'mySimulator' based on Android 2.1-update1,
 with the following hardware config:
 hw.lcd.density=160

To launch the emulator, enter the following command:

```
emulator -avd <nameOfSimulator>
```

Again, *<nameOfSimulator>* is the name you used in the previous command.

The first time you run this command, it will take several minutes to cycle through. You'll see a plain screen that says Android (as shown in Figure 3-8), which will finally transition to an Android home screen (as shown in Figure 3-9).

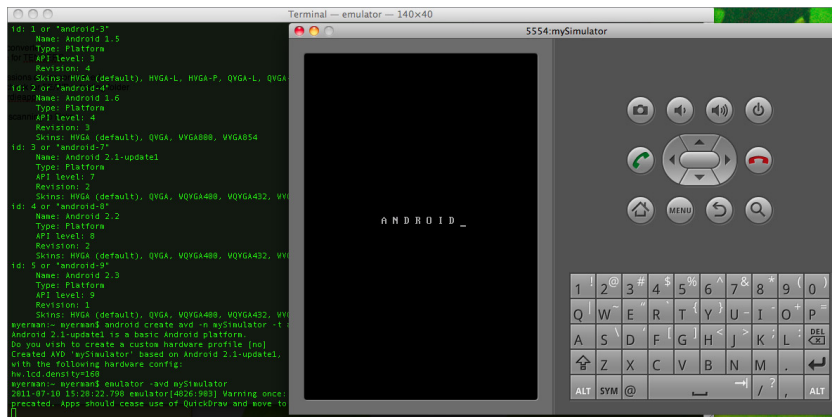


FIGURE 3-8 Initial plain screen

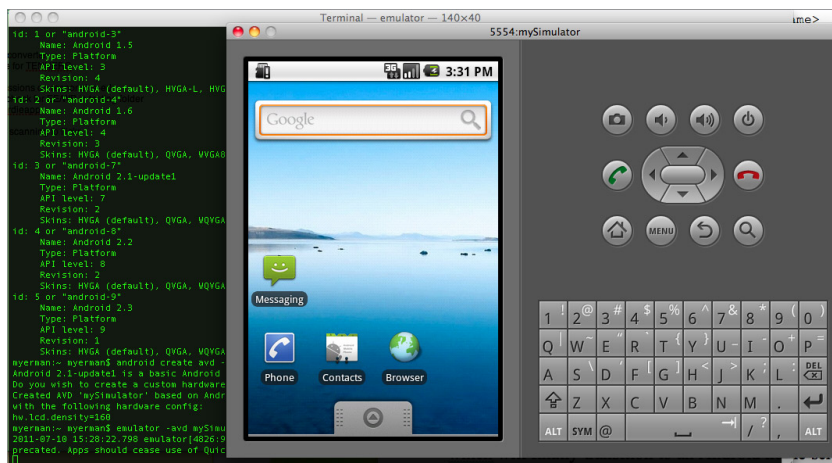


FIGURE 3-9 Android home screen

Note that this emulator can connect to the Internet (as shown in Figure 3-10, where the emulator has been used to connect to a Google Plus account) and do other interesting things. But certain parts of the emulator won't work (such as the camera, as shown in Figure 3-11).

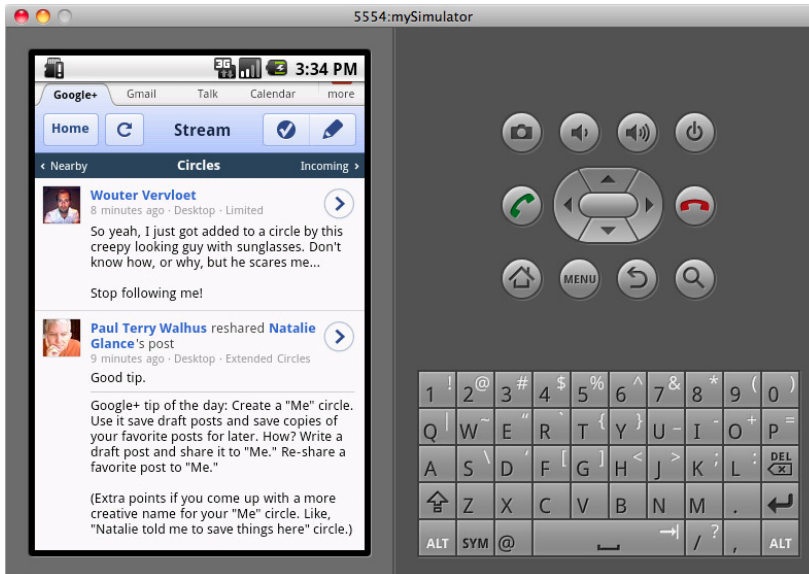


FIGURE 3-10 Emulator connected to the Internet



FIGURE 3-11 Error message indicating the camera is unavailable in the emulator

Another great tool to make your life easier is the `droidgap` wizard. Essentially, you run the following command from the command line:

```
droidgap wiz
```

You then answer a series of questions that relate to your project — where the HTML files live, the name of your application, the package name you want to use, where you want to publish your final project, and which version of Android to target.

Here is a sample run:

```
droidgap wiz
```

```
Public name for your app (e.g. MyApp):  
myTestApp
```

```
Package name for your app (e.g. com.example.myapp):  
com.testing.mytestapp
```

```
Path to your web app directory (e.g. the directory that has your HTML, CSS,  
and JavaScript files):  
~/dropbox/websites/testing
```

```
Path to directory where droidgap should output your files (NOTE - must not exist):  
~/desktop/mytestapp
```

```
Android SDK platform you are targeting (leave blank for a list of available  
targets):  
android-7
```

```
Here we go!
```

```
Making output directory  
Building phonegap.jar  
Creating Android project  
Copying your web app directory  
Creating AndroidManifest.xml  
Copying PhoneGap libs  
Creating strings.xml  
Creating java file
```

```
Done!
```

The result is a `mytestapp` folder copied to the desktop. This folder contains a whole bunch of files and folders that constitute the Android project, as shown in Figure 3-12.

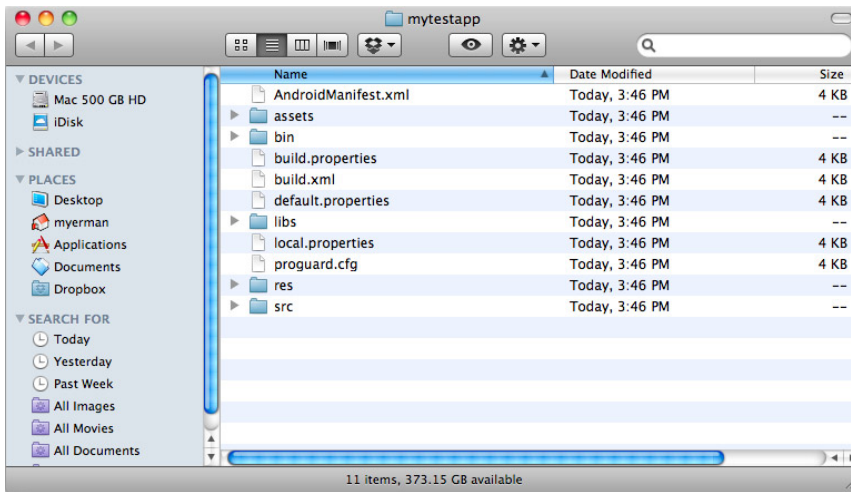


FIGURE 3-12 Files and folders constituting the Android project

To make this specific project run in the emulator, follow these instructions:

1. Ensure that the emulator is running. (See the previous instructions for how to get an emulator to run.)
2. Open a new shell/Terminal window and change directories to your project directory. In the previous example, the target directory was `~/Desktop/mytestapp`.
3. Enter **ant debug** on the command line and press Enter. This creates a `debug.apk` file in the project's `bin` directory. In the case of the previous example, the file will be called `mytestapp-debug.apk`.
4. You can now run the following command to make your project show up in the emulator for further testing:

```
adb -e install -r ~/Desktop/mytestapp/bin/mytestapp-debug.apk
```

5. To open the app in the emulator, simply click its icon, and then test your app to make sure everything works.

USING PHONEGAP WITH A BLACKBERRY DEVICE

To set up a new project for BlackBerry, open a Terminal or shell window and change directories to `C:\Dev\phonegap\BlackBerry\WebWorks`.

On the command line, enter the following:

```
ant create -Dproject.path=C:\Dev\phonegap\BlackBerry\WebWorks\  
    <name of your directory>
```

Here, *<name of your directory>* is the location of your HTML files. On the command line, change directories to that project directory, and then open the `project.properties` file. Edit the following line:

```
bbwp.dir=C:\\BBWP
```

You can now build a project by running `ant build` while you are in the project directory. After that, you can run `ant load-simulator` to run your app in the Simulator.

USING PHONEGAP WITH OTHER DEVICES

If you're working with webOS or Symbian, the process for displaying your app in a Simulator is very similar. In both cases, you want to open a Terminal or shell window and navigate to the folder into which you placed the PhoneGap code, then navigate to either the webOS or Symbian folder (as appropriate).

Next, open the prescribed emulator for your device that you installed along with PhoneGap in Chapter 2, and type **make** in the Terminal window. This deploys your code to the Simulator and lets you test your code.

SUMMARY

This chapter has walked you through the process of working with the various simulators, virtual devices, and other tools needed to actually test your code.

In Chapter 4, you start working directly with the PhoneGap API.

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Working with the iPhone simulator	The iPhone simulator allows you to run your code directly on your Mac, in case you don't have access to an iOS device or need to create and test code quickly.
Working with the Android emulator	The Android emulator lets you quickly test your code without a device.
Working with Blackberry	On BlackBerry, you build and run code via the command line, and then can view it on an emulator.
Working with other devices	On webOS and Symbian devices, you'll have access to command-line tools to help you debug and run your test code.

4

Events

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Understanding events
- Using the events listener
- Understanding event Types

Now that you've learned something about PhoneGap, have installed it, and can make your way around your chosen development environment, it's time to dive into the PhoneGap API.

In this chapter, you learn how PhoneGap handles events. An event in PhoneGap is similar to other JavaScript events. An action has occurred on the device — for example, the document object model (DOM) has loaded and, therefore, the device is “ready” — and now PhoneGap can do other things in response to it.



NOTE *For those of you who have worked in web development, you know that the DOM provides a standardized, versatile view of a document's contents. By using the DOM, you can easily manipulate a document — adding, removing, and editing nodes as needed. For example, if you want to display status messages to the application user, create an HTML `div` with an `id` of `statusmessages`, then use JavaScript to populate that `div` with HTML messages.*

UNDERSTANDING EVENTS

To put it simply, an *event* is any action that can be detected by PhoneGap. In traditional JavaScript programming, any element on a web page can have certain events that can trigger a bit of JavaScript. For example, an `onmouseover` event on a link might cause a pop-up window to appear, or an `onclick` event might cause a preview pane to open.

From that point of view, examples of events might be a mouse click, an image loading, rolling over a certain link or other DOM element, selecting an input field on a form, submitting a form, or even typing a certain letter using the keyboard. For the most part, all of these types of events are also available to you when you're developing PhoneGap applications. But other events are specific to PhoneGap, including the following:

- backbutton
- deviceready
- menubutton
- pause
- resume
- searchbutton
- online
- offline

Of all the events, `deviceready` is the most important one for you to consider. Without it, your application won't know if PhoneGap has fully loaded. Once it has fired, you can safely call any PhoneGap function, which, in turn, can safely access the native API.

To put it even more strongly, the first thing you should do on any PhoneGap project is to listen for the `deviceready` event. Once it fires, you know two things: The DOM has loaded, and so has the PhoneGap API. At that point, you can try to detect other events, or do anything else with your application.

USING THE EVENTS LISTENER

To use any event, you'll want to use an *event listener*. For example, to detect the `deviceready` event, you'll want to do this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap Device Ready Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // call the phonegap api
      }

    </script>
  </head>
  <body>
  </body>
</html>
```

In the preceding code, notice that you are attaching an event listener with `document.addEventListener`. Typically, when the event listener fires for `deviceready`, it means that the HTML document's DOM has loaded. You'll want to use a separate listener for each event that you want to detect and respond to.

All the other event listeners would be registered inside the `onDeviceReady()` function. So, if you wanted to detect a pause or resume event, your code would look like this:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap Device Ready Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // call the phonegap api
        document.addEventListener("pause", onPause, false);
        document.addEventListener("resume", onResume, false);
      }

      function onPause(){
      }

      function onResume(){
      }

    </script>
  </head>
  <body>
  </body>
</html>
```

This is a good place to remind you that `deviceready` may be an event, but it's not a standard browser event. It only works within a PhoneGap context. If you tried to run this event in a normal web browser, it would never fire.

UNDERSTANDING EVENT TYPES

Let's get into a bit more detail for each of the different event types.

backbutton

The `backbutton` event fires when the user presses the Back button on an Android device.

To detect this event, register an event listener as shown here:

```
document.addEventListener("backbutton", onBackButton, false);

function onBackButton(){
    //handle the back button
}
```

As with other events, you shouldn't register this one until you've detected a `deviceready` event:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap backbutton Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // Register the event listener
        document.addEventListener("backbutton", onBackButton, false);
      }

      // Handle the back button
      //
      function onBackButton() {
      }

    </script>
  </head>
  <body>
  </body>
</html>
```

deviceready

As mentioned, the `deviceready` event is the most important event you can detect. In terms of primacy, you must detect it first, before you do anything else, because, once it fires, you are cleared to call the PhoneGap API.

```
document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady(){
    //ready!
}
```

For those who are developing on BlackBerry OS 4.6, the Research in Motion (RIM) Browserfield doesn't support custom events, so `deviceready` will never fire. Instead of checking for `deviceready`, you'll need to use `PhoneGap.available` instead, as shown here:

```

function onLoad() {
    var intervalID = window.setInterval(
        function() {
            if (PhoneGap.available) {
                window.clearInterval(intervalID);
                onDeviceReady();
            }
        },
        500
    );
}

function onDeviceReady() {
    // use the phonegap api!
}

```

menubutton

The `menubutton` event fires when the user presses the Menu button on an Android device.

To detect this event, register an event listener as shown here:

```

document.addEventListener("menubutton", onMenuButton, false);

function onMenuButton(){
    //handle the menu button
}

```

As with other events, you shouldn't register this one until you've detected a `deviceready` event:

```

<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap menubutton Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // Register the event listener
        document.addEventListener("menubutton", onMenuButton, false);
      }

      // Handle the menu button
      //
      function onMenuButton() {
      }

    </script>
  </head>
  <body>
  </body>
</html>

```

pause

The `pause` event fires when an application is put into the background. An application is considered paused when it leaves the foreground, not when it is shut down.

To detect this event, register an event listener as shown here:

```
document.addEventListener("pause", onPause, false);

function onPause(){
    //handle the pause event
}
```

As with other events, you shouldn't register this one until you've detected a `deviceready` event:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap pause Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // Register the event listener
        document.addEventListener("pause", onPause, false);
      }

      // Handle the pause
      //
      function onPause() {
      }

    </script>
  </head>
  <body>
  </body>
</html>
```



WARNING Note that, on iOS devices, any calls that go through Objective-C won't work, nor will any interactive calls (like alerts). Any calls of this nature, as well as any calls made by plug-ins, will occur when your application resumes.

resume

The `resume` event fires when a paused application is put back into the foreground.

To detect this event, register an event listener as shown here:

```
document.addEventListener("resume", onResume, false);

function onResume(){
    //handle the resume event
}
```

As with other events, you shouldn't register this one until you've detected a `deviceready` event:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap resume Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // Register the event listener
        document.addEventListener("resume", onResume, false);
      }

      // Handle the resume
      //
      function onResume() {
      }

    </script>
  </head>
  <body>
  </body>
</html>
```

searchbutton

The `searchbutton` event fires when the user presses the Search button on an Android device.

To detect this event, register an event listener as shown here:

```
document.addEventListener("searchbutton", onSearchButton, false);

function onSearchButton(){
    //handle the search button
}
```

As with other events, you shouldn't register this one until you've detected a `deviceready` event:

```
<!DOCTYPE html>
<html>
  <head>
```

```
<title>PhoneGap searchbutton Example</title>

<script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
<script type="text/javascript" charset="utf-8">

document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady() {
    // Register the event listener
    document.addEventListener("searchbutton", onSearchButton, false);
}

// Handle the search button
//
function onSearchButton() {
}

</script>
</head>
<body>
</body>
</html>
```

online

The online event fires when a PhoneGap application is online (that is, connected to the Internet). This is a new event added with Version 0.9.6, and is supported only on iOS, Android, and BlackBerry devices.

To detect this event, register an event listener as shown here:

```
document.addEventListener("online", isOnline, false);

function isOnline(){
    //handle the online event
}
```

As with other events, you shouldn't register this one until you've detected a deviceready event:

```
<!DOCTYPE html>
<html>
<head>
    <title>PhoneGap online Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady() {
    // Register the event listener
```

```

        document.addEventListener("online", isOnline, false);
    }

    // Handle the online event
    //
    function isOnline() {
    }

    </script>
</head>
<body>
</body>
</html>

```

offline

The `offline` event fires when a PhoneGap application is offline (that is, not connected to the Internet). This is a new event added with Version 0.9.6, and is supported only on iOS, Android, and BlackBerry devices.

To detect this event, register an event listener as shown here:

```

document.addEventListener("offline", isOffline, false);

function isOffline(){
    //handle the offline event
}

```

As with other events, you shouldn't register this one until you've detected a `deviceready` event:

```

<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap offline Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // Register the event listener
        document.addEventListener("offline", isOffline, false);
      }

      // Handle the offline event
      //
      function isOffline() {
      }

    </script>
  </head>
  <body>
  </body>
</html>

```

Now that you know how the different event types are used, it's time to let you try a few things out. First you'll create a simple application that responds to `pause` and `resume` events.

TRY IT OUT A Simple Application That Responds to Events

Enter the following to create a simple application that responds to `pause` and `resume` events:



```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap Event Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // Register the event listeners
        document.addEventListener("pause", onPause, false);
        document.addEventListener("resume", onResume, false);
      }

      // Handle the pause
      //
      function onPause() {
        alert("Paused!");
      }

      // Handle the resume
      //
      function onResume() {
        alert("Resumed!");
      }

    </script>
  </head>
  <body>
  </body>
</html>
```

Code file [chapter4.html] available for download at Wrox.com.

How It Works

The first event listener is the key. It detects when the `deviceready` event fires. Once this event fires, you know you're safe to call the rest of the PhoneGap API.

When it does fire, the `deviceready` event runs the `onDeviceReady()` function, which, in turn, registers two new listeners: one for the `pause` event, and a second for the `resume` event.

That was simple enough. Now it's time to create an Android-specific application that detects when the buttons are pressed.

TRY IT OUT A Simple Application That Responds to Button Events

Enter the following to create a simple application that responds to the Android buttons being pressed:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap Button Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // Register the event listeners
        document.addEventListener("searchbutton", onSearch, false);
        document.addEventListener("menubutton", onMenuButton, false);
        document.addEventListener("backbutton", onBackButton, false);
      }

      // Handle the backbutton
      //
      function onBackButton() {
        alert("You hit the back button!");
      }

      // Handle the menubutton
      //
      function onMenuButton() {
        alert("You hit the menu button!");
      }

      // Handle the searchbutton
      //
      function onSearchButton() {
        alert("You hit the search button!");
      }

    </script>
  </head>
  <body>
  </body>
</html>
```

How It Works

As with the first example, this simple application first checks to ensure that the PhoneGap application is ready, and then registers three event listeners, one for each of the Android device buttons (Search, Menu, and Back buttons).

That's all there is to it — you now know the basics of events. From here, you learn how to handle the rest of the PhoneGap API.

SUMMARY

In this chapter, you learned how to detect the different PhoneGap events, specifically `deviceready`, `online`, `offline`, `pause`, `resume`, `menubutton`, `searchbutton`, and `backbutton`.

In Chapter 5, you'll learn how to get information on your device and detect a network.

EXERCISES

1. Open the Google homepage as soon as you detect that the PhoneGap application is online.
2. Display the current time when the device is ready.



NOTE *Answers to the Exercises can be found in Appendix A.*

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
deviceready	To detect when the device is ready, use the <code>deviceready</code> event.
pause	To detect when the application has been paused, use the <code>pause</code> event.
resume	To detect when the application has been resumed, use the <code>resume</code> event.
menubutton	To detect when the Menu button has been pressed on an Android device, use the <code>menubutton</code> event.
searchbutton	To detect when the Search button has been pressed on an Android device, use the <code>searchbutton</code> event.
backbutton	To detect when the Back button has been pressed on an Android device, use the <code>backbutton</code> event.
online	To detect when the application is online (connected to the Internet), use the <code>online</code> event.
offline	To detect when the application is offline (not connected to the Internet), use the <code>offline</code> event.

5

Working with the Device, the Network, and Notifications

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Discovering information about your device
- Determining the network status
- Creating custom notifications

In Chapter 4, you learned about PhoneGap life cycle events — how to tell when the device was ready, how to pause and resume your application, and how to tell whether certain buttons were pressed on an Android device.

In this chapter, you take the next logical step and learn more about the device itself, network connectivity, and custom notifications.

GETTING INFORMATION FROM THE DEVICE

PhoneGap has a runtime object called `device` that holds useful information about the device. Following are the elements of `device`:

- `device.name` — The name of the device (for example, my iPhone)
- `device.phonegap` — The version of PhoneGap
- `device.platform` — The type of device, (for example, iPhone)
- `device.uuid` — The unique number of the device
- `device.Version` — The version of the OS that is running

It's important to note that `device` is assigned to the `window` object, so it is implicitly in the global scope. In other words, both of the following variables reference the same device:

```
//both of these reference the same device
var myPhoneName = window.device.name;
var myPhoneName = device.name;
```

In this section, you learn how to get the following:

- The device's name
- The PhoneGap version
- The device's platform
- The device's Universally Unique Identifier (UUID)
- The device's operating system version

Getting the Device Name

To get the device's name, use `device.name`, as shown here:

```
var myPhoneName = device.name;
```

The value that's returned is set by the manufacturer, and can vary from product to product, and even across versions of the same product. For example, here is a quick list of return values for various phones:

- Android Nexus One returns `Passion` (which is a code name).
- Android Motorola Droid returns `voles`.
- BlackBerry Bold 8900 returns `8900`.
- iPhone returns the name set in iTunes (for example, `Tom's phone`).

The `device.name` feature is supported on Android, BlackBerry, and iPhone devices.



NOTE Remember, on an Android, the `device.name` returned is often the code name of the product, instead of the model name. On iPhone, this returns whatever name has been set for the device in iTunes.

Getting the PhoneGap Version

To get the version of PhoneGap on the device, use `device.phonegap`, as shown here:

```
var myDevicePhoneGap = device.phonegap;
```

The `device.phonegap` feature is supported on Android, Blackberry, and iPhone devices.

Getting the Device Platform

To get the device's operating system name, use `device.platform`, as shown here:

```
var myDevicePlatform = device.platform;
```

Depending on the device, this command returns the following:

- Android
- Blackberry
- iPhone
- webOS

The `device.platform` feature is supported on Android, Blackberry, and iPhone devices.



NOTE Experienced iPhone developers might expect an iPhone device to return `iOS` (which is how Apple has rebranded its devices). Also, some BlackBerry devices might return the platform version instead of the platform name. For example, it might return `1.10.3.5` instead of `BlackBerry`.

Getting the Device UUID

Every device should have a UUID assigned to it by the device manufacturer. The UUID can be different lengths; depending on the device, it should always be specific to a model and platform. For example, Android uses a random 64-bit integer; BlackBerry uses a nine-digit personal identification number (PIN); and iPhone uses a string of hash values.

To get the device's UUID, use `device.uuid`, as shown here:

```
var myDeviceID = device.uuid;
```

The `device.uuid` feature is supported on Android, BlackBerry, and iPhone devices.

Getting the Operating System Version

To get the device's operating system, use `device.version`, as shown here:

```
var myDeviceOS = device.version;
```


Depending on the operating system, you'll get different return values. For example, on Android, Froyo OS returns `2.2` and Éclair OS returns `2.1`, `2.0.1`, or `2.0`. The BlackBerry Bold 9000 using OS 4.6 returns `4.6.0.282`. An iPhone running iOS 3.2 returns `3.2`.

The `device.version` feature is supported on Android (2.1 and higher), BlackBerry, and iPhone devices.

Now that you’ve learned how to get the different individual pieces of information about your device, let’s put everything you know into action in the next “Try It Out” section.

TRY IT OUT Getting Device Information

In this exercise, you create a simple script that lets you use the device elements to return information about your device. Enter the following code:



Available for download on Wrox.com

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

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

        // Use an event listener to detect if PhoneGap is ready
        //
        function onLoad() {
            document.addEventListener("deviceready", onDeviceReady, false);
        }

        // okay, PhoneGap is ready
        //
        function onDeviceReady() {
            var myDiv = document.getElementById('props');

            myDiv.innerHTML = 'Device Name: ' + device.name + '<br />' +
                              'Device PhoneGap: ' + device.phonegap + '<br />' +
                              'Device Platform: ' + device.platform + '<br />' +
                              'Device UUID: ' + device.uuid + '<br />' +
                              'Device Version: ' + device.version + '<br />';
        }

    </script>
</head>
<body onload="onLoad()">
    <p id="props">Loading device properties...</p>
</body>
</html>
```

Code file [chapter5a.html] available for download at Wrox.com.

How It Works

In this exercise, you created a very basic HTML document that first loads the `phonegap.js` library. Next, you wrote an `onLoad()` function that registers an event handler using `addEventListener`. Specifically, you want to listen for the `deviceready` event, and when that fires, run the `onDeviceReady()` function.

The `onDeviceReady()` function does two things. First, it pulls in the paragraph with the document object model (DOM) ID "props" using `getElementById`, and assigns it to the `myDiv` variable. It then prints out device information to that DOM ID using the `innerHTML` function.

If you were using jQuery, you could simplify the process like this:

```
function onDeviceReady() {
    $("#props").html('Device Name: ' + device.name + '<br />' +
        'Device PhoneGap: ' + device.phonegap + '<br />' +
        'Device Platform: ' + device.platform + '<br />' +
        'Device UUID: ' + device.uuid + '<br />' +
        'Device Version: ' + device.version + '<br />');
}
```

Note that the rest of the HTML document is fairly simple, but contains vital details. For example, the `body` tag uses the `onload` parameter to initiate the `onLoad()` function, which registers the event handler that detects for `deviceready`. Furthermore, the single paragraph has a DOM ID of "props", which is what will be populated with the results of calling `device.name`, `device.phonegap`, `device.platform`, `device.uuid`, and `device.version`.

Now that you've successfully created a script to get device information, it's time to learn how to detect a network with your device.

CHECKING FOR A NETWORK

If you're a web or desktop developer, you often take it for granted that the network is always present, either via a wired or wireless connection. When you start working with mobile devices, you can't always take the network's presence for granted. The signal may be nonexistent, or it may be very weak, or the user might switch from cellular to WiFi, or back again.

Luckily for you, the PhoneGap API includes a `Connection` object, which gives you access to the device's cellular and WiFi connection information.

In this section, you learn how to determine the connection type.

Determining the Connection Type

To determine what kind of network connection can be made, you'll be using `connection.type`.

Following is an example:

```
function checkConnection(){
    var myState = navigator.network.connection.type;

    //return a specific state
}
```

The `connection.type` function will return one of a possible list of connection types available:

- UNKNOWN
- ETHERNET
- WIFI
- CELL_2G
- CELL_3G
- CELL_4G
- NONE

It's a good idea to dress up these messages by providing users with a customer notification. For example:

```
function checkConnection() {
    var networkState = navigator.network.connection.type;

    var states = {};
    states[Connection.UNKNOWN] = 'Unknown connection';
    states[Connection.ETHERNET] = 'Ethernet connection';
    states[Connection.WIFI] = 'WiFi connection';
    states[Connection.CELL_2G] = 'Cell 2G connection';
    states[Connection.CELL_3G] = 'Cell 3G connection';
    states[Connection.CELL_4G] = 'Cell 4G connection';
    states[Connection.NONE] = 'No network connection';

    alert('Connection type: ' + states[networkState]);
}
```

For example, if the connection type were `NONE`, then the alert box would contain the message `No network connection`.

Now that you've learned how to determine the connection type, it's time to try out a complete script.

TRY IT OUT Checking Network Availability

In this exercise, you create a simple script that checks for network connectivity. Enter the following code:



Available for
download on
Wrox.com

```
<!DOCTYPE html>
<html>
  <head>
    <title>Connectivity Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady() {
```

```

        checkConnection();
    }

    function checkConnection() {
        var networkState = navigator.network.connection.type;

        var states = {};
        states[Connection.UNKNOWN] = 'Unknown connection';
        states[Connection.ETHERNET] = 'Ethernet connection';
        states[Connection.WIFI] = 'WiFi connection';
        states[Connection.CELL_2G] = 'Cell 2G connection';
        states[Connection.CELL_3G] = 'Cell 3G connection';
        states[Connection.CELL_4G] = 'Cell 4G connection';
        states[Connection.NONE] = 'No network connection';

        alert('Connection type: ' + states[networkState]);
    }

</script>
</head>
<body>
    <p>A dialog box will report the network state.</p>
</body>
</html>

```

Code file [chapter5b.html] available for download at Wrox.com.

How It Works

Once more, you have built a very simple HTML document. First, you load the `phonegap` library, and then register the event handler for `deviceready` events.

Once the device is ready, you run `navigator.network.isReachable` on `google.com`. This website is typically always up, so it's a good test for network connectivity. (In other words, you might not want to test a server that might be down when you do the test.)

The most important bit about this attempt to connect is the callback function (imaginatively named `myCallback`), which will contain the `reachability` code returned by the `network.isReachable` function.

Once again, note that some devices will store the `reachability` code in different formats, so check for both `reachability.code` first (this covers iPhones) and then `reachability` (for Android and other devices).

Next, you set up a simple object that contains the network status constants from PhoneGap as keys, with your own custom messages as values — that way, you avoid any cryptic-looking messages being delivered to the person using the device.

Next, you run a JavaScript `alert()` that contains the connectivity message. This will pop open a window on the device that the user can then close.

Finally, be sure to run the `onLoad()` function from the `body` tag, because this will set the entire process in motion.

In the next section, you learn how to use custom PhoneGap notifications to enhance the user experience.

USING NOTIFICATIONS

In the previous “Try It Out” exercise, you used a standard JavaScript `alert()` function to display important information to the user (specifically, the status of network connectivity). Although you can use this kind of alert system on a device, PhoneGap comes with four different types of notifications you can use.

In this section, you learn how to use the following:

- Alerts
- Confirmation dialogs
- Beeps
- Vibrations

Using Alerts

To show a custom alert or dialog box, use the `notification.alert` function, as shown here:

```
navigator.notification.alert(message, callback, [title], [button]);
```

This function takes two required and two optional parameters, in the following order:

- `message` — A string that contains the dialog message (for example, “The network status is WIFI only”).
- `callback` — The callback function to invoke when the alert is dismissed.
- `title` — A string that contains the alert’s title (optional).
- `button` — A string that contains the name of the button (for example, “OK”) (optional).

The `notification.alert` function is available on Android, BlackBerry OS 4.6, webOS, and iPhone devices.

Here’s a complete example with callback:

```
function gameOverDismissed() {  
    // calculate or store their final score...  
}  
  
navigator.notification.alert(  
    [AU: Be sure you replace all tabs in your code with five spaces.]  
    'Game Over!',           // message  
    gameOverDismissed,      // callback  
    'Game Over',           // title  
    'Done'                  // buttonName  
);
```


If you're on BlackBerry or webOS, note that you will only be able to send in a message (no titles, button names, or callbacks available), as shown here:

```
//BlackBerry 4.6 / webOS
navigator.notification.alert('Game Over! ');
```

Using Confirmation Dialogs

A confirmation dialog is similar to an alert, except that it may contain multiple buttons, each of which might fire off a different process. For example, a confirmation dialog that asks, “Do you wish to continue?” might have a Yes and a No button. Selecting one or the other results in different outcomes.

To create a confirmation dialog, use the `notification.confirm` function. You'll need to wrap it in a function, because it is generally called from a link or button on the HTML interface, as shown here:

```
// process the confirmation dialog result
function onConfirm(button) {
    alert('You selected button ' + button);
}

// Show a custom confirmation dialog
//
function showConfirm() {
    navigator.notification.confirm(
        'Game Over!', // message
        onConfirm,    // callback to invoke with index of button pressed
        'Game Over',  // title
        'Restart,Exit' // buttonLabels
    );
}
```

Using Beeps

Sometimes you don't need an alert or confirmation dialog — a simple beep will do when a user does something. To create a beep, use the `notification.beep` function, as shown here:

```
navigator.notification.beep(2);
```

The function takes a single argument, an integer that indicates how many times you want it to beep. This function is available on Android, BlackBerry, webOS, and iPhone devices.



NOTE Note that Android will play the default Notification ringtone specified under the Settings ⇄ Sound and Display panel. The iPhone will also ignore the beep count argument.

Using Vibrations

Sometimes it's not appropriate to have the device make a sound. For these kinds of situations, PhoneGap provides a `vibrate` function available as `notification.vibrate`, which takes a duration argument in milliseconds, as shown here:

```
navigator.notification.vibrate(2000);
```

The `vibrate` function is available on Android, BlackBerry, webOS, and iPhone devices.



NOTE The iPhone will ignore the milliseconds argument and vibrate for a set amount of time.

Now that you've learned how to use the different notifications, the next “Try It Out” exercise walks you through using them all in the context of a complete script.

TRY IT OUT Using All Four Notifications

In this exercise, you use all the notifications examined so far. Enter the following code:



Available for
download on
Wrox.com

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

<script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
<script type="text/javascript" charset="utf-8">

// Wait for PhoneGap to load
//
function onLoad() {
    document.addEventListener("deviceready", onDeviceReady, false);
}

// PhoneGap is ready
//
function onDeviceReady() {
    // Empty
}

// Show a custom alert
//
function showAlert() {
    navigator.notification.alert(
        'Game Over!', // message
        alertCallback, //callback
        'Game Over', // title
```

```

        'Done' // buttonName
    );
}

//alert call back
function alertCallback(){
    //do something... like calculate final score
}

// process the confirmation dialog result
function onConfirm(button) {
    alert('You selected button ' + button);
}

// Show a custom confirmation dialog
//
function showConfirm() {
    navigator.notification.confirm(
        'Game Over!', // message
        onConfirm,    // callback to invoke with index of button pressed
        'Game Over', // title
        'Restart,Exit' // buttonLabels
    );
}

// Beep twice
//
function playBeep() {
    navigator.notification.beep(2);
}

// Vibrate for 4 seconds
//
function vibrate() {
    navigator.notification.vibrate(4000);
}

</script>
</head>
<body onload="onLoad()">
    <p><a href="#" onclick="showAlert(); return false;">Show Alert</a></p>
    <p><a href="#" onclick="showConfirm(); return false;">Show Confirmation</a></p>
    <p><a href="#" onclick="playBeep(); return false;">Play Beep</a></p>
    <p><a href="#" onclick="vibrate(); return false;">Vibrate</a></p>
</body>
</html>

```

Code file [chapter5c.html] available for download at Wrox.com.

How It Works

As with previous “Try It Out” exercises, for this example, you have created a very simple HTML document that loads the `phonegap` libraries, and then registers a handler for the `deviceready` event.

However, instead of actually doing anything, the `onDeviceReady()` function is left empty, because the main document contains a series of links that call functions that, in turn, fire off the different notifications.

For example, by clicking the first link, the `showAlert()` function is triggered, which displays a custom alert box. The second link triggers the `showConfirm()` function, which displays a custom confirmation box. The third link triggers `playBeep()`, and the fourth, `vibrate()`.

SUMMARY

In this chapter, you learned how to get information from the device, detect the presence of the network, and send notifications to the user using a variety of options.

In Chapter 6, you will learn how to use the accelerometer.

EXERCISES

1. Create a simple HTML form that allows a user to enter a web address and then press a button to test the network connectivity. Have the default site be `google.com`.
 2. Create an informational footer that contains information about the device the person is using. (Extra points are awarded if you add a button that allows the user to show/hide the footer.)
 3. Create a confirmation dialog that prompts the user to push one button to ring a bell, and another button for vibrate.
-



NOTE *Answers to the Exercises can be found in Appendix A.*

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
Getting the device name	Get the device's name with <code>device.name</code> .
Getting the version	Get the device's PhoneGap version with <code>device.phonegap</code> .
Getting the device platform	Get the device's platform with <code>device.platform</code> .
Getting the device UUID	Get the device's UUID with <code>device.uuid</code> .
Getting the device OS version	Get the device's OS version with <code>device.version</code> .
Checking for network connectivity	Use <code>connection.type</code> to determine network reachability.
Possible network connection types	The possible network connection types are UNKNOWN, ETHERNET, WIFI, CELL_2G, CELL_3G, CELL_4G, and NONE.
Creating custom alerts	Create custom alerts with <code>notification.alert</code> .
Creating custom notifications	Create custom notifications with <code>notification.confirm</code> .
Creating beeps	Create beeps with <code>confirmation.beep</code> .
Creating vibrations	Create vibrations with <code>confirmation.vibrate</code> .

6

Accelerometer

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Learning about the accelerometer
- Using accelerometer methods
- Learning about options and arguments

So far, you've learned about basic events and how to detect them, and you've learned how to get basic information about the device, its network connectivity, and about notifications.

In the next three chapters, you're going to learn about the accelerometer, the compass, and geolocation, starting with the accelerometer.

GETTING TO KNOW THE ACCELEROMETER

In this section, you'll learn what the accelerometer is and what you can use it for.

What Is the Accelerometer?

The *accelerometer* is a device that captures motion in the X, Y, and Z directions.

To help visualize the X, Y, and Z directions, place your device face-up on a flat surface, like a desk. Ideally, this surface would offer very little friction so that, for example, you can push the device left and right without having to lift it. For the purposes of this example, let's assume that wherever you put the device is 0, 0, 0 on a coordinate system. Any which way you move the device will result in a change of coordinates, either into the positive or negative quadrants.

If the device is flat on the surface of the desk, and you move it to the left or right, you’ve moved it along the X axis. Moving it to the left gives you a negative result; moving it to the right gives you a positive result. If you were to move the device away from you on the desk or toward you, you’re moving it on the Y axis. Moving it away would result in a positive result along the axis; and moving it toward you would result in a negative result.

So far, so good. You’re dealing with very simple coordinates that were covered in high-school algebra. The Z direction represents up and down, adding a third dimension. If you were to pick up the device off the surface and bring it closer to your face, you’d result in a positive Z axis value. If you were to drop the device under the surface (or somehow push it through the surface!), you’d end up with a negative value along the Z axis.

Of course, it’s very difficult to move something along just one axis without affecting the other axes. If you were to pick up a device (and, thus, move it closer to your face), you’d end up in the positive Z-index quadrant, but it’s also quite likely that you’ll bring it closer to your body (negative Y-axis). If you were tracking the movement of the device every second (or less), you’d also detect some movement along the X-axis as you moved it.

Now, imagine what would happen if you tossed the device back onto the surface of the desk — don’t toss it too hard, or you’ll break it! The accelerometer inside the device would detect all kinds of movement along the X, Y, and Z directions.

Using the Accelerometer

Okay, you’re probably saying to yourself, “All of this accelerometer stuff is pretty neat — but what would I use it for?” Remember that the accelerometer can detect motion, tilt, and acceleration, so there are plenty of creative applications.

Here are just three:

- How about creating an app that detects motion in the person (say, like jumping jacks or jumping rope) and keeps track of them while the person exercises? All that person would have to do is turn on your application and put the device in a pocket.
- For low-light situations, you could automatically take a photo with the device’s camera if the device is perfectly still.
- You could build a game in which you control pieces that react to the tilt or motion of the device.

Showcase of Accelerometer Applications

If you own an iPhone, you’ve probably either heard about or played “DoodleJump,” shown in Figure 6-1. “DoodleJump” is an incredibly addictive game that lets your character jump from level to level using only the accelerometer to detect your left/right movement.

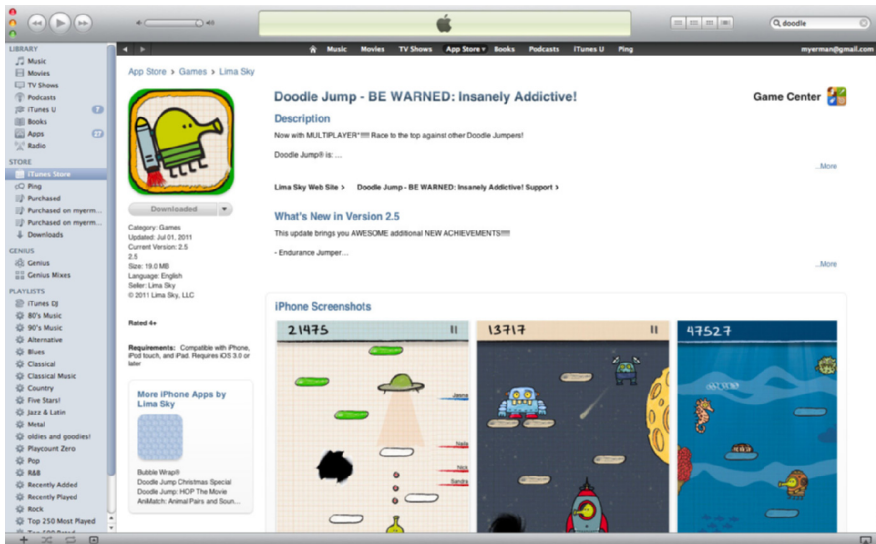


FIGURE 6-1: “DoodleJump” application

Another popular iPhone game is “Super Monkey Ball,” shown in Figure 6-2. The idea behind this game is to guide a cute little monkey encased in a transparent ball through a variety of obstacles by tilting and turning your iPhone.

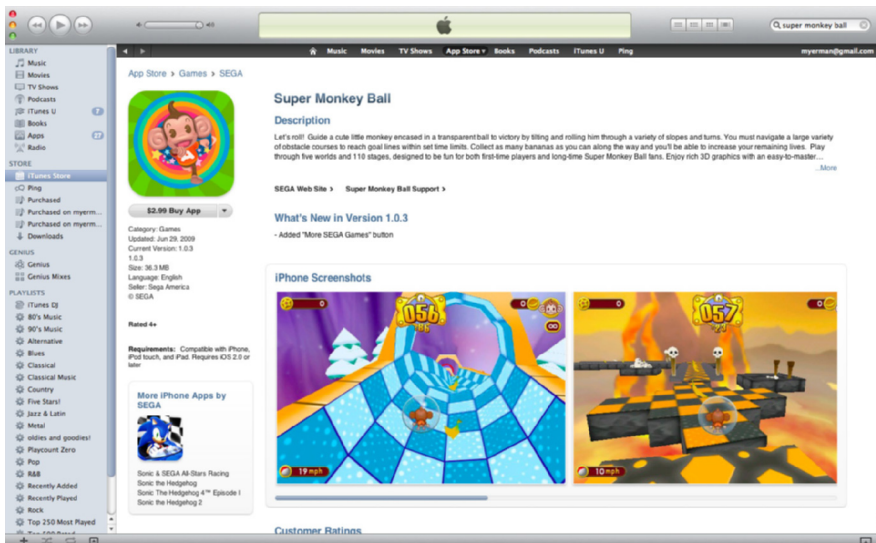


FIGURE 6-2: “Super Monkey Ball” application

On the Android side, you have “Hyperspace” (shown in Figure 6-3), in which you use tilt and movement to control a ball as it shoots across a series of obstacles. The game was featured on the Syfy channel in the U.S.

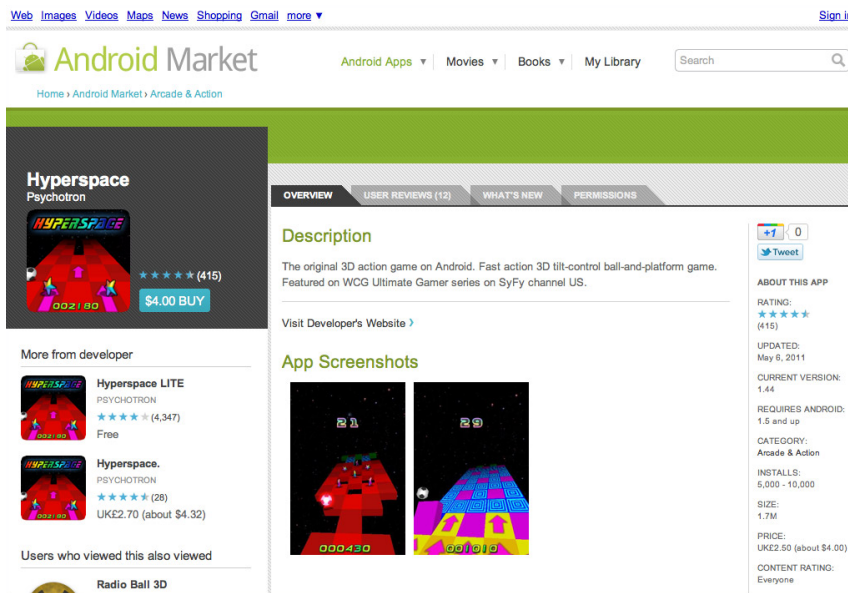


FIGURE 6-3: “Hyperspace” application

Of course, there have been any number of very silly apps that use the accelerometer. On the iPhone, there were various wine and beer apps that let you “pour” out a drink when you tilt the device. On Android, there’s “Lighter,” a virtual flame that turns in whatever direction you tilt the device, as shown in Figure 6-4.

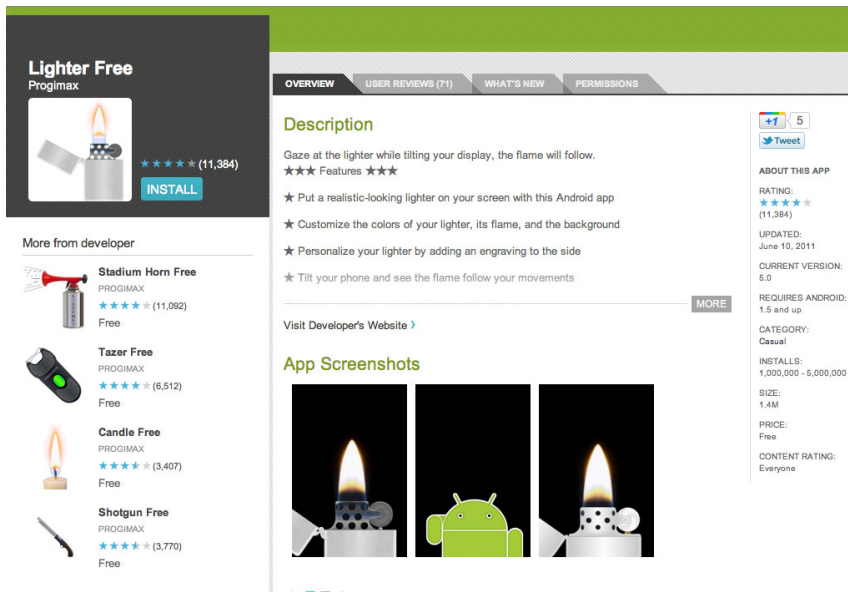


FIGURE 6-4: “Lighter” application

On a more serious side, there's "Sensorfit" for the Android (shown in Figure 6-5), which detects your movement with the accelerometer and tracks your fitness.

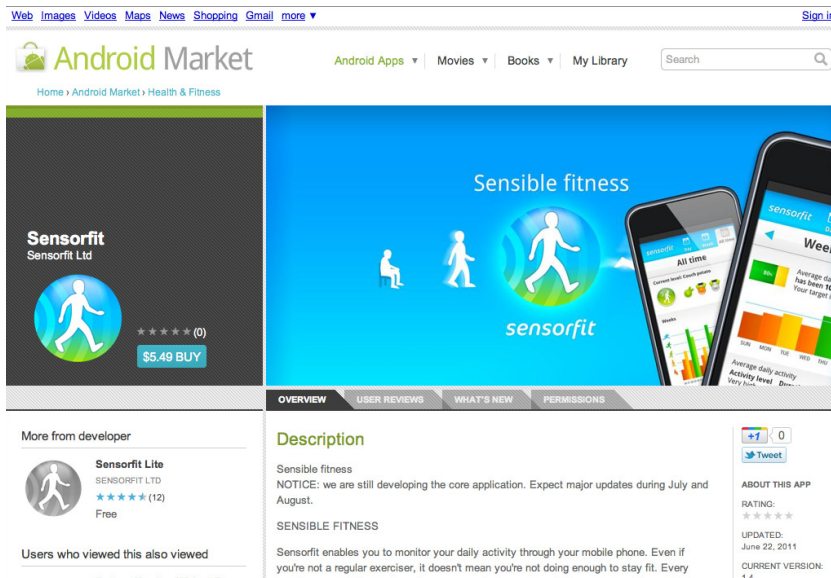


FIGURE 6-5: "Sensorfit" application

The BlackBerry also has a built-in accelerometer, so you have games like "Jumper" (shown in Figure 6-6). This game is similar to "DoodleJump" in that you control a character's jumping by moving left and right.

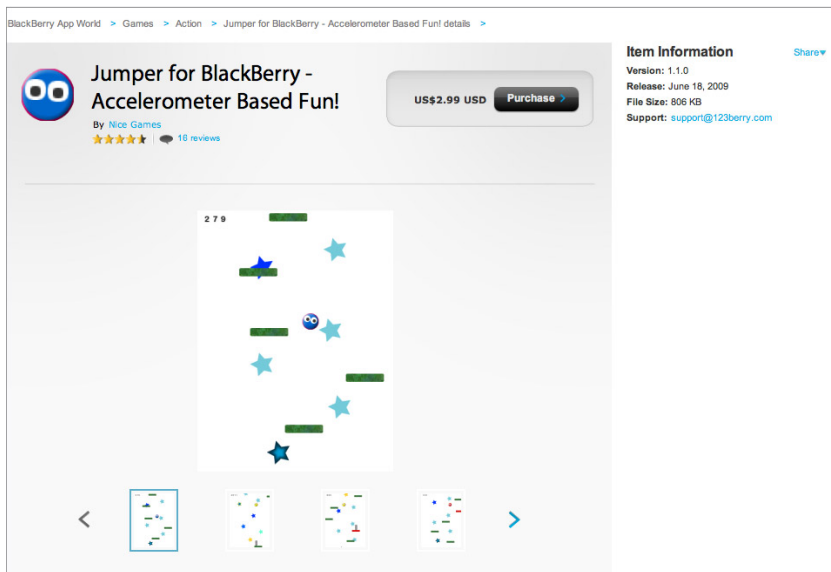


FIGURE 6-6: "Jumper" application

USING THE ACCELERATION OBJECT

The `Acceleration` object is a read-only object that contains accelerometer data captured at a specific point in time. Following are the object's properties:

- `x` — Amount of motion on the X-axis, expressed as a number.
- `y` — Amount of motion on the Y-axis, expressed as a number.
- `z` — Amount of motion on the Z-axis, expressed as a number.
- `timestamp` — Timestamp for creation, expressed in milliseconds.

This object is created and populated by PhoneGap, and returned by an `accelerometer` method. Following is an example:

```
navigator.accelerometer.getCurrentAcceleration(onSuccess, onError);

//you now have access to an acceleration object
//which contains x, y, z, and timestamp data
function onSuccess(acceleration) {
    alert('Acceleration X: ' + acceleration.x + '\n' +
        'Acceleration Y: ' + acceleration.y + '\n' +
        'Acceleration Z: ' + acceleration.z + '\n' +
        'Timestamp: '      + acceleration.timestamp + '\n');
};

function onError() {
    alert('Sorry! Error!');
};
```

USING ACCELEROMETER METHODS

In this section, you'll learn how to use the main methods of the accelerometer.

getCurrentAcceleration

To get the current acceleration along the X, Y, and Z axis, use `accelerometer.getCurrentAcceleration`, as shown here:

```
navigator.accelerometer.getCurrentAcceleration(accelerometerSuccess,
    accelerometerError);
```

As noted earlier in this chapter, the acceleration data is returned via the `accelerometerSuccess` callback function:

```
function onSuccess(acceleration) {
    alert('Acceleration X: ' + acceleration.x + '\n' +
        'Acceleration Y: ' + acceleration.y + '\n' +
        'Acceleration Z: ' + acceleration.z + '\n' +
        'Timestamp: '      + acceleration.timestamp + '\n');
```

```

};

function onError() {
    alert('oooooops!');
};

navigator.accelerometer.getCurrentAcceleration(onSuccess, onError);

```



WARNING On iOS devices, note that simply calling the `getCurrentAcceleration()` method won't work the way you think it should. Instead of giving you the current acceleration, it will report the last value reported from a PhoneGap accelerometer call. Instead, you'll need to use the `watchAcceleration()` method.

watchAcceleration

The `watchAcceleration()` method allows you to get acceleration data at a regular interval. If you set a variable to contain the method call, and pass in a frequency parameter as one of the options, you will get acceleration data on a periodic basis. Following is an example:

```

function onSuccess(acceleration) {
    alert('Acceleration X: ' + acceleration.x + '\n' +
        'Acceleration Y: ' + acceleration.y + '\n' +
        'Acceleration Z: ' + acceleration.z + '\n' +
        'Timestamp: ' + acceleration.timestamp + '\n');
};

function onError() {
    alert('onError!');
};

var options = { frequency: 1000 }; // Update every second

var watchID = navigator.accelerometer.watchAcceleration(onSuccess, onError,
    options);

```

The previous code will create a new alert every second. A more useful function might be something like the following, in which certain document object model (DOM) elements are refreshed in the HTML, thus giving you a more functional readout, as shown here:

```

function onSuccess(acceleration) {
    var myX = document.getElementById('my_x');
    var myY = document.getElementById('my_y');
    var myZ = document.getElementById('my_z');
    var myT = document.getElementById('my_timestamp');

    myX.innerHTML(acceleration.x);
    myY.innerHTML(acceleration.y);

```

```

    myZ.innerHTML(acceleration.z);
    myT.innerHTML(acceleration.timestamp);

    function onError() {
        alert('oooops!');
    };

    var options = { frequency: 1000 }; // Update every second

    var watchID = navigator.accelerometer.watchAcceleration(onSuccess, onError,
        options);

```

To make this code work, you'd have to be sure to have corresponding DOM elements, like so:

```

<div id='my_x'></div>
<div id='my_y'></div>
<div id='my_z'></div>
<div id='my_timestamp'></div>

```



WARNING Here's another iOS quirk regarding the accelerometer. PhoneGap will restrict the interval to a minimum of every 40 milliseconds (ms) and a maximum of every 1,000 ms. So, if you request an update every 3,000 ms (3 seconds), PhoneGap will request an interval of 1,000 ms from the device, but invoke the success callback at your requested interval of 3,000 ms.

clearWatch

To stop watching the acceleration previously created with the `watchAcceleration()` method, use the `clearWatch()` method and reference the variable you created. Following is an example:

```

navigator.accelerometer.clearWatch(watchID);

```

Normally, you'd fire this event as a response to a button click, like this:

```

<button onclick="stopWatch();">Stop Watching</button>

```

You could, of course, assign this event to some kind of interval — for example, collect data for 30 seconds and then stop.

ACCELEROMETER OPTION

The only accelerometer option that you can access is `frequency`, which you learned about earlier in this chapter in the section, "watchAcceleration."

To set the `frequency` to 5 seconds, use the following:

```
var options = { frequency: 5000 }; // Update every 5 seconds

var watchID = navigator.accelerometer.watchAcceleration(onSuccess, onError,
  options);
```

To set the frequency to half a second, use the following:

```
var options = { frequency: 500 }; // Update every .5 seconds

var watchID = navigator.accelerometer.watchAcceleration(onSuccess, onError,
  options);
```

TRY IT OUT Building a Simple Application

Enter the following code to create a complete application that works with the accelerometer. As soon as the application loads, move your device around to get updates on its X, Y, and Z acceleration.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Acceleration Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      var watchID = null;

      // Wait for PhoneGap to load
      document.addEventListener("deviceready", onDeviceReady, false);

      // PhoneGap is ready, start watching
      function onDeviceReady() {
        startWatch();
      }

      // Start watching the acceleration
      function startWatch() {
        // Update acceleration every 3 seconds
        var options = { frequency: 3000 };

        watchID = navigator.accelerometer.watchAcceleration(onSuccess, onError,
          options);
      }

      // Stop watching the acceleration
      function stopWatch() {
        if (watchID) {
          navigator.accelerometer.clearWatch(watchID);
          watchID = null;
        }
      }

      // onSuccess: Get a snapshot of the current acceleration
      function onSuccess(acceleration) {
```

```
var element = document.getElementById('accelerometer');
element.innerHTML = 'Acceleration X: ' + acceleration.x + '<br />' +
    'Acceleration Y: ' + acceleration.y + '<br />' +
    'Acceleration Z: ' + acceleration.z + '<br />' +
    'Timestamp: ' + acceleration.timestamp + '<br />';
}

// onError: Failed to get the acceleration
//
function onError() {
    alert('ooooops!');
}

</script>
</head>
<body>
    <div id="accelerometer">Waiting for accelerometer...</div>
    <button onclick="stopWatch();">Stop Watching</button>
</body>
</html>
```

Code file [chapter6.html] available for download at Wrox.com.

How It Works

As soon as the application loads, the `startWatch()` function is fired. This function uses the `watchAcceleration()` method to start watching the accelerometer and reporting X, Y, Z, and timestamp information back to the HTML display.

That's everything you need to know now about the accelerometer. In the next few chapters, you'll learn about the compass and geolocation services.

SUMMARY

In this chapter, you learned how to use the accelerometer. You now know how to detect movement on the device in X, Y, and Z directions. In chapters 7 and 8 you'll learn how to use the compass and geolocation tools.

In Chapter 7, you learn how to use the compass.

EXERCISES

1. Write an application that beeps if the device is held at a certain angle. (Feel free to establish what those X, Y, and Z parameters should be.)
-

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
<code>accelerometer</code> <code>.getCurrentAcceleration</code>	To detect acceleration, use the <code>getCurrentAcceleration</code> method.
<code>accelerometer</code> <code>.watchAcceleration</code>	To monitor ongoing acceleration, use the <code>watchAcceleration</code> method.
<code>accelerometer.clearWatch</code>	To stop monitoring ongoing acceleration, use the <code>clearWatch</code> method.
<code>Acceleration</code> object	To capture read-only data about the current state of acceleration, use the data returned in the <code>Acceleration</code> object.

7

Compass

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Getting to know the compass
- Using Compass methods
- Using the Compass options

In Chapter 6, you learned how to use the accelerometer. In this chapter, you learn how to use a device's built-in compass.

GETTING TO KNOW THE COMPASS

In this section, you learn what the compass is, and how you can use it.

What Is the Compass?

Simply put, the compass built into your device obtains the direction in which the device is pointing. Ideally, if you're running the PhoneGap `Compass` API, whichever direction you turn the device, you'll get back a heading (a number between 0 and 359.99) that corresponds to the direction in which you're pointing.

Specifically, your device has a solid-state or digital compass. It uses two or three (depending on the model of your device) magnetic field sensors to supply data back to your device. The device uses this data to provide a correct heading with some rapid trigonometric calculations.

If your device is any of the following, it also contains a *magnetometer*:

- HTC HD2
- HTC Dream

- HTC Desire HD
- HTC Evo
- HTC Wildfire
- Apple
- Sony Ericsson Xperia X8
- Sony Ericsson Xperia X10
- iPhone 4
- Motorola Droid[4]
- Motorola Quench
- Motorola Atrix 4g
- Nokia N97
- Nokia E72
- Nokia N8
- Blackberry Torch
- Samsung Galaxy S
- Samsung Nexus One
- Samsung Nexus S

A magnetometer can measure the strength and/or direction of a magnetic field, produced either by a machine or in nature. This chapter does not go into detail about magnetometers.

Differences Among the Different Devices

Just about all modern smartphones come with a built-in compass. All of them can provide you with a heading. Just take out your iPhone, Android, BlackBerry, or webOS phone and fire up the native compass application. While the application is running, simply turn in different directions to display a heading.

From a look-and-feel perspective, the various native compasses look slightly different, but they all provide pretty much the same amount of information.

For example, the compass in the iPhone 3GS (shown in Figure 7-1) offers a visual compass that tells you where north is, plus a readout in degrees and cardinal points (for example, NW) of your heading. At the bottom, it provides a GPS readout of your latitude and longitude.

The compass in an Android device (shown in Figure 7-2) also offers a compass display with a heading, and lets you bookmark your current location.



FIGURE 7-1: The compass in the iPhone 3GS

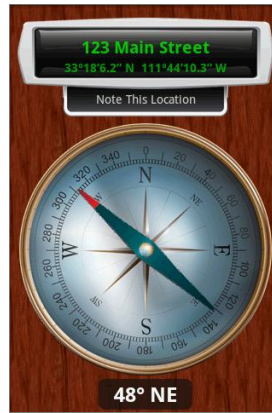


FIGURE 7-2: The compass in an Android device

The compass in the BlackBerry Torch (shown in Figure 7-3) offers the same visual compass as the iPhone and Android, but includes a direction of travel, presumably an alternative to providing a degree heading.

Finally, the compass in a webOS phone (shown in Figure 7-4) offers just a compass display with a heading, and a button that links back to a map display.



FIGURE 7-3: The compass in the BlackBerry Torch

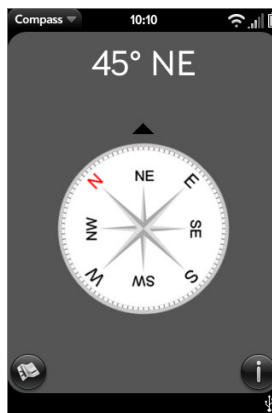


FIGURE 7-4: The compass in a webOS phone



NOTE As of PhoneGap Version 0.9.6, the `Compass` API supports only iOS and Android devices.

Using the Compass

Why use the compass at all? Why would you want to create an application that contains a compass, if your device already includes a native compass application? What's the point? Well, for one thing, if you create an application that uses the `Compass` API, you can create more specific results than just a general heading.

Here are just three creative ways to add a compass to any application:

- Create an application that detects what direction you're pointing, and then shows you what constellations are visible in the night sky, either with visual positioning or in a list.
- Create an app that tells you how far away various popular bars, hotels, and landmarks in your city are from where you're currently standing.
- Create an application that tells you in which direction important religious or cultural landmarks are located. For example, if you were Muslim, knowing the current direction for Mecca would be very important for prayers.

Showcase of Compass Applications


If you own an iPhone, you should check out "Spyglass AR," as shown in Figure 7-5. According to the software developer, "Spyglass" is a practical and fun application that is augmented with reality navigation and a powerful toolkit for the outdoors. "Spyglass" features the following:

- A high-tech viewfinder
- A compass that conforms to military specifications
- A gyrocompass
- Maps
- A GPS tracker
- A speedometer
- An optical rangefinder
- A visual sextant
- A gyro horizon
- An inclinometer

- An angular calculator
- A five-times zoom sniper scope
- A camera

You can use “Spyglass” to tag, share, find, and track your position, multiple locations, bearings, the sun, the moon, and stars, all in real time.

App Store > Navigation > Pavel Ahafonau



Spyglass ~ AR compass, rangefinder, GPS tracker, stars, maps

Description

Spyglass is practical and fun augmented reality navigation and a powerful toolkit for the outdoors. Spyglass features a hi-tech viewfinder, milspec compass, gyrocompass, maps, GPS tracker, speedometer, optical rangefinder, visual sextant, gyro horizon,....

[...More](#)

[Pavel Ahafonau Web Site >](#) [Spyglass ~ AR compass, rangefinder, GPS tracker, stars, maps Support >](#)

What's New in Version 3.3.3

- ✓ Improved Compatibility: iPod Touch 4, iPhone 3GS, iPhone 4, all iPad and iPad 2 models
- ✓ Critical bug fixes, including a bug making the coordinate parser losing the signedness for DMS and HMS formats for geo and star coordinates in between -1.0 and 0.0 degrees

\$3.99 Buy App

This app is designed for both iPhone and iPad


Category: Navigation
Updated: Jun 28, 2011
Current Version: 3.3.3
3.3.3
Size: 2.5 MB
Languages: English, Chinese, Dutch, French, German, Italian, Japanese, Portuguese, Russian,....
Seller: Pavel Ahafonau
© 2009-2011 Pavel Ahafonau

[...More](#)

Rated 4+

Requirements: Compatible with iPhone, iPod touch, and iPad.
Requires iOS 3.1 or later

More by Pavel Ahafonau



Commander Compass ~ milsp
Commander Compass Lite

Screenshots

iPhone iPad

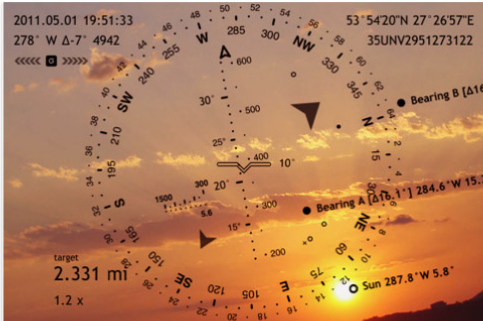
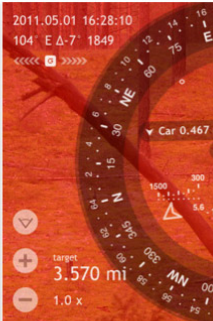



FIGURE 7-5: “Spyglass AR” application description

Another iPhone compass application is “Digital Compass” (shown in Figure 7-6), which combines a digital compass readout with a GPS. This application is only for the iPhone 3G.

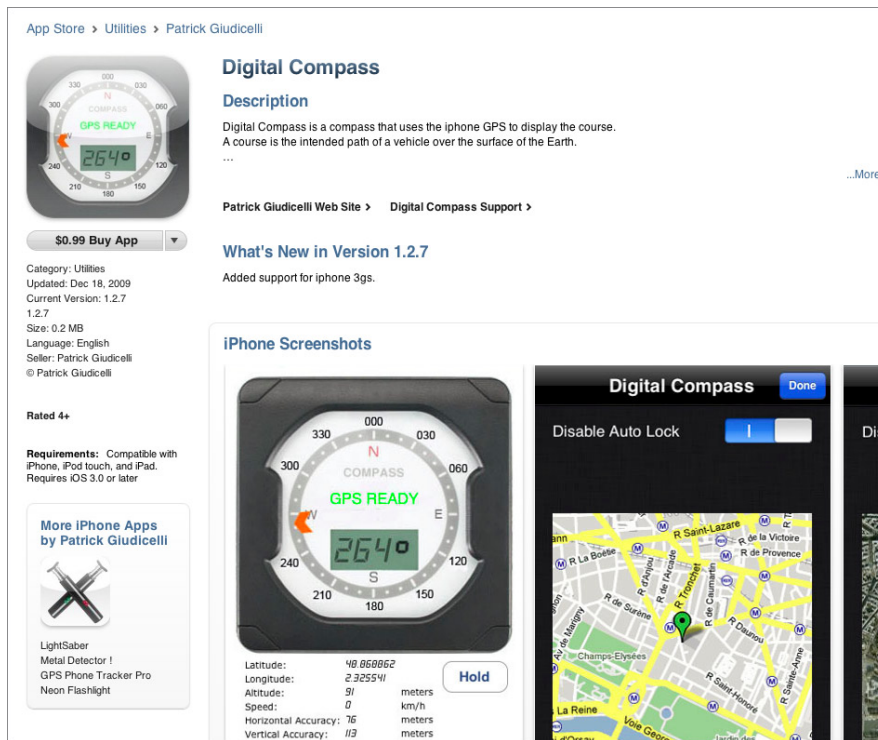


FIGURE 7-6: “Digital Compass” application description

On the Android side, the most popular compass application has already been featured in this chapter in Figure 7-2. Figure 7-7 shows how this application also enables you to write notes about locations.

USING COMPASS METHODS

In this section, you learn how to use the main methods of the compass.

getCurrentHeading

To get the current heading, use `compass.getCurrentHeading`, as shown here:

```
navigator.compass.getCurrentHeading(compassSuccess, compassError,
    [compassOptions]);
```

As shown in this code snippet, `compassOptions` is an argument that accepts a list of options. This is an entirely optional item. You don't have to include it.

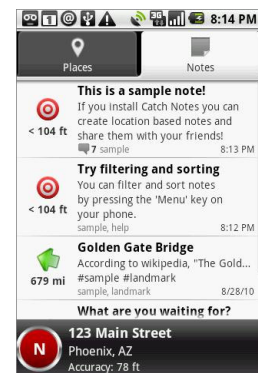


FIGURE 7-7: Notes for the Android compass application

If successful, you'll get back a heading (a degree reading between 0 and 359.99) as part of the `successCompass` callback function, like this:

```
function onSuccess(heading) {
    alert('Heading: ' + heading);
}

function onError() {
    alert('oooops!');
}

navigator.compass.getCurrentHeading(onSuccess, onError);
```

watchHeading

The `watchHeading()` method enables you to get headings at a regular interval. If you set a variable to contain the method call, and pass in a frequency parameter as one of the options, you will get heading data on a periodic basis.

Following is an example of how to use this method:

```
function onSuccess(heading) {
    alert(heading);
}

function onError() {
    alert('oooops!');
}

var options = { frequency: 2000 }; // Update every 2 seconds
//note that frequency is the only option available!

var watchID = navigator.compass.watchHeading(onSuccess, onError, options);
```

Here, you create a new alert every 2 seconds. A more useful function might be something like the following, in which certain document object model (DOM) elements are refreshed in the HTML, thus giving you a more functional readout:

```
function onSuccess(heading) {
    var element = document.getElementById('heading');
    element.innerHTML = 'Heading: ' + heading;
}

function onError() {
    alert('oooops!');
}

var options = { frequency: 2000 }; // Update every 2 seconds

var watchID = navigator.compass.watchHeading(onSuccess, onError, options);
```

To make this code work, be sure to add a corresponding DOM element, like so:

```
<div id='heading'></div>
```

clearWatch

To stop processing a `watchHeading` command, that was previously created with the `watchHeading()` method, use the `clearWatch()` method and reference the variable you created.

Following is an example:

```
navigator.compass.clearWatch(watchID);
```

Normally, you'd fire this event as a response to a button click, like this:

```
<button onclick="stopWatch();">Stop Watching</button>
```

USING A COMPASS OPTION

The only compass option that you can access is frequency, which you learned about earlier in this chapter in the section, “`watchHeading`.”

Following is an example of how to set the frequency to 5 seconds:

```
var options = { frequency: 5000 }; // Update every 5 seconds
var watchID = navigator.compass.watchHeading(onSuccess, onError, options);
```

Following is an example of how to set the frequency to half a second:

```
var options = { frequency: 500 }; // Update every .5 seconds
var watchID = navigator.compass.watchHeading(onSuccess, onError, options);
```

TRY IT OUT Building a Simple Application

Let's create a complete application that works with the compass. Enter the following code. As soon as the application loads, tap the Start Watching button, and move your device to get heading updates.



Available for
download on
Wrox.com

```
<!DOCTYPE html>
<html>
  <head>
    <title>Compass Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      var watchID = null;
      document.addEventListener("deviceready", onDeviceReady, false);

      // PhoneGap is ready
```

```

function onDeviceReady() {
    startWatch();
}

// Start watching the compass
function startWatch() {

    // Update compass every 2 seconds
    var options = { frequency: 2000 };

    watchID = navigator.compass.watchHeading(onSuccess, onError, options);
}

// Stop watching the compass
function stopWatch() {
    if (watchID) {
        navigator.compass.clearWatch(watchID);
        watchID = null;
    }
}

// onSuccess: Get the current heading
function onSuccess(heading) {
    var element = document.getElementById('heading');
    element.innerHTML = 'Heading: ' + heading;
}

// onError: Failed to get the heading
function onError() {
    alert('oooops!');
}

</script>
</head>
<body>
    <div id="heading">Waiting for heading...</div>
    <button onclick="startWatch();">Start Watching</button>
    <button onclick="stopWatch();">Stop Watching</button>
</body>
</html>

```

Code file [chapter7.html] available for download at Wrox.com.

How It Works

As soon as the user taps the Start Watching button, the `startWatch()` function is fired. This function uses the `watchHeading()` method to start watching the compass, and reports heading information back to the HTML display.

Now, let's take a look at some issues related to the user interface (UI).



FIGURE 7-8: Bare-bones UI

IMPROVING THE LOOK AND FEEL

Note that the code in the previous exercise provides only a bare-bones UI, as shown in Figure 7-8.

Let's now apply some jQTouch “goodness” to the UI. If you’ve downloaded the ZIP file from [jqTouch.com](http://jqtouch.com) and unzipped it, you should have a set of directories that match Figure 7-9.

Copy the `jqtouch` and `themes` folders into your Xcode project. The easiest way to do that is to right-click the `www` folder in Xcode and select `Open With Finder`, as shown in Figure 7-10.

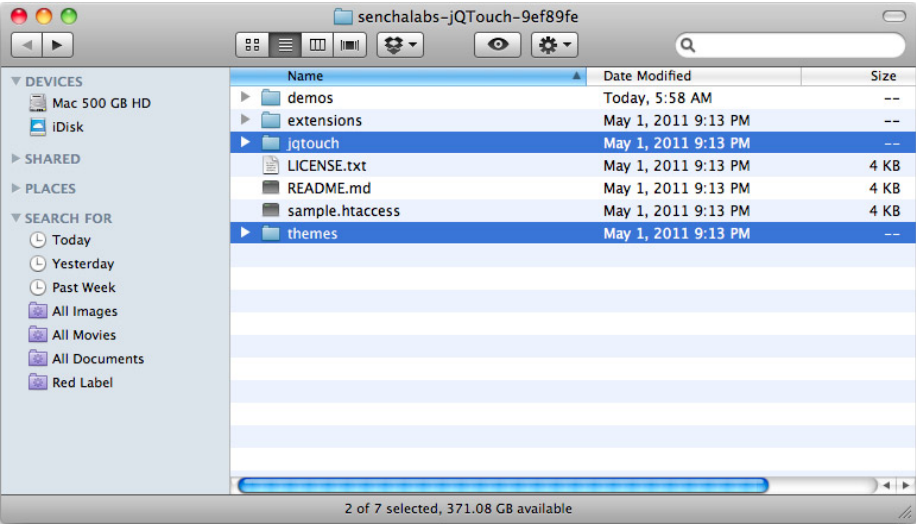


FIGURE 7-9: Files and folders after decompressing the jQTouch ZIP file

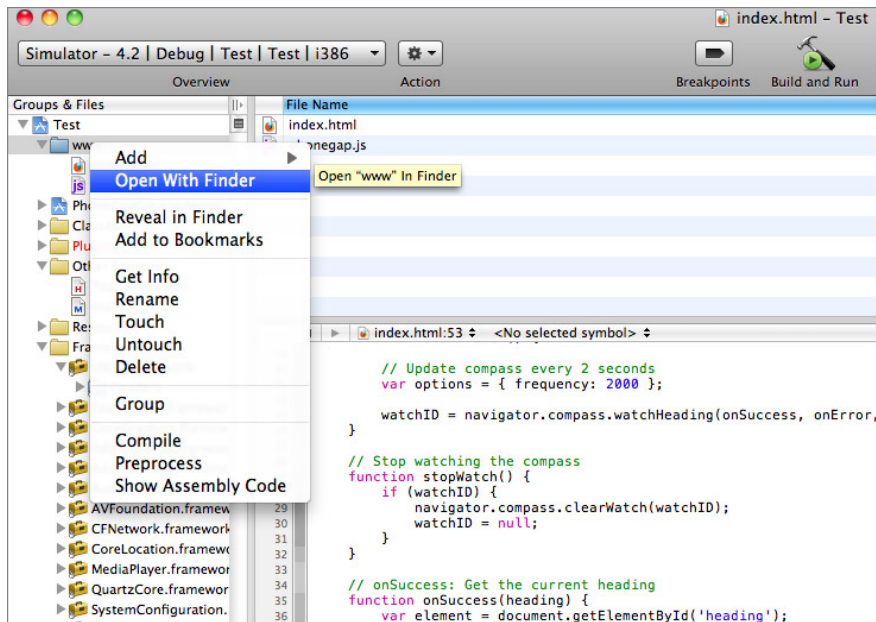


FIGURE 7-10: Selecting the Open With Finder option

When you're finished, your `www` folder should look like Figure 7-11.

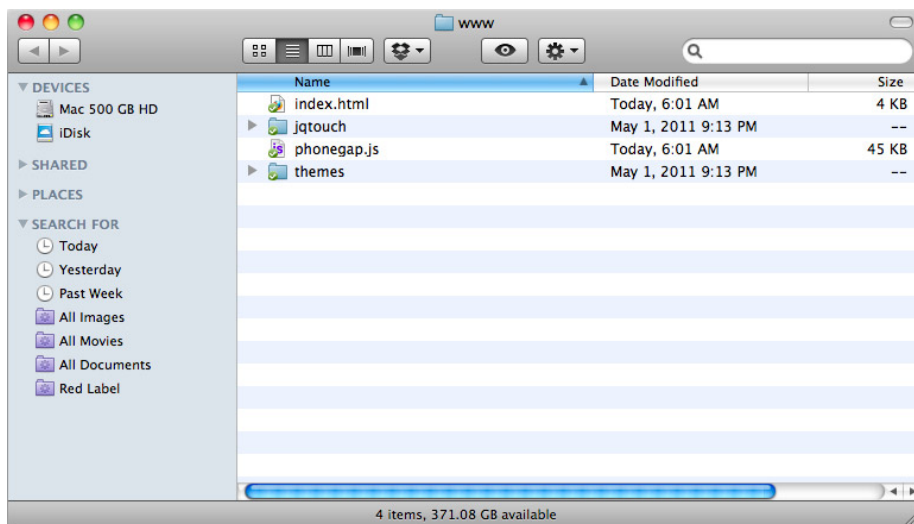


FIGURE 7-11: Contents of the new `www` folder

It's now time to add some important information to the head portion of the HTML file. Stick the following code above the first call to load `phonegap.js`:

```
<style type="text/css" media="screen">@import "jqtouch/jqtouch.css";</style>
<style type="text/css" media="screen">@import "themes/jqt/theme.css";</style>
<script src="jqtouch/jquery-1.4.2.js" type="text/javascript"
    charset="utf-8"></script>
<script src="jqtouch/jqtouch.js" type="application/x-javascript"
    charset="utf-8"></script>
<script type="text/javascript" charset="utf-8">
    var jQT = new $.jQTouch({
        icon: 'jqtouch.png',
        icon4: 'jqtouch4.png',
        addGlossToIcon: false,
        startupScreen: 'jqt_startup.png',
        statusBar: 'black',
        preloadImages: [
            'themes/jqt/img/activeButton.png',
            'themes/jqt/img/back_button.png',
            'themes/jqt/img/back_button_clicked.png',
            'themes/jqt/img/blueButton.png',
            'themes/jqt/img/button.png',
            'themes/jqt/img/button_clicked.png',
            'themes/jqt/img/grayButton.png',
            'themes/jqt/img/greenButton.png',
            'themes/jqt/img/redButton.png',
            'themes/jqt/img/whiteButton.png',
            'themes/jqt/img/loading.gif'
        ]
    });
</script>
```

Next, take out the old HTML that's inside the body tags and put the following code in there:

```
<div id="home" class="current">
    <div class="toolbar">
        <h1>Compass</h1>

    </div>
    <div id="heading">Waiting for heading...</div>
    <ul class="individual">
        <li id='startwatching' onclick="startWatch();">Start Watching</li>
        <li id='stopwatching' onclick="stopWatch();">Stop Watching</li>
    </ul>
</div>
```

Notice a DOM element has been created with an `id` of `home`. Inside that DOM element is a second `div` with a class of `toolbar`. The DOM element with an `id` of `heading` comes after that. (This is where the heading information will be placed.) After that comes an unordered list with two elements. This code uses the `individual` class to simulate having two buttons side by side. `onClick` events are assigned to each list item, and the new UI is in.

Figure 7-12 shows the results.

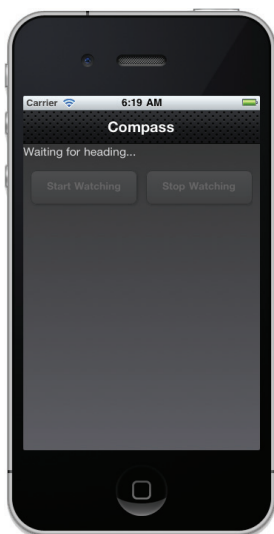


FIGURE 7-12: Improved UI

SUMMARY

In this chapter, you learned how to use the compass. You now know how to detect the device's heading and how to monitor (or stop monitoring) that heading.

EXERCISES

1. Display a special message if the device is oriented north. Otherwise, don't display a special message.



NOTE *Answers to the Exercises can be found in Appendix A.*

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
<code>compass.getCurrentHeading</code>	Use <code>compass.getCurrentHeading</code> to detect a heading.
<code>compass.watchHeading</code>	Use <code>compass.watchHeading</code> to monitor a device heading.
<code>compass.clearWatch</code>	Use <code>compass.clearWatch</code> to stop monitoring a device heading.

8

Geolocation

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Understanding geolocation
- Using the `Position`, `PositionError`, and `Coordinates` objects
- Using Geolocation methods
- Using Geolocation options and arguments

In Chapter 7, you learned how to use the compass. In this chapter, you learn how to use the device's built-in geolocation services.

GETTING TO KNOW GEOLOCATION

In this section, you learn what geolocation is, and how you can use it.

What Is Geolocation?

Simply put, *geolocation* involves the identification of a real-world geographic location of a device — in this case, the smartphone you have with you. The most common way to determine a device's geolocation is to use an internal GPS feature to determine latitude and longitude.

However, some devices might use other services to infer a location — for example, an IP address, Radio Frequency Identification (RFID), WiFi Media Access Control (MAC) addresses, or cell tower triangulation. Because of the wide divergence of methods used

to obtain geolocation coordinates, no one can guarantee that the PhoneGap API will return a device's actual location.

Using Geolocation

The most obvious reason to use geolocation is the recent trend to indicate where something happened. For example, geolocation data is being added to photos (so you can better organize them on Flickr photo galleries by geographic location, for example), and social location services like Gowalla and Foursquare allow you to “check in” to a location. Some existing applications also combine geolocation data (particularly readings of latitude and longitude) to pinpoint your location on a map, or add important information to a compass display.

However, geolocation applications mostly pinpoint real-world locations (such as bars, restaurants, and other landmarks) and associate them to the device in the user's hand.

From your point of view as a developer, the biggest consideration you will have is how accurate geolocation data will be on any given device. Depending on the type of smartphone being used, the resolution (in other words, accuracy) will vary. Bad weather might cause interference with the GPS satellite, or the user might be indoors or underground and not getting good reception. If the user is in a place where many stores, restaurants, and other landmarks are in close proximity to one another, the user might not get very accurate readings of where he or she is. (Anyone who has used a social location service to check in to a bar has experienced this problem.)

Following are three creative ways you might add geolocation data to any application:

- Add not only latitude and longitude, but also altitudes and headings to photos taken with the smartphone.
- Allow users to “check in” to any latitude/longitude coordinate so that they can keep track of their outdoor hikes (similar to a breadcrumbing feature).
- Tell a user where people are tweeting from in the local vicinity. Many Twitter users post geolocation data with their tweets, and you could use the Twitter search API to pull down tweets that are within a few miles of a smartphone user.

Showcase of Geolocation Applications

Quite possibly the most famous geolocation apps out there are “foursquare” (shown in Figure 8-1) and “Gowalla” (shown in Figure 8-2). Both apps are free and available on lots of different device platforms. You can sign up and start using them right away. The next time you're in a restaurant or bar, you can use these apps to check in and use Facebook or Twitter to let your friends know where you are.

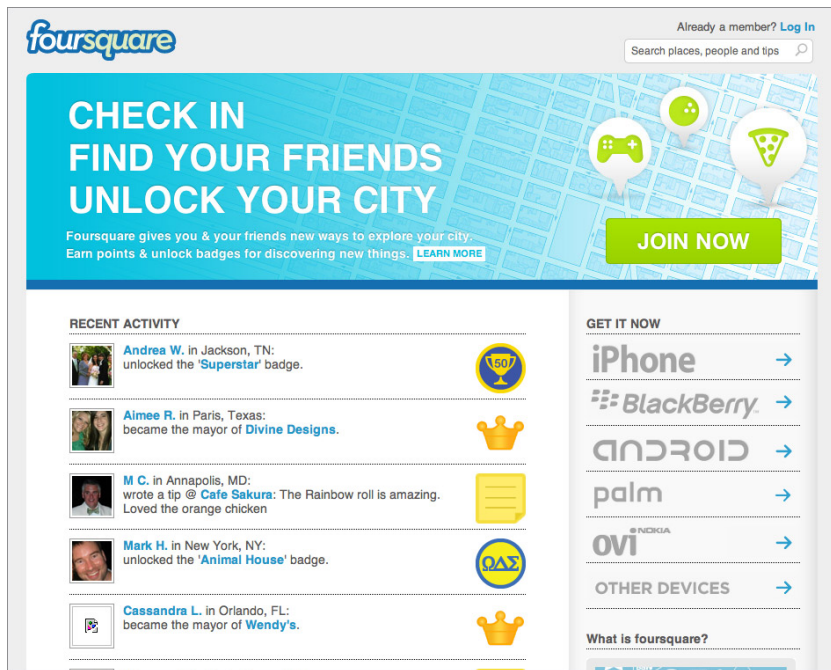


FIGURE 8-1: “foursquare” application

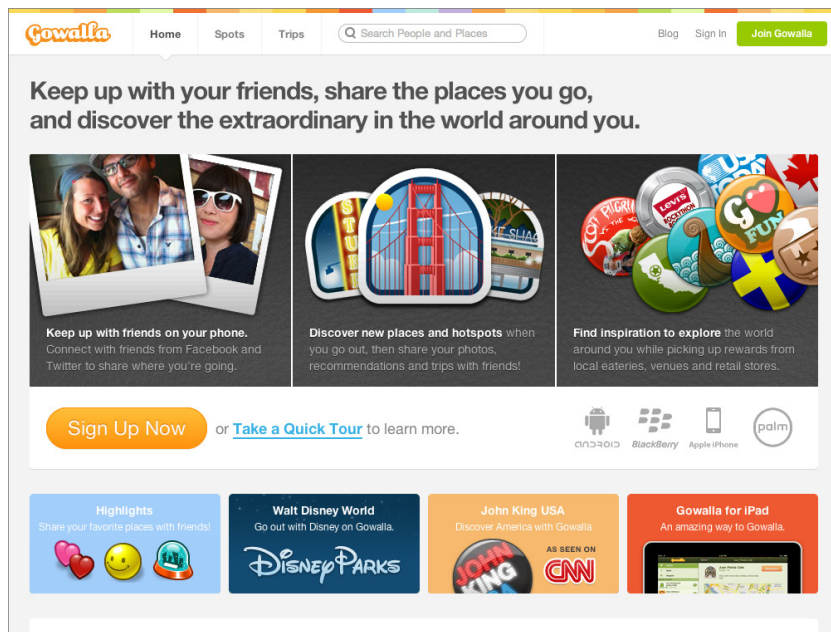


FIGURE 8-2: “Gowalla” application

“Gowalla” and “foursquare” both offer a “social gaming” environment — players not only check in, but get access to deals and offers created by retail locations. On Gowalla, you can take trips and pick up items that other players have left for you.

Of course, if you’re a Facebook or Google+ (a social network from Google) user, you can join in the geolocation games as well. As shown in Figure 8-3, the Facebook “Places” service allows you to check in to a location and then tell your Facebook friends where you are. You can also use the service to figure out where your friends are at the moment, and can even use a “Here Now” feature to see if any of your friends is in the same place as you.

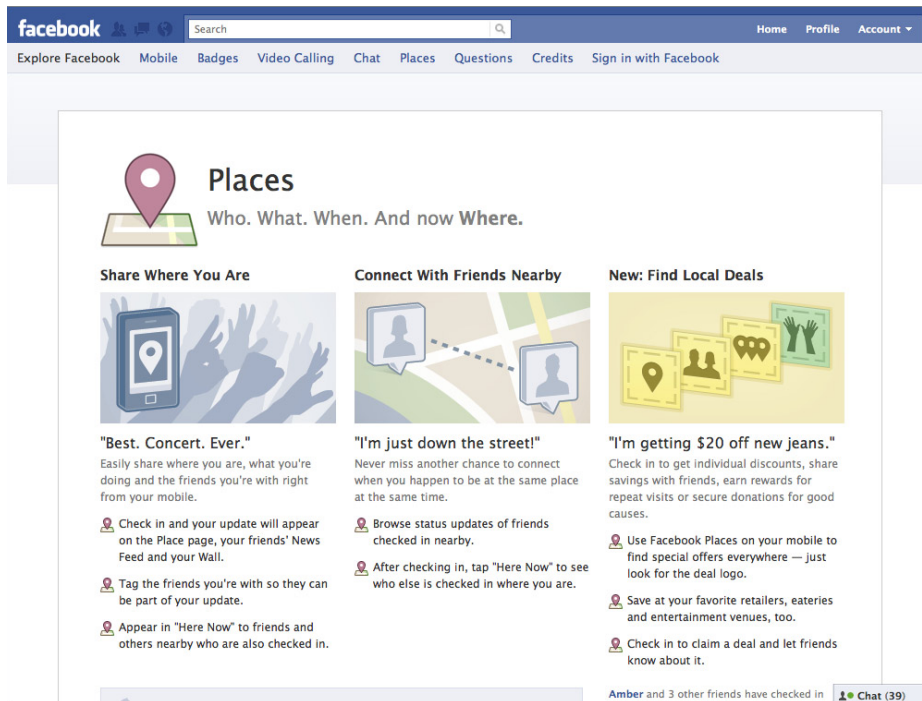


FIGURE 8-3: Using the Facebook “Places” service

On Google+, you can easily check in to a place by tapping the checkbox icon in the upper-right corner of the application page, as shown in Figure 8-4. You can then select a location from the list of nearby venues, add a note and a photograph, and post it to your stream.

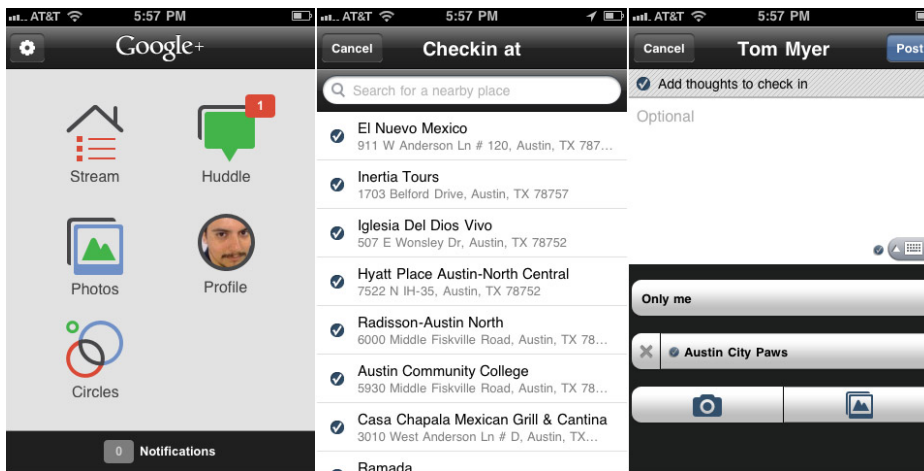


FIGURE 8-4: Using Google+ to check in

As you can see, lots of great applications already take advantage of geolocation data.

THE POSITION, POSITIONERROR, AND COORDINATES OBJECTS

The PhoneGap Geolocation API uses three read-only objects that contain geolocation data:

- Position
- PositionError
- Coordinates

Each of these objects is created and populated when you use different methods, and the data is returned as part of a specific callback function.

Position Object

The `Position` object contains coordinates that are created by the geolocation API. Following are the two properties for this object:

- `coords` — This is a set of geographic coordinates (latitude, longitude, altitude, and so on).
- `timestamp` — This creates a timestamp (in milliseconds).

For example, calling `geolocation.getCurrentPosition()` as shown here results in latitude, longitude, altitude, accuracy, altitude accuracy, heading, speed, and timestamp being returned in the `onSuccess` callback function:

```
var onSuccess = function(position) {
    alert('Latitude: ' + position.coords.latitude + '\n' +
```

```
        'Longitude: '      + position.coords.longitude      + '\n' +
        'Altitude: '      + position.coords.altitude        + '\n' +
        'Accuracy: '       + position.coords.accuracy         + '\n' +
        'Altitude Accuracy: ' + position.coords.altitudeAccuracy + '\n' +
        'Heading: '        + position.coords.heading         + '\n' +
        'Speed: '          + position.coords.speed           + '\n' +
        'Timestamp: '      + new Date(position.timestamp)    + '\n');
    }
    function onError(error) {
        alert('code: '      + error.code      + '\n' +
            'message: ' + error.message + '\n');
    }

    navigator.geolocation.getCurrentPosition(onSuccess, onError);
```

PositionError Object

As you can see from the previous example, you also have access to an `onError` callback function. This function returns both an error code and an error message in case of a problem, encapsulated as a read-only `PositionError` object.

The error code will be one of the following predefined error codes (they're constants):

- `PositionError.PERMISSION_DENIED`
- `PositionError.POSITION_UNAVAILABLE`
- `PositionError.TIMEOUT`

The error message describes the details of the error encountered.

Coordinates Object

The `Coordinates` object contains a set of read-only properties. You've already seen them previously as part of the discussion on the `Position` object. In fact, the `Coordinates` data is attached to the `Position` object, which is then returned to the user via the `onSuccess` callback function.

Following are the properties of the `Coordinates` object:

- `latitude` — This is the latitude expressed in decimal degrees.
- `longitude` — This is the longitude expressed in decimal degrees.
- `altitude` — This is the height (in meters) above sea level.
- `accuracy` — This is the accuracy of latitude/longitude reading (in meters).
- `altitudeAccuracy` — This is the accuracy of altitude coordinate (in meters).
- `heading` — This is the direction of travel (specified in degrees relative to true north).
- `speed` — This is the current ground speed (specified in meters per second).

USING GEOLOCATION METHODS

In this section, you learn how to use the main methods of the Geolocation API.

getCurrentPosition

To get the current position, use `geolocation.getCurrentPosition`, as shown here:

```
navigator.geolocation.getCurrentPosition(geolocationSuccess, [geolocationError],
    [geolocationOptions]);
```

The callback function that contains the current position is `geolocationSuccess`. The optional callback that is called in case of an error is `geolocationError`. An optional set of data you can pass in to the Geolocation API is included in `geolocationOptions`.

The `getCurrentPosition()` method is an asynchronous function that returns current position data. As mentioned earlier in this chapter, you use the `onSuccess` callback function to actually retrieve the position data, as shown here:

```
var onSuccess = function(position) {
    alert('Latitude: '          + position.coords.latitude          + '\n' +
          'Longitude: '         + position.coords.longitude         + '\n' +
          'Altitude: '          + position.coords.altitude          + '\n' +
          'Accuracy: '          + position.coords.accuracy          + '\n' +
          'Altitude Accuracy: ' + position.coords.altitudeAccuracy + '\n' +
          'Heading: '           + position.coords.heading           + '\n' +
          'Speed: '             + position.coords.speed             + '\n' +
          'Timestamp: '         + new Date(position.timestamp)       + '\n');
};

// onError Callback receives a PositionError object
//
function onError(error) {
    alert('code: '      + error.code      + '\n' +
          'message: '   + error.message   + '\n');
}

navigator.geolocation.getCurrentPosition(onSuccess, onError);
```

watchPosition

The `watchPosition()` method enables you to get position data at a regular interval. If you set a variable to contain the method call, and pass in a frequency parameter as one of the options, you will get data on a periodic basis.

Following is an example:

```
function onSuccess(position) {
    alert('Latitude: ' + position.coords.latitude + ' ' +
          'Longitude: ' + position.coords.longitude);
}
```

```
}

function onError(error) {
    alert('code: ' + error.code + '\n' +
        'message: ' + error.message + '\n');
}

var watchID = navigator.geolocation.watchPosition(onSuccess, onError,
    { frequency: 2000 });
```

This code creates a new alert every 2 seconds. A more useful function might be something like the following, in which certain document object model (DOM) elements are refreshed in the HTML, thus giving you a more functional readout:

```
function onSuccess(position) {
    var element = document.getElementById('geolocation');
    element.innerHTML = 'Latitude: ' + position.coords.latitude + '<br />' +
        'Longitude: ' + position.coords.longitude + '<br />' +
        '<hr />' + element.innerHTML;
}

// onError Callback receives a PositionError object
//
function onError(error) {
    alert('code: ' + error.code + '\n' +
        'message: ' + error.message + '\n');
}

//
var watchID = navigator.geolocation.watchPosition(onSuccess, onError,
    { frequency: 2000 });
```

To make this code work, be sure to add a corresponding DOM element, like so:

```
<div id='geolocation'></div>
```

clearWatch

To stop processing the `watchHeading` command that was previously created with the `watchPosition()` method, use the `clearwatch()` method and reference the variable you created, as shown in the following example:

```
navigator.geolocation.clearWatch(watchID);
```

Normally, you'd fire this event as a response to a button click, like this:

```
<button onclick="stopWatch();">Stop Watching</button>
```

You could, of course, assign this event to work for a time interval — for example, collecting data for 30 seconds and then stopping.

USING GEOLOCATION OPTIONS

You have access to four optional parameters to customize the retrieval of geolocation data:

- `frequency` — This specifies how often (in milliseconds) to retrieve geolocation data.
- `enableHighAccuracy` — This is used to receive the best possible (most accurate) results.
- `timeout` — This specifies the maximum length of time (in milliseconds) before invoking the `onSuccess` callback function.
- `maximumAge` — This indicates to accept a cached position whose age is no greater than the specified time (in milliseconds).

So, for example, the following options would set a maximum age of 5 seconds, a timeout of 5 seconds, and enable high accuracy:

```
var options = { maximumAge: 5000, timeout: 5000, enableHighAccuracy: true };
watchID = navigator.geolocation.watchPosition(onSuccess, onError, options);
```

TRY IT OUT Building a Simple Application

Let's create a complete application that works with the Geolocation API. Enter the following code. As soon as the application loads, you'll start getting updates every 3 seconds.



```
<!DOCTYPE html>
<html>
  <head>
    <title>Device Properties Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

document.addEventListener("deviceready", onDeviceReady, false);

var watchID = null;

function onDeviceReady() {
  // Update every 3 seconds
  var options = { frequency: 3000 };
  watchID = navigator.geolocation.watchPosition(onSuccess, onError, options);
}

// onSuccess Geolocation
//
function onSuccess(position) {
  var element = document.getElementById('geolocation');
  element.innerHTML = 'Latitude: ' + position.coords.latitude + '<br />' +
    'Longitude: ' + position.coords.longitude + '<br />' +
    '<hr />' + element.innerHTML;
}

// onError Callback receives a PositionError object
```

```
//
function onError(error) {
    alert('code: ' + error.code + '\n' +
        'message: ' + error.message + '\n');
}

</script>
</head>
<body>
    <p id="geolocation">Watching geolocation...</p>
</body>
</html>
```

Code file [chapter8.html] available for download at Wrox.com.

How It Works

As soon as the application loads, the `watchPosition()` method is fired. Every 3 seconds the device will get a new update on longitude and latitude.

Next, let's take a look at how to improve the user interface (UI).

IMPROVING THE LOOK AND FEEL

If you look at the UI for the geolocation application you created in the previous “Try It Out” exercise, you’ll obviously notice that it’s pretty bland, as shown in Figure 8-5. In fact, all it does is keep adding geolocation information to the screen until you turn it off.



FIGURE 8-5: Rather bland geolocation UI

To begin improving the UI, first ensure that your project has the necessary jQTouch folders in it, as described in Chapter 7. Stick the following code above the first call to `loadPhonegap.js`:

```
<style type="text/css" media="screen">@import "jqtouch/jqtouch.css";</style>
<style type="text/css" media="screen">@import "themes/jqt/theme.css";</style>
<script src="jqtouch/jquery-1.4.2.js" type="text/javascript" charset="utf-8"></script>
<script src="jqtouch/jqtouch.js" type="application/x-javascript" charset="utf-8"></script>
<script type="text/javascript" charset="utf-8">
  var jQT = new $.jQTouch({
    icon: 'jqtouch.png',
    icon4: 'jqtouch4.png',
    addGlossToIcon: false,
    startupScreen: 'jqt_startup.png',
    statusBar: 'black',
    preloadImages: [
      'themes/jqt/img/activeButton.png',
      'themes/jqt/img/back_button.png',
      'themes/jqt/img/back_button_clicked.png',
      'themes/jqt/img/blueButton.png',
      'themes/jqt/img/button.png',
      'themes/jqt/img/button_clicked.png',
      'themes/jqt/img/grayButton.png',
      'themes/jqt/img/greenButton.png',
      'themes/jqt/img/redButton.png',
      'themes/jqt/img/whiteButton.png',
      'themes/jqt/img/loading.gif'
    ]
  });
</script>
```

Next, take out the old HTML that's inside the `body` tags and put this in there:

```
<div id="home" class="current">
  <div class="toolbar">
    <h1>Geolocation</h1>

  </div>
  <p id="geolocation">Watching geolocation...</p>
</div>
```

Figure 8-6 shows the result.

This is still not a great UI, but it's a lot better. How can you improve it? Well, for one thing, you can simplify the return data from the `onSuccess` function. Instead of spewing out all that information, perhaps this would do:

```
function onSuccess(position) {
  var element = document.getElementById('geolocation');
  element.innerHTML = 'Position: ' + position.coords.latitude + ', ' +
```

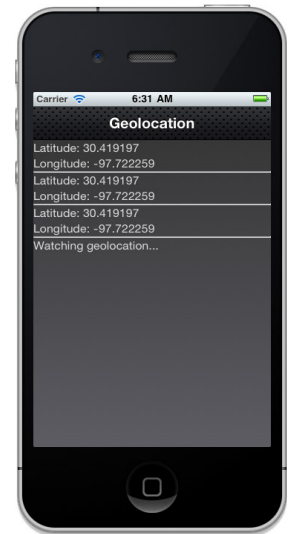


FIGURE 8-6: Improved geolocation UI

```

        + position.coords.longitude + '<br />' +
        '<br/>' + element.innerHTML;
    }

```

When you look at this new code in the Simulator, you'll see something similar to Figure 8-7.

The information is more compact, certainly, and you could improve it even more with bolding and other tricks. But at least now it's a lot easier on the eyes.

SUMMARY

In this chapter, you learned how to use the geolocation features of the PhoneGap API. You now know how to retrieve latitude, longitude, maybe some altitude, and other data points of your current position.

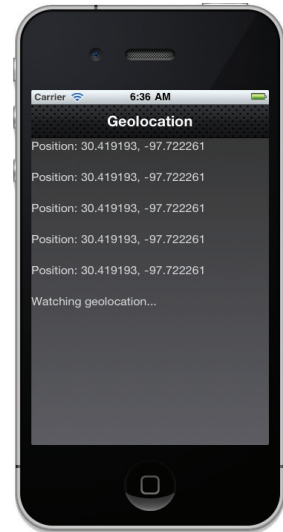


FIGURE 8-7: Geolocation UI with more compact information

EXERCISES

1. Add a Stop Watching button to the previous “Try It Out” example.
2. Write an application that displays a special message if the speed drops below a certain threshold. (Feel free to set any threshold you like.)



NOTE Answers to the Exercises can be found in Appendix A.

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
<code>geolocation.getCurrentPosition</code>	Use <code>geolocation.getCurrentPosition</code> to detect geolocation coordinates.
<code>geolocation.watchPosition</code>	Use <code>geolocation.watchPosition</code> to monitor geolocation coordinates.
<code>geolocation.clearWatch</code>	Use <code>geolocation.clearWatch</code> to stop monitoring geolocation coordinates.



Media

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Working with media files
- Using the Media object
- Using Media methods
- Handling errors

In the past few chapters, you have learned how to use the accelerometer, the compass, and geolocation services. In this chapter, you learn how to use the device's built-in media services.

LEARNING ABOUT MEDIA FILES

In this section, you learn what the `Media` API is and what services are available to you to use media files.

What Are Media Files?

The `Media` API lets you record and play back audio on a device. You can typically play audio files loaded locally on the device, or play audio files retrieved from the Internet.



NOTE An important thing to remember is that the current PhoneGap implementation doesn't adhere to World Wide Web Consortium (W3C) specifications for media capture, and any future implementations may deprecate the current APIs. Furthermore, the `Media` API is compatible only with Android and iOS devices.

Using Media Files

Being able to play music or other audio files directly from your smartphone is quite convenient. For example, if you're on a long trip and don't have a book handy, no problem, you can listen to an audio book.

Furthermore, the capability to record audio has been pretty convenient, too. If you're waiting for a meeting to start, and need to record a few reminders for later, you could use your device as a memo recorder. In the past, the author has used his iOS device to record up to an hour's worth of audio for a podcast. The audio quality was just fine, and it was easy to incorporate the resulting MP3 file into audio-editing tools.

Following are just three creative ways to add media playing/recording to any application:

- Create a simple audio recorder, which is perfect for simple reminders.
- Incorporate audio recording/playback features to a note-taking application like Evernote.
- Create a simple audio player that plays files out on the Web.

Showcase of Media Applications

An abundance of voice recorders and music-playing applications exist for both iOS and Android devices — almost too many to mention here.

On the iOS front, standard applications like “Voice Memos” and “iPod” (shown in Figure 9-1) allow you to record voice memos and play audio files, respectively.

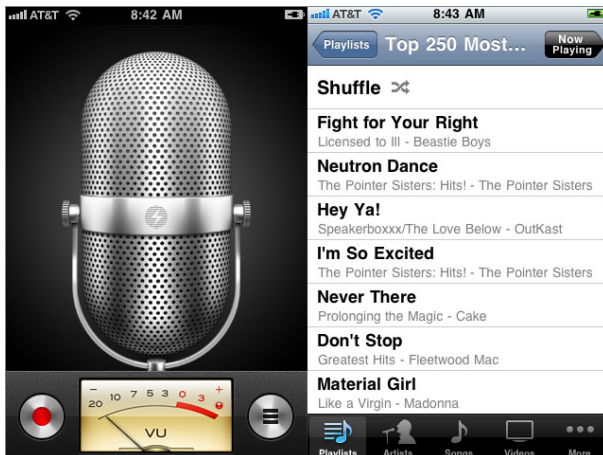


FIGURE 9-1: “iPod” application

Of course, just because Apple provides these apps for free doesn't mean that others don't exist. Figure 9-2 shows just a partial list of voice recorders available on the App Store. All of these provide the same basic functions — record, pause, play, and stop recording. Some offer the capability to share your recordings with others via e-mail, file sharing, or social media services.

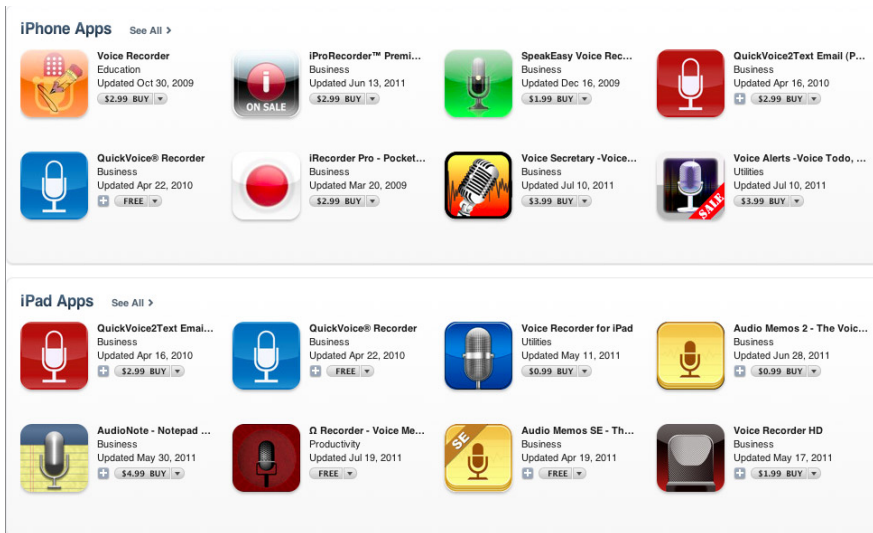


FIGURE 9-2: Apple App Store offerings for voice recorders

If you were to search for music apps, you'd also get a fairly long list of apps, as shown in Figure 9-3.

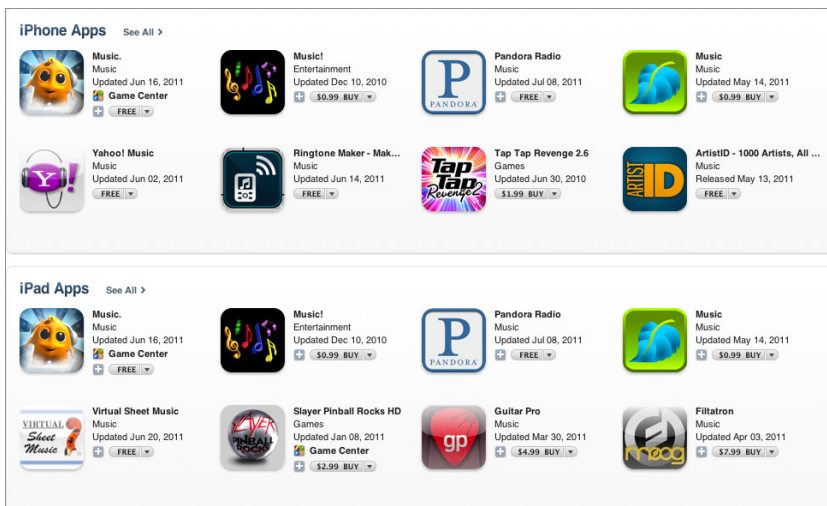


FIGURE 9-3: Apple App Store offerings for music applications

One popular choice is “Pandora” (shown in Figure 9-4), which is tied to the music service on the Web. The “Pandora” service offers an interesting way to sample music. Let’s say you want to listen to some music from the rock group Creedence Clearwater Revival (CCR). You create a “station” for that band and then “Pandora” will find music not just selections recorded by CCR, but also songs from other bands that match the “genetic makeup” of CCR tracks. You can then share your new “radio station” with others.

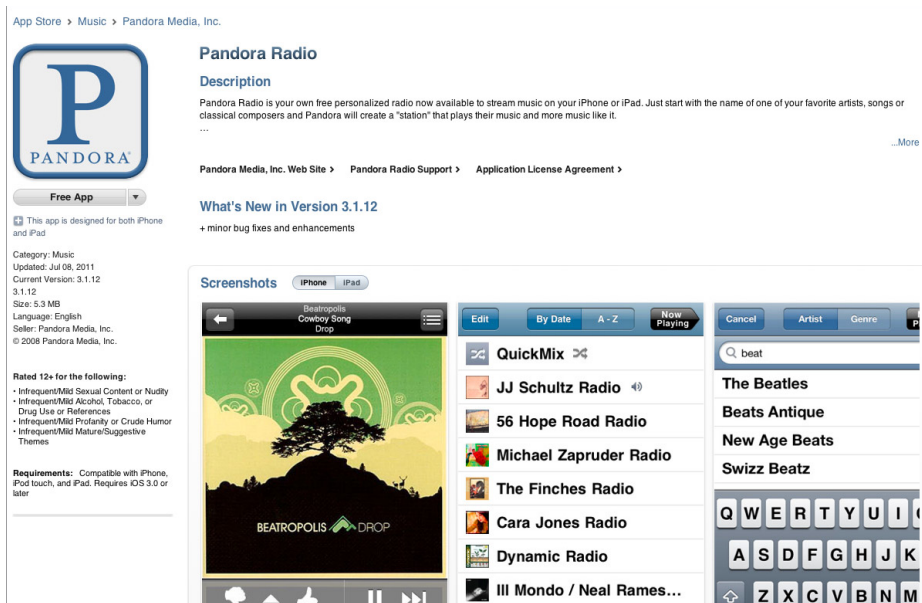


FIGURE 9-4: “Pandora” application

Most (if not all) of the more popular music applications are also available on Android. Note that, as shown in Figure 9-5, the “Pandora” interface is pretty phenomenal.

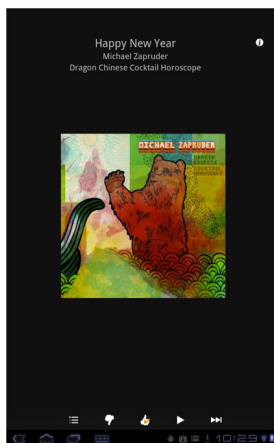


FIGURE 9-5: “Pandora” interface

Possibly the most popular voice recorder on an Android device is “Voice Recorder” by Mamoru Tokashiki (shown in Figure 9-6). You can create all kinds of voice recordings, give them custom names, and even set an automatic start and stop time for recordings.

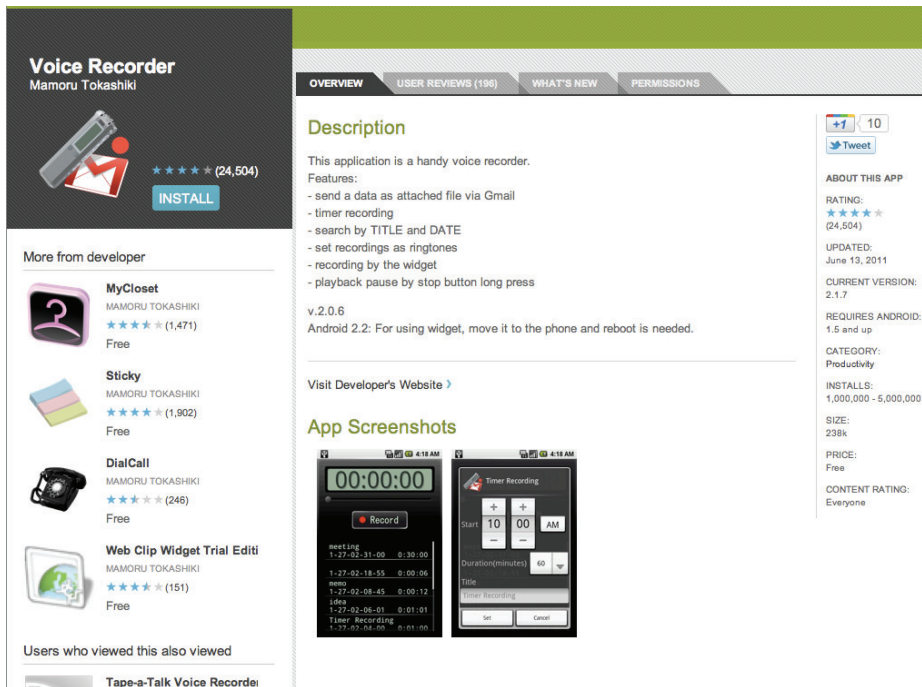


FIGURE 9-6: “Voice Recorder” application

THE MEDIA OBJECT

The PhoneGap `Media` object has three parameters, two of which are optional:

- `src` — This is a URI (that is, a URL to a file on the Internet) containing the audio content.
- `mediaSuccess` — This is an optional callback function invoked after a `Media` object has completed the current play, record, or stop action.
- `mediaError` — This is an optional callback function invoked if there was an error.

The best way to learn about the `Media` API is to start learning about its methods.

USING MEDIA METHODS

In this section, you learn how to use the main methods of the `Media` API. The methods allow you to play, record, and pause, among other things.



WARNING If you're working with Version 0.9.4 or earlier on PhoneGap, you must ensure that all your callback functions are global in scope.



NOTE In the following discussions, example URLs and filenames for media objects have been used. However, you should use your own appropriate URLs and filenames.

getCurrentPosition

To get the current position within an audio file, use `media.getCurrentPosition`, as shown here:

```
media.getCurrentPosition(mediaSuccess, [mediaError]);
```

The `mediaSuccess` function is a callback that is called with the current position, and the optional `mediaError` is a callback that's called in case of an error.

The `getCurrentPosition()` method is an asynchronous function that returns the current position of the underlying audio file of a `Media` object, and is supported only on Android. Following is an example:

```
var audio_file = new Media(src, onSuccess, onError);

// Update media position every second
var mediaTimer = setInterval(function() {
    // get media position
    audio_file.getCurrentPosition(
        // success callback
        function(position) {
            if (position > -1) {
                console.log((position/1000) + " sec");
            }
        },
        // error callback
        function(e) {
            console.log("Error getting pos=" + e);
        }
    );
}, 1000);
```

getDuration

`getDuration()` is a synchronous function that returns the duration (in milliseconds) of an audio file. To convert to seconds, you will need to divide by 1,000. If the duration is unknown, a value of -1 is returned. Following is an example:

```
var audio_file = new Media(src, onSuccess, onError);

// Get duration
var counter = 0;
var myTimer = setInterval(function() {
    counter = counter + 100;
    if (counter > 2000) {
        clearInterval(myTimer);
    }
}, 100);
```

```

        var myDur = audio_file.getDuration();
        if (mydur > 0) {
            clearInterval(myTimer);
            document.getElementById('audio_duration').innerHTML =
                (dur/1000) + " sec";
        }
    }, 100);

```

play

To start or resume playing an audio file, use the `play()` function, as shown in the following example:

```
media.play();
```

To actually play a file, you must pass in a URI, or a path to a locally stored file, as shown here:

```

function playFile(url) {
    // Play the audio file at url
    var audio_file = new Media(url, onSuccess, onError);
    audio_file.play();
}

```

Traditionally, you'd insert a Play link or button on the interface with the URL of the music file passed in, as shown here:

```

<a href="#" onclick="playFile('http://example.com/audio/mysong.mp3');">Play
  Audio</a>

```

The `play` function is supported on iOS and Android devices.

pause

To pause the playing of an audio file, use the `pause()` method, as shown here:

```
media.pause();
```



NOTE `pause()` is a synchronous function.

Following is an example of an audio file being paused after 15 seconds (a great way to offer up an audio sample, by the way):

```

function playFile(url) {
    // Play the audio file at url
    var my_file = new Media(url,onSuccess,onError);

    // Play audio
    my_file.play();

    // Pause after 10 seconds

```

```
        setTimeout(function() {  
            media.pause();  
        }, 15000);  
    }
```

A more traditional way to do this is to have the pause handled in its own function, and then create a link or button that triggers that function, as shown here:

```
function pauseMedia(){  
    media.pause();  
}  
  
<a href="#" onClick="pauseMedia();">Pause</a>
```

release

On an Android device, you can release the underlying operating system's audio resources using the `release()` method. This is particularly important because there is a finite number of OpenCore instances for media playback. The OpenCore libraries support media playback for audio, video, and image formats. You should always call this release function when you no longer need a media resource. Following is an example:

```
media.release();
```

So, the complete process would look like this:

```
var my_file = new Media(src, onSuccess, onError);  
  
my_file.play();  
my_file.stop();  
my_file.release();
```

startRecord

To start recording, use the `startRecord()` method, as shown here:

```
function startRecording() {  
    var src = "recording.mp3";  
    var myRecording = new Media(src, onSuccess, onError);  
    myRecording.startRecord();  
}
```

If you're using an iOS device, you'll need to use the `startAudioRecord()` method, and the file you stipulate to record to must already exist. Following is an example:

```
function startRecording() {  
    var src = "recording.mp3"; //must exist!  
    var myRecording = new Media(src, onSuccess, onError);  
    myRecording.startAudioRecord();  
}
```

Of course, you'd create a link or button that triggers the `startRecording()` function, as shown here:

```
<a href="#" onClick="startRecording();">Start Recording</a> stopRecord
```

To stop recording a file, use the `stopRecord()` method, as shown here:

```
function stopRecording(){
    media.stopRecord();
}
```

To actually make this function work, you'd use a link that triggers the function, as shown here:

```
<a href="#" onClick="stopRecording();">Stop Recording</a>
```

If you're using an iOS device, you'll need to use the `stopAudioRecord()` method, as shown here:

```
function stopRecording(){
    media.stopAudioRecord();
}
```

stop

To stop the playing of an audio file, use the `stop()` method, as shown here:

```
media.stop();
```

To make this work, wrap it in a function, and then trigger it with a link or button, as shown here:

```
function stopMedia(){
    media.stop();
}
<a href="#" onClick="stopMedia()">Stop Playing</a>
```

HANDLING ERRORS

The `MediaError` object is returned whenever you use the `mediaError` callback function. It contains two parts:

- `code` — This is one of the predefined codes.
- `message` — This is an error message that describes the details of the error.

The predefined error codes are constants:

- `MediaError.MEDIA_ERR_ABORTED`
- `MediaError.MEDIA_ERR_NETWORK`
- `MediaError.MEDIA_ERR_DECODE`
- `MediaError.MEDIA_ERR_NONE_SUPPORTED`

A more user-friendly approach would be to set up an array of error states that use the constants as keys, and your own friendlier error statement as the value. That way, you could display the friendlier message to the user. Following is an example:

```
function onError(myerror){
    var errors = {};
    errors[MediaError.MEDIA_ERR_ABORTED]= 'Stopped playing!';
    errors[MediaError.MEDIA_ERR_NETWORK]= 'Network error!';
    errors[MediaError.MEDIA_ERR_DECODE] = 'Could not decode file!';
    errors[MediaError.MEDIA_ERR_NONE_SUPPORTED] = 'Format not supported!';

    alert('Media error: ' + error[myerror]);
}
```

TRY IT OUT Building a Simple Audio Player

Let's create a complete application that plays an audio file. Enter the following code:



Available for
download on
Wrox.com

```
<html>
<head>
    <title>Media Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

        document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady() {
    playAudio("http://example.com/audio.mp3"); //change me!
}

var my_media = null;
var mediaTimer = null;

// Play audio
//
function playAudio(src) {
    // Create Media object from src
    my_media = new Media(src, onSuccess, onError);

    // Play audio
    my_media.play();

    // Update my_media position every second
    if (mediaTimer == null) {
        mediaTimer = setInterval(function() {
            // get my_media position
            my_media.getCurrentPosition(
                // success callback
                function(position) {
                    if (position > -1) {
                        setAudioPosition((position/1000) + " sec");
                    }
                }
            );
        }, 1000);
    }
}
```



```

        },
        // error callback
        function(e) {
            console.log("Error getting pos=" + e);
            setAudioPosition("Error: " + e);
        }
    );
}, 1000);
}

function pauseAudio() {
    if (my_media) {
        my_media.pause();
    }
}

function stopAudio() {
    if (my_media) {
        my_media.stop();
    }
    clearInterval(mediaTimer);
    mediaTimer = null;
}

function onSuccess() {
    console.log("playAudio():Audio Success");
}

function onError(error) {
    alert('code: ' + error.code + '\n' +
        'message: ' + error.message + '\n');
}

function setAudioPosition(position) {
    document.getElementById('audio_position').innerHTML = position;
}

</script>
</head>
<body>
    <a href="#" onclick="playAudio('http://example.com/audio.mp3');">
        Play Audio</a>
    <a href="#" onclick="pauseAudio();">Pause Playing Audio</a>
    <a href="#" onclick="stopAudio();">Stop Playing Audio</a>
    <p id="audio_position"></p>
</body>
</html>

```

Code file [chapter9a.html] available for download at Wrox.com.

How It Works

As soon as the application loads, it starts playing an audio file at a specified URI . The user can also play the same file by pressing the Play Audio button.

Now that you know how to build a simple audio player, let's look a simple application that will record audio.

TRY IT OUT Building an App That Records Audio

In this exercise, you create a complete application that records audio. Enter the following code:



Available for
download on
Wrox.com

```
<html>
<head>
  <title>Device Properties Example</title>

  <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
  <script type="text/javascript" charset="utf-8">

    document.addEventListener("deviceready", onDeviceReady, false);

    function recordAudio() {
      var src = "myrecording.mp3";
      var mediaRec = new Media(src, onSuccess, onError);

      mediaRec.startRecord();
    }

    function onDeviceReady() {
      startRecording();
    }

    function startRecording() {
      recordAudio();
    }

    function stopRecording(){
      mediaRec.stopRecord();
    }

    function onSuccess() {
      console.log("recordAudio():Audio Success");
    }

    function onError(error) {
      alert('code: ' + error.code + '\n' +
        'message: ' + error.message + '\n');
    }

    function setAudioPosition(position) {
```

```

        document.getElementById('audio_position').innerHTML = position;
    }

</script>
</head>
<body>
    <a href="#" onClick="startRecording()">Start Recording</a><br/><br/>
    <a href="#" onClick="stopRecording()">Stop Recording</a>
    <p id="audio_position"></p>
</body>
</html>

```

Code file [chapter9b.html] available for download at Wrox.com.

How It Works

As soon as the application loads, the user can start recording and stop recording using two links at the bottom of the HTML window.

Now, let's discuss how to apply some design.

IMPROVING THE LOOK AND FEEL

As has been the case with the applications you built in the past few chapters, the look and feel of the media player application is very bare and minimal — almost uninviting — as shown in Figure 9-7. It is also difficult to use because it uses links.



FIGURE 9-7: Current media player application look and feel

So, to improve this user interface (UI), first ensure that your project has the necessary jQTouch folders in it, as described in Chapter 7. Place the following code above the first call to load `phonegap.js`:

```
<style type="text/css" media="screen">@import "jqtouch/jqtouch.css";</style>
<style type="text/css" media="screen">@import "themes/jqt/theme.css";</style>
<script src="jqtouch/jquery-1.4.2.js" type="text/javascript" charset="utf-8"></script>
<script src="jqtouch/jqtouch.js" type="application/x-javascript" charset="utf-8"></script>
<script type="text/javascript" charset="utf-8">
    var jQT = new $.jQTouch({
        icon: 'jqtouch.png',
        icon4: 'jqtouch4.png',
        addGlossToIcon: false,
        startupScreen: 'jqt_startup.png',
        statusBar: 'black',
        preloadImages: [
            'themes/jqt/img/activeButton.png',
            'themes/jqt/img/back_button.png',
            'themes/jqt/img/back_button_clicked.png',
            'themes/jqt/img/blueButton.png',
            'themes/jqt/img/button.png',
            'themes/jqt/img/button_clicked.png',
            'themes/jqt/img/grayButton.png',
            'themes/jqt/img/greenButton.png',
            'themes/jqt/img/redButton.png',
            'themes/jqt/img/whiteButton.png',
            'themes/jqt/img/loading.gif'
        ]
    });
</script>
```

Next, take out the old HTML that's inside the `body` tags and replace it with the following:

```
<div id="home" class="current">
    <div class="toolbar">
        <h1>Audio Player</h1>

    </div>
    <ul class="rounded">
        <li id='play' class='whitebutton' onclick="playAudio('http://example.com/audio.mp3');">Play</li>
        <li id='pause' class='graybutton' onclick="pauseAudio();">Pause</li>
        <li id='stop' class='graybutton' onclick="stopAudio();">Stop</li>
    </ul>
    <p id="audio_position"></p>
</div>
```

Notice that this code assigns classes to the list items. The Play button has a class of `whitebutton`, and the other two buttons get a class of `graybutton`. These classes are part of the jQTouch cascading style sheet (CSS) scheme. The result is similar to what is shown in Figure 9-8.



FIGURE 9-8: Improved media player interface

SUMMARY

In this chapter, you learned how to use the `Media` API to record and play back audio files.

EXERCISES

1. Create an application that will start recording after a 5-second delay.
2. Play a local media file.



NOTE *Answers to the Exercises can be found in Appendix A.*

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
<code>media.play</code>	Use <code>media.play</code> to play a previously recorded file.
<code>media.pause</code>	Use <code>media.pause</code> to pause a previously recorded file.
<code>media.stop</code>	Use <code>media.stop</code> to stop playing a previously recorded file.
<code>media.startRecording</code>	Use <code>media.startRecording</code> to start a recording.
<code>media.stopRecording</code>	Use <code>media.stopRecording</code> to stop a recording.
<code>media.getDuration</code>	Use <code>media.getDuration</code> to find out the duration of a recording.
<code>media.getCurrentPosition</code>	Use <code>media.getCurrentPosition</code> to find out current position in a recording.

10

Camera

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Getting to know the camera
- Using the Camera object
- Using the `getPicture` method
- Using Camera options and arguments

In Chapter 9, you learned how to use the `Media` API to play and record audio files. In this chapter, you learn how to use the device's built-in camera.

GETTING TO KNOW THE CAMERA

In this section, you learn about the camera (and photo gallery) and how you can use it.

What Is the Camera?

Just about every smartphone these days comes with a camera. Unfortunately, that's about all you can say. There are absolutely no standards that govern megapixels, optical zoom, the existence of a flash, shutter speed, and ISO. (For the non-photographers, ISO is used to define sensitivity to light — the higher the number, the more light is gathered. Therefore, it reduces the amount of flash, for example, that you might use in a low-light situation.) Some cameras support video capture, and some don't. Those that do offer different support for HD capture. Still others support removable memory cards.

Following are some examples of the wide array of included camera features on various devices:

- HTC Sensation 4G (Android) comes with an 8-megapixel camera and dual LED flash.
- Samsung Gravity (also Android) comes with a 3-megapixel camera and an LED flash.
- iPhone 4 comes with a 5-megapixel camera with LED flash and front-facing VGA camera for video conferencing and self-portraits.
- iPhone 3GS comes with a 3-pixel camera and no flash.
- Palm webOS-powered pre-2 comes with a 5-megapixel camera with LED flash, but the Palm webOS Pixi has a 2-megapixel camera with LED flash.
- BlackBerry Torch 9800 has a 5-megapixel camera with LED flash.
- BlackBerry Bold 9780 comes with a 5-megapixel camera, but the BlackBerry Bold 9000 has a 2-megapixel camera, both with flash.

So, what's the bottom line? When you use the `Camera` API, you'll be able to control the picture quality only to a certain degree — that is, you'll always be bound by the quality of the camera you're working with.

What Is the Photo Gallery?

If your smartphone has a camera, it also has a photo gallery. The photo gallery is a repository of pictures taken with the device's camera, and other pictures you've synched from your main computer to the smartphone.

On most phones, the photo gallery usually displays a grid of thumbnail images, any of which you can tap to view a larger version of the image. Figure 10-1 shows an iPhone 3GS photo gallery and Figure 10-2 shows a typical Android photo gallery.



FIGURE 10-1: iPhone 3GS photo gallery

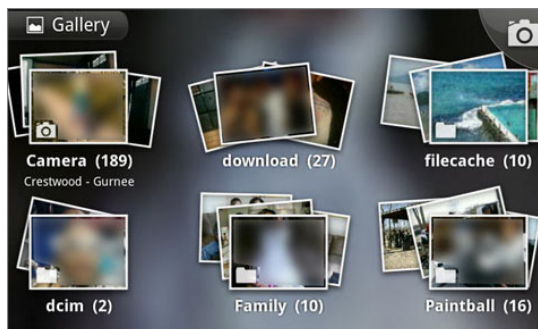


FIGURE 10-2: Android photo gallery

When you're viewing a single image, you can perform a variety of operations, such as share the photo via e-mail or delete the image. Figure 10-3 shows what happens when you tap the share icon on an iOS device.

Also, most devices have the capability to organize photos into photo albums. These are usually imported from a laptop or desktop computer, for example, or consist of photos from the photo gallery that have been further categorized by the phone's user.

It is important to mention the photo gallery and photo albums because when you work with the `Camera` API, you have the option of taking a picture with the camera, or selecting an image from the gallery or photo albums.

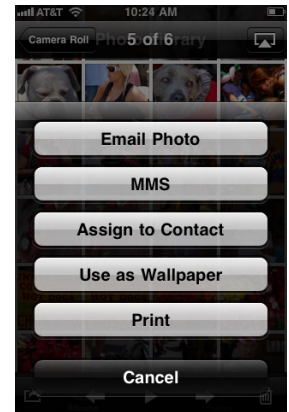


FIGURE 10-3: Sharing a photo on an iOS device

Using the Camera

People love to take pictures, and having a reasonably good camera built into the phone they're already carrying makes it very easy to snap pictures at parties, on vacation, or anywhere you happen to be. Even those who would never consider themselves good photographers (or who are intimidated by the options on a digital camera, or even a point-and-shoot camera) can use their smartphones to capture memories.

Here are just three creative ways to use the camera as part of any application:

- Allow users to take a picture, add metadata to that picture (title, tags, and so on), and then share that picture with friends on social networks.
- Create a scavenger hunt game that involves taking pictures of items on a list, and sharing those pictures with a small group of people.
- Add filters and effects to photos taken with the camera (or selected from the photo gallery). These effects can be simple (such as adding borders or changing brightness/contrast) or just plain fun (such as adding fun-house mirror distortions to faces).

Showcase of Camera Applications

Running a simple Google search for “iPhone camera apps” or “Android camera apps” returns a giant list of available camera applications. Although most smartphones come with built-in camera/photo gallery services, this fact hasn't stopped everyone from building camera-related tools.

A popular camera application is “Instagram,” shown in Figure 10-4. “Instagram” is a free app that lets you take a picture, apply a bit of tilt-shift magic and one of about a dozen filters, and then post that picture to members of the service. Others can then view your images, leave comments, and “like” what you've done.

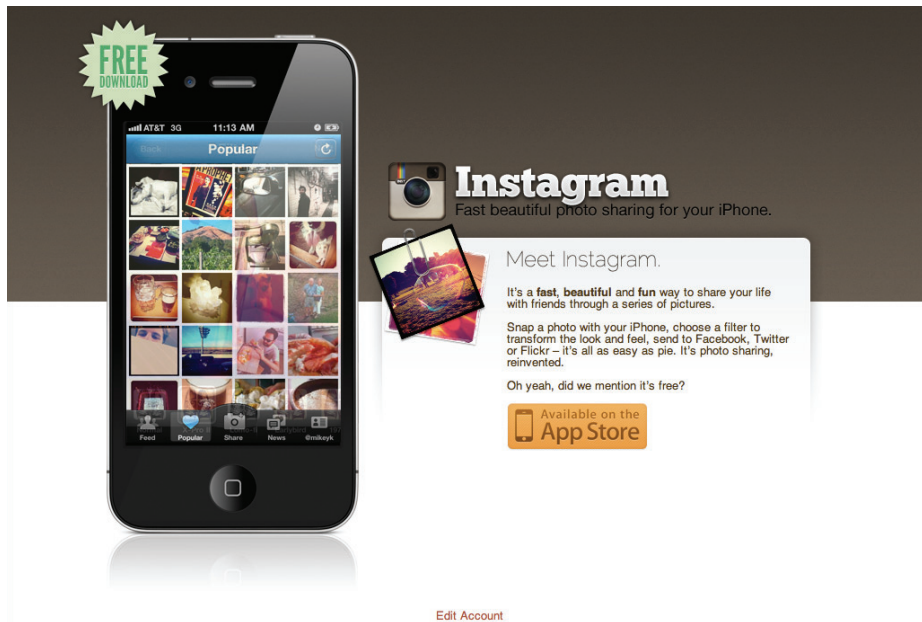


FIGURE 10-4: “Instagram” application



NOTE Tilt-shift is a digital way to simulate how medium- and large-format photographers could tilt their cameras to create miniaturized scenes.

Flickr, which was one of the first social photography services (now part of Yahoo!) also has a mobile app. You can easily view photos of your own and from your photo galleries of your friends with the app, as shown in Figure 10-5.

One of the most interesting and successful camera applications is “CameraBag,” which is shown in Figure 10-6. It contains a dozen modern and vintage photo effects that you can apply to your pictures. You can even save multiple effects onto your images.

“QuadCamera” (shown in Figure 10-7) enables you to take a series of timed photos and then places them into a four-up or eight-up grid. It’s a great way to create interesting collages of different subjects.



FIGURE 10-5: Flickr photo sharing on a mobile device

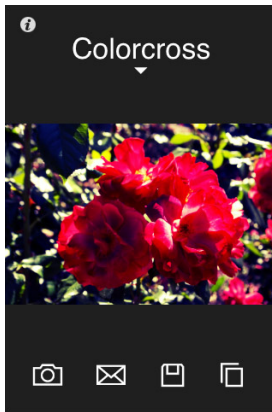


FIGURE 10-6: “CameraBag” application

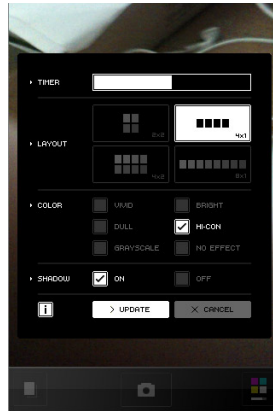


FIGURE 10-7: “QuadCamera” application

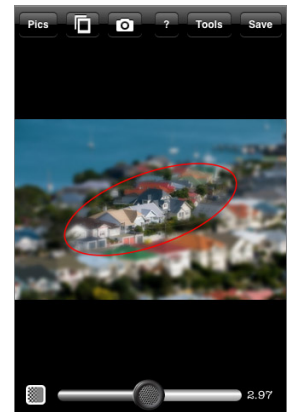


FIGURE 10-8: “TiltShift” application

“TiltShift” lets you apply tilt-shift filters to a picture you’ve taken. The by-product of tilt-shifting is that the subjects appear to be little toys, as you can see in Figure 10-8. Of course, if used selectively, it can also simulate depth-of-field to a certain degree.

USING THE CAMERA OBJECT

Whether you work with the camera directly, or pick an image from the photo gallery, you’ll be interacting with the `Camera` object. This object has one method, `getPicture()`, which you learn more about shortly. Following are the options for the `Camera` object:

- The `sourceType`, which might be `PHOTOLIBRARY` or `SAVEDPHOTOALBUM` (to grab a photo from the photo gallery, or from an imported photo album, respectively) or `CAMERA` to take a picture.
- On `CameraSuccess`, you can return a Base64-encoded photo image (`DATA_URL`, which is the default setting) or a path to the image file location in the local storage (`FILE_URI`). On some smartphones, it might be better to make use of the `FILE_URI` to save memory.
- You can tweak the quality of the picture taken, and even allow editing of the file afterward.

USING THE GETPICTURE METHOD

In this section, you learn how to use the one and only method of the `Camera` object: `getPicture()`.

You use the `camera.getPicture()` method to take a picture, or prompt the user to select a picture from a photo gallery, as shown here:

```
navigator.camera.getPicture( cameraSuccess, cameraError, [ cameraOptions ] );
```

This function opens the default camera application on the device and either takes a picture (this is the default mode), or allows the user to select a picture from the photo gallery. It then closes the camera application and returns the user to your application.

The `cameraSuccess` callback function contains the image data from the camera operation. For example, the following code snippet shows how you would take an image with the camera at 35 percent quality and return the data as a Base64-encoded image:

```
navigator.camera.getPicture(onSuccess, onFail, { quality: 35 });

function onSuccess(imgData) {
    var image = document.getElementById('camera_image');
    image.src = "data:image/jpeg;base64," + imgData;
}

function onFail(message) {
    alert('Failed because: ' + message);
}
```

And here's how to do the same thing, except retrieve an image file from a specified location:

```
var cameraOptions = { quality: 35, destinationType:
    Camera.DestinationType.FILE_URI };

navigator.camera.getPicture(onSuccess, onFail, cameraOptions);

function onSuccess(imguri) {
    var image = document.getElementById('camera_image');
    image.src = imageuri;
}

function onFail(message) {
    alert('Failed because: ' + message);
}
```

To capture an editable photo, this is what you'd do:

```
navigator.camera.getPicture(onSuccess, onFail, { quality: 50, allowEdit:true });

function onSuccess(imgData) {
    var image = document.getElementById('camera_image');
    image.src = "data:image/jpeg;base64," + imgData;
}

function onFail(message) {
    alert('Failed because: ' + message);
}
```

To prompt the user to select a photo from the photo gallery, use the following:

```
navigator.camera.getPicture(onSuccess, onFail, { quality: 50, destinationType:
    destinationType.FILE_URI, sourceType:pictureSource.PHOTOLIBRARY });

function onSuccess(imgUri) {
```

```

        var image = document.getElementById('camera_image');
        image.src = imageUrl;
    }

    function onFail(message) {
        alert('Failed because: ' + message);
    }

```

Following is an example of how to prompt the user to select a photo from a saved photo album:

```

navigator.camera.getPicture(onSuccess, onFail, { quality: 50, destinationType:
    destinationType.FILE_URI, sourceType:pictureSource.SAVEDPHOTOALBUM });

function onSuccess(imgUri) {
    var image = document.getElementById('camera_image');
    image.src = imageUrl;
}

function onFail(message) {
    alert('Failed because: ' + message);
}

```

USING CAMERA OPTIONS

You have access to four optional parameters to customize what the camera does:

- **quality** — This is a number between 0 and 100. The higher the number, the better the quality. Some devices won't handle the memory load of a high-quality picture.
- **destinationType** — This is the format of the return value. This is either Base64-encoded data (`DATA_URL`) or a file path (`FILE_URI`).
- **sourceType** — This is the source of the image. By default, this is set to `CAMERA`, but can also be `PHOTOLIBRARY` or `SAVEDPHOTOALBUM`.
- **allowEdit** — If set to `true`, this allows simple editing of images before selection.

For example, the following options set quality to 70 percent, return a Base64-encoded image, and allow editing (the `CAMERA` is set by default):

```

var options = { quality:70, destinationType: Camera.DestinationType.DATA_URL,
    allowEdit: true };
navigator.camera.getPicture(onSuccess, onFail, options);

```

To set quality to 70 percent, return a Base64-encoded image, and allow editing when you choose from the photo library, you could use the following:

```

var options = { quality:70, destinationType: Camera.DestinationType.DATA_URL,
    allowEdit: true, sourceType: Camera.PictureSourceType.PHOTOLIBRARY };
navigator.camera.getPicture(onSuccess, onFail, options);

```


Of course, depending on the type of device the user is running, you'll end up running into quite a few quirks. Table 10-1 highlights some idiosyncrasies for various devices.

TABLE 10-1: Idiosyncrasies for Various Devices

DEVICE	IDIOSYNCRASY
Android	Ignores the <code>allowEdit</code> parameter.
	<code>PHOTOLIBRARY</code> and <code>SAVEDPHOTOALBUM</code> both display the same photo gallery.
BlackBerry	Ignores the <code>quality</code> , <code>sourceType</code> , and <code>allowEdit</code> parameters.
	Application must have key injection permissions to close the native Camera application after a photo is taken.
	Taking a very large image may result in the inability to encode the image for later-model devices with higher-resolution cameras.
Palm	Ignores the <code>quality</code> , <code>sourceType</code> , and <code>allowEdit</code> parameters.
iOS	You must set <code>quality</code> below 50 to avoid memory errors on some devices.
	When you use <code>FILE_URI</code> to save your image, photos are saved to the application's <code>Documents/tmp</code> directory. Unfortunately, that directory is deleted when the application closes, so you'll need to manage the process of saving images to a more permanent location.

TRY IT OUT Building a Simple Camera Application

Let’s create a complete application that takes a photo with the default camera application. Enter the following:



Available for download on Wrox.com

```
<!DOCTYPE html>
<html>
  <head>
    <title>Take a Photo</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready",onDeviceReady,false);

      function onDeviceReady() {
      }

      // Called when a photo is successfully retrieved
      //
      function onPhotoSuccess(imageData) {
var myImage = document.getElementById('myImage');

myImage.style.display = 'block';

myImage.src = "data:image/jpeg;base64," + imageData;
```

```

    }

    function capturePhoto() {
        navigator.camera.getPicture(onPhotoSuccess, onPhotoFail, { quality: 50 });
    }

    function onPhotoFail(message) {
        alert('Failed because: ' + message);
    }

</script>
</head>
<body>
    <button onclick="capturePhoto();">Capture Photo</button> <br>

    <img style="display:none;width:60px;height:60px;" id="myImage" src="" />
</body>
</html>

```

Code file [chapter10a.html] available for download at Wrox.com.


How It Works

As soon as the application loads, the user can take a picture by clicking the Capture Photo button. This button triggers the `capturePhoto()` function, which takes a picture with the default camera (with a quality of 50 percent), and then stuffs the Base64-encoded image into the `myImage` document object model (DOM) element in the HTML.

Now that you know how the camera works, let's allow the user to select an image from either a photo gallery or a photo album.

TRY IT OUT Select an Image

In this exercise, you create a complete application that allows the user to select an image from either a photo gallery or a saved photo album. Enter the following:



Available for download on Wrox.com

```

<!DOCTYPE html>
<html>
<head>
    <title>Capture Photo</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

        document.addEventListener("deviceready",onDeviceReady,false);

        function onDeviceReady() {
        }

        function onPhotoSuccess(imageURI) {

```

```
var myImg = document.getElementById('myImage');

myImg.style.display = 'block';

myImg.src = imageURI;
}

function getPhoto(source) {
    navigator.camera.getPicture(onPhotoSuccess, onFail, { quality: 50,
        destinationType: destinationType.FILE_URI,
        sourceType: source });
}

function onFail(message) {
    alert('Failed because: ' + message);
}

</script>
</head>
<body>
<button onclick="getPhoto(pictureSource.PHOTOLIBRARY);">From Photo Library
    </button><br>
    <button onclick="getPhoto(pictureSource.SAVEDPHOTOALBUM);">From Photo
        Album</button><br>
    <img style="display:none;" id="myImage" src="" />
</body>
</html>
```

Code file [chapter10b.html] available for download at Wrox.com.

How It Works

Instead of taking a picture with the default camera application, this example allows the user to select an image from the photo gallery or a photo album.

Next, let's use jQTouch to spruce up the first application you wrote.

IMPROVING THE LOOK AND FEEL

When you look at the first camera application (that is, the one that takes a photo), you get the impression that it's so Spartan even the Spartans would feel uncomfortable around it. Figure 10-9 shows how the first camera application looks on the device.



FIGURE 10-9: Minimal display resulting from first application

You can change that up fairly quickly. Ensure that your project has the necessary jQTouch folders in it, as described in Chapter 7. Place the following code above the first call to load `phonegap.js`:

```
<style type="text/css" media="screen">@import "jqtouch/jqtouch.css";</style>
<style type="text/css" media="screen">@import "themes/jqt/theme.css";</style>
<script src="jqtouch/jquery-1.4.2.js" type="text/javascript" charset="utf-8"></script>
<script src="jqtouch/jqtouch.js" type="application/x-javascript" charset="utf-8"></script>
<script type="text/javascript" charset="utf-8">
  var jQT = new $.jQTouch({
    icon: 'jqtouch.png',
    icon4: 'jqtouch4.png',
    addGlossToIcon: false,
    startupScreen: 'jqt_startup.png',
    statusBar: 'black',
    preloadImages: [
      'themes/jqt/img/activeButton.png',
      'themes/jqt/img/back_button.png',
      'themes/jqt/img/back_button_clicked.png',
      'themes/jqt/img/blueButton.png',
      'themes/jqt/img/button.png',
      'themes/jqt/img/button_clicked.png',
      'themes/jqt/img/grayButton.png',
      'themes/jqt/img/greenButton.png',
      'themes/jqt/img/redButton.png',
      'themes/jqt/img/whiteButton.png',
```

```

        'themes/jqt/img/loading.gif'
    ]
    });
</script>

```

Next, take out the old HTML that's inside the `body` tags and put the following in there:

```

    <div id="home" class="current">
    <div class="toolbar">
        <h1>Camera</h1>

    </div>
    <ul class="rounded">
    <li id='takepicture' class='redButton' onclick="capturePhoto();">Take
        Photo</li>
    </ul>
    <img style="display:none;width:60px;height:60px;" id="myImage" src="" />
    </div>

```

Notice the use of a class of `redButton` to really make the button stand out, as shown in Figure 10-10.

Of course, there's also a `blueButton` class, which is probably even more startling, as shown in Figure 10-11. You get the idea.



FIGURE 10-10: Making the button stand out



FIGURE 10-11: Using the `blueButton` class

SUMMARY

In this chapter, you learned how to use the camera and photo gallery features on your smartphone with the Camera API.

EXERCISES

1. Create a 3-second delay before taking a photo.
2. Add an option that allows the user to choose the quality of a photo taken with a camera.



NOTE Answers to the Exercises can be found in Appendix A.

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPT
<code>camera.getPicture</code>	Use <code>camera.getPicture</code> to take a picture with the camera.

11

Storage

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Learning about storage options
- Using the Database object
- Using the localStorage object
- Learning about options and arguments

Up until now, you've been working with parts of the PhoneGap API that deal with the accelerometer, the compass, the camera, and other device-specific capabilities.

In this chapter, you learn how to use HTML5 storage options directly in your code. The PhoneGap API has support for both web SQL databases and the local storage option. You learn how to use each of them here.

LEARNING ABOUT STORAGE OPTIONS

If you're coming from the world of PHP, Rails, or Django development (to name just a few), then you're used to building web applications that store their data in some kind of database (most likely, MySQL).

Furthermore, if you've been doing any HTML5 development, you've probably run into the database options available on the client side. For the first time, PhoneGap developers are able to create client-side applications that use a somewhat reliable database back end.

For example, if you use Chrome or Safari to visit html5demos.com/database, you will see the page fill up with tweets. Now, at first glance, this page appears to be powered by a traditional database like MySQL. But if you right-click the page and click Inspect Element, you'll see something similar to Figure 11-1.

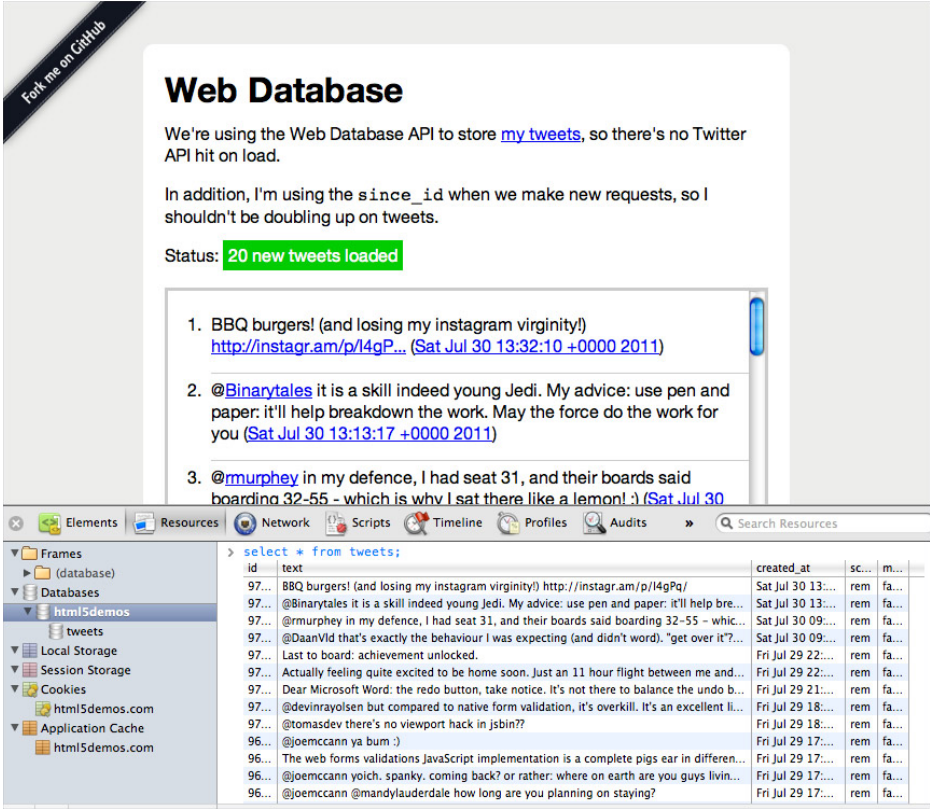


FIGURE 11-1: Result of clicking Inspect Element

If you click the Resources tab and then scroll down to the databases portion, you will see that that an `html5demos` database exists, and inside that is a table called `tweets`.

If you were to click in the right pane and issue the SQL command `select * from tweets`, you'd get a listing very similar to the one shown at the bottom of Figure 11-1.

As you can see, the fields here are very similar to what you're used to working with in `MySQL` or another SQL database. Of course, you can also just click the icon for the `tweets` table to see everything that's stored in that table.

`Web Database` (essentially an implementation of `Sqlite3`) is good for complex data storage requirements (in other words, when you want to keep track of many different columns of information on each item), but sometimes your storage requirements aren't quite as complex.

In fact, you may only want to store some very simple information (as in a key/value pair), and only keep it around for a very short period of time (in other words, until the user closes the browser or application). If this is more your speed, then the local and session storage methods are what you want.

There's a demo over at `html5demos.com/storage`. Figure 11-2 shows the result of entering a value of `hello` for the `sessionStorage` variable, and a value of `goodbye` for the `localStorage` variable.

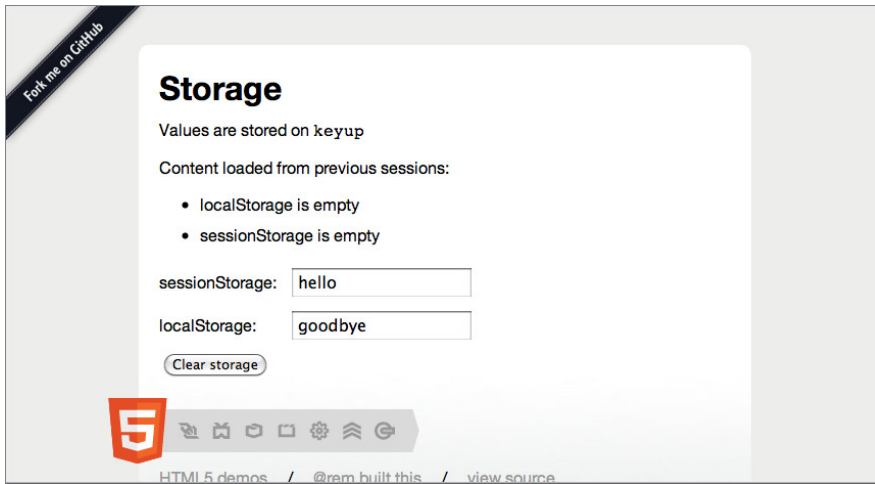


FIGURE 11-2: Entering values for sessionStorage and localStorage

If you still have the Resources tab open, scroll down to `Session Storage` to see the value of `hello` stored there, as shown in Figure 11-3. In Figure 11-4 you can see the value of `goodbye` stored in `Local Storage`.

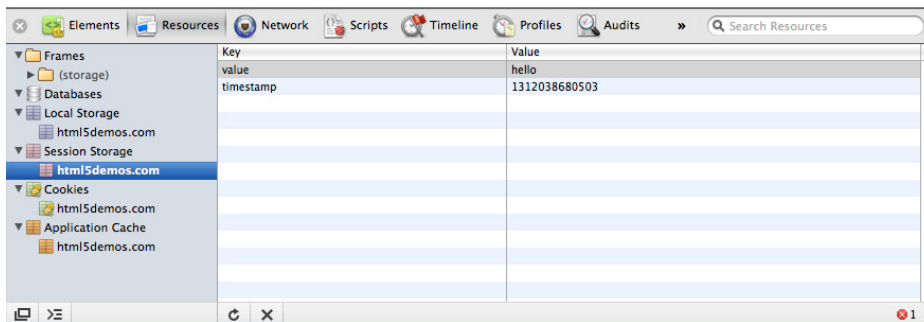


FIGURE 11-3: Value of `hello` stored in Session Storage

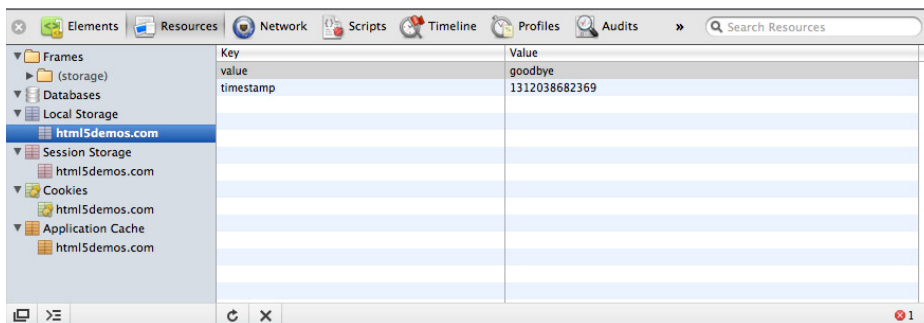


FIGURE 11-4: Value of `goodbye` stored in Local Storage

What's the difference between `Session Storage` and `Local Storage`? Well, the `Session Storage` objects are deleted when you close the browser window, but the `Local Storage` ones persist.

Another thing to remember is that `Storage` objects are usually good for 5 MB to 10 MB of data, depending on the browser used. Because you'll be building apps that go on a phone, you should keep your `Storage` objects smaller than 4 MB to be safe. If you're building web databases, the 5 MB limit also holds, but you should keep it smaller than that if you can.

The question then becomes, if HTML5 supports both database and local storage options, why bother with PhoneGap's `Storage` implementation? Well, some devices will already support these features, and, in that case, the PhoneGap API will defer to the device implementation. Other devices don't have this support, and, in those cases, PhoneGap's implementation will step in.

USING THE DATABASE OBJECT

In this section, you learn how to work with databases. Specifically, you learn how to do the following:

- Open a database
- Run a SQL query
- View a result set

Once you've finished with this section, you'll be able to create just about any database to support your PhoneGap applications.

Opening a Database

In the world of SQLite, you create a database by opening it. To open a database, use the `window.openDatabase()` method, as shown here:

```
var myDB = window.openDatabase(name, version, displayname, size);
```

This command takes four arguments:

- `name` — This is the name you want to give to the database.
- `version` — This is the version of the database.
- `displayname` — This is the display name of the database.
- `size` — This is the size of the database (in bytes).

For example, you can run the following command:

```
var myDB = window.openDatabase("photos", "1.0", "Photos DB", 1000000);
```

This creates a 1 MB database called `photos` and assigns it to the `myDB` variable. Typically, you'll see this kind of activity attached to a function, or even the `onDeviceReady()` listener, as shown here:


```
document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady() {
    var myDB = window.openDatabase("photos", "1.0", "Photos DB", 1000000);
}
```

Running a SQL Query

When you run a SQL query, what you're really doing is using the `SQLTransaction` object. This object contains methods that let you execute SQL statements against a previously opened database.

Here's the basic format of the `executeSql()` method:

```
myDB.executeSql('SELECT * FROM table1');
```



NOTE In this example, `myDB` is used only because it was used in the previous example. You would need to use whatever database variable you've already opened. Also note that you can use whatever legal SQL statement you want. The example shows a select statement, but you could run others.

Following is an example of a drop statement:

```
myDB.executeSql('DROP TABLE IF EXISTS table1');
```

Here's a create table statement:

```
myDB.executeSql('CREATE TABLE IF NOT EXISTS table1 (id unique, firstname
    varchar, lastname varchar)');
```

Following is an insert statement:

```
myDB.executeSql('INSERT INTO TABLE (id, firstname, lastname) VALUES (1,
    "Thomas ", "Myer ")');
```

And here's a delete statement:

```
myDB.executeSql('DELETE FROM TABLE where id=1');
```

There's nothing very surprising here. If you know your way around SQL, you won't have any trouble creating tables, inserting data, and retrieving it. Of course, you will have to watch out how you nest your quotation marks — if you're using single quotation marks on the outside, use double quotation marks on values inside your statements.

If you need to run a whole bunch of queries at once, you can create a transaction, which then calls your queries, as shown here:

```
myDB.transaction(populateDB, errorDB, successDB);
function populateDB(tx) {
```

```
tx.executeSql('DROP TABLE IF EXISTS table1');
tx.executeSql('CREATE TABLE IF NOT EXISTS table1 (id unique, data varchar)');
tx.executeSql('INSERT INTO table1 (id, data) VALUES (1, "testing 1")');
tx.executeSql('INSERT INTO table1 (id, data) VALUES (2, "testing 2")');
}

function errorDB(err) {
    alert("Error processing SQL: "+err);
}

function successDB() {
    alert("success!");
}
```

Note that the transaction calls a JavaScript function to execute the individual queries, and includes calls to `success` and `error` callback functions. You learn more about result sets next.

Viewing a Result Set

One of the more common things you'll do with your database is storing and retrieving values from it. Whenever you run a transaction, you'll want to know how it all turned out.

The transaction method provides you with a `success` callback. The `executeSql()` method also provides you with `success` and `error` callbacks. It's in that `success` callback that you'll see your result set.

If you're used to working with `mySQL` (or some other SQL database), you may think this is all you need to know. However, don't get too far ahead of yourself. What you get back from the `success` callback function is the `SQLResultSet` object, which contains the following three properties:

- `insertId` — This is the row ID that the `SQLResultSet` object's SQL statement inserted into the database. (This applies only if you ran an `insert` statement.)
- `rowAffected` — This is the number of rows that were changed by the SQL statement. (This returns 0 if none were affected, as with a `select` statement.)
- `rows` — This is a `SQLResultSetRowList` representing the rows returned.

In the following example, let's run a very basic SQL `select` query and then report back what is returned:

```
function queryDB(tx) {
    tx.executeSql('SELECT * FROM test', [], querySuccess, errorDB);
}

function querySuccess(tx, results) {
    // this will be empty -- no rows were inserted.
    alert("Insert ID = " + results.insertId);
    // this will be 0 -- select statement
    alert("Rows Affected = " + results.rowAffected);
    // the number of rows returned by the select statement
    alert("# of rows = " + results.rows.length);
}
```

```

    }

    function errorDB(err) {
        alert("Error processing SQL: "+err.code);
    }

    db.transaction(queryDB, errorDB);

```

As far as the `rows` property goes, you can access the values stored in them using the `item()` method. Each time you use it, you get back a row of data specified by the index you pass it. You end up with a JavaScript object with properties that match the database column names from your `select` statement.

So, to rewrite the `querySuccess()` function from the previous example, you would use the following:

```

function querySuccess(tx, results) {
    //first get the number of rows in the result set
    var len = results.rows.length;
    for (var i=0;i<len;i++){
        alert("Row = " + i + " Firstname = " + results.rows.item(i).firstname + "
            Lastname = " + results.rows.item(i).lastname);
    }
}

```

The previous example uses a `for` loop to “loop” through the record set. The `for` loop is a good construct here because you don’t know how many times you must loop through a record set, because each record set could be a different length, depending on the query and the stored data.

In order to loop through a record set, you must know the length of that record set (hence the use of `results.rows.length`). But once you have that, you can start at the first record retrieved, and process each row using `results.row.item(i).field_name`, where `field_name` is a match for the database field name.

Handling Errors

If there’s an error, PhoneGap throws a `SQLException` object, which contains the following two properties:

- `code` — This is one of the predefined error codes.
- `message` — This is a description of the error.

The error codes themselves are constants, as shown here:

- `SQLException.UNKNOWN_ERR`
- `SQLException.DATABASE_ERR`
- `SQLException.VERSION_ERR`
- `SQLException.TOO_LARGE_ERR`
- `SQLException.QUOTA_ERR`

- `SQLError.SYNTAX_ERR`
- `SQLError.CONSTRAINT_ERR`
- `SQLError.TIMEOUT_ERR`

TRY IT OUT Building a Simple Database

A very common requirement when you're building a database-backed application is to populate the database as soon as the device is ready. Of course, it wouldn't do you any good to completely wipe out the user's data every time he or she opened your app, so you need to put in a few error checks. Enter the following code to create a database app:



Available for
download on
Wrox.com

```
<!DOCTYPE html>
<html>
  <head>
    <title>Database Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function populateDB(tx) {
        tx.executeSql('DROP TABLE IF EXISTS test1');
        tx.executeSql('CREATE TABLE IF NOT EXISTS test1 (id unique, name)');
        tx.executeSql('INSERT INTO test1 (id, name) VALUES (1, "Tony")');
        tx.executeSql('INSERT INTO test1 (id, name) VALUES (2, "Bill")');
        tx.executeSql('INSERT INTO test1 (id, name) VALUES (3, "Thomas")');
      }

      function queryDB(tx) {
        tx.executeSql('SELECT * FROM test1', [], querySuccess, errorCallback);
      }

      // Query the success callback
      //
      function querySuccess(tx, results) {
        //first get the number of rows in the result set
        var len = results.rows.length;
        var status = document.getElementById("status");
        var string = "Rows: " +len+"<br/>";

        for (var i=0;i<len;i++){
          string += results.rows.item(i).name + "<br/>";
        }

        status.innerHTML = string;
      }

      function errorDB(err) {
        alert("Error processing SQL: "+err.code);
      }
    </script>
  </head>
</html>
```

```

    }

    function successDB() {
        var db = window.openDatabase("Test", "1.0", "Test", 200000);
        db.transaction(queryDB, errorCallback);
    }

    function onDeviceReady() {
        var db = window.openDatabase("Test", "1.0", "Test", 200000);
        db.transaction(populateDB, errorCallback, successDB);
    }

    </script>
</head>
<body>
    <h1>Names</h1>
    <div id='status'></div>
</body>
</html> Code file [chapter11a.html] available for download at Wrox.com.

```

How It Works

As soon as the application loads and the device is ready, the code runs `openDatabase()` to create a test database. Following that, the code then runs a series of transactions that create a `test1` database table, and inserts three records into that database.

Once that particular transaction runs, the `success` callback runs, which calls a second transaction that queries the database. The result of that function is to loop through the record set and display the names in the document object model (DOM) element with an ID of `status`.

Figure 11-5 shows what this might look like. (It's bare-bones, but it works.)

Now, let's suppose that you introduce some kind of error into your database code, such as a misspelling when you run your `select` statement, as shown here:

```

function queryDB(tx) {
    tx.executeSql('SELEC * FROM test1', [], querySuccess, errorCallback);
}

```

PhoneGap will throw an error and display that error to you in an alert notification (because that's what's defined in the `error` callback function). That error will look something like Figure 11-6.



FIGURE 11-5: Implementation of the database application

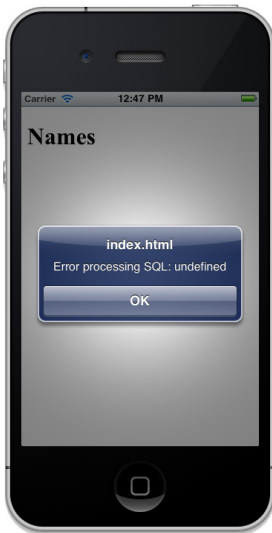


FIGURE 11-6: PhoneGap throwing an error

Now that you’ve built a simple database application, let’s look at the `localStorage` object.

USING THE LOCALSTORAGE OBJECT

If you must store values for your application, you may decide that working with a database is just too much of a hassle. All you may need is some way to store and retrieve key/value pairs, and, thus, add some kind of persistence to your app.

The `localStorage` object does exactly that, giving you access to the local storage interface. To use it, invoke `window.localStorage`, as shown here:

```
var myStorage = window.localStorage;
```

The `localStorage` object has the following five methods:

- `key` — This returns the name of the key at the position specified.
- `getItem` — This returns the item identified by its key.
- `setItem` — This saves an item at the key provided.
- `removeItem` — This removes an item identified by the provided key.
- `clear` — This removes all the key/value pairs.

For example, to retrieve a specific key, the one at position 5, you would use the following:

```
var myKeyName = window.localStorage.key(5);
```

To store a value, use the following:

```
var myValue = window.localStorage.setItem("first_name", "Thomas");
```

To retrieve a value, use the following:

```
var myName = window.localStorage.getItem("first_name");
//myName is now equal to Thomas
```

To remove an item, use the following:

```
window.localStorage.removeItem("first_name");
```

To clear out all keys, use the following:

```
window.localStorage.clear();
```

TRY IT OUT Storing and Retrieving Key/Value Pairs

This exercise includes a very simple app that prompts the user to enter a name, stores that value in local storage, and then displays that value as the user types. Enter the following code:



```
<!DOCTYPE html>
<html>
  <head>
    <title>localStorage Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {

      }

      function storeName(){
        var myName = document.getElementById("name").value;
        window.localStorage.setItem("name", myName);

        var result = document.getElementById("result");
        result.innerHTML = "Your name is " + window.localStorage.getItem("name");
        return false;
      }

    </script>
  </head>
```

```

<body>
  <form>
    <p>What's your name?<br/>
    <input type='text' id='name' onKeyUp='storeName()'></p>

    </form>

    <div id='result'></div>
  </body>
</html> Code file [chapter11b.html] available for download at Wrox.com.

```

How It Works

As soon as the app loads, the user can enter his or her first name into the field. As he or she enters each key, the `storeName()` function is triggered, which takes the value from the input field and stores it in the local storage variable called `name`. Finally, it displays this value in the result DOM element, as shown in Figure 11-7.

Of course, if you only wanted the results to show up when you're finished, attach an `onBlur()` handler to the text input field. As soon as the user taps Done on the keyboard, or, for example, moves to another field, the `storeName()` function will trigger.

So, the new code would look like this:



```

<!DOCTYPE html>
<html>
  <head>
    <title>localStorage Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {

      }

      function storeName(){
        var myName = document.getElementById("name").value;
        window.localStorage.setItem("name", myName);

        var result = document.getElementById("result");
        result.innerHTML = "Your name is " + window.localStorage.getItem("name");
        return false;
      }

    </script>

```

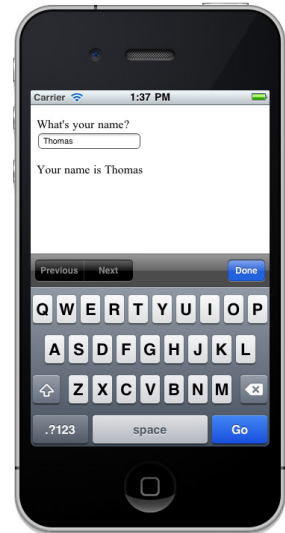


FIGURE 11-7: Result of user entering a name


```
</head>
<body>
  <form>
    <p>What's your name?<br/>
    <input type='text' id='name' onBlur='storeName()'></p>

  </form>

  <div id='result'></div>
</body>
</html> Code file [chapter11c.html] available for download at Wrox.com.
```

That's all you need to know about working with databases and local storage to get you started.

SUMMARY

In this chapter, you learned how to store values in a database or in local storage. You learned how to open a database; create transactions; execute SQL statements; and get, set, and remove local storage data. In Chapter 12, you learn about the filesystem.

EXERCISES

1. On the `localStorage` example, ask the user to enter his or her first and last name.
2. Apply the jQTouch look and feel to the database example.



NOTE Answers to the Exercises can be found in Appendix A.

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
<code>window.openDatabase()</code>	Use <code>window.openDatabase()</code> to open a database.
<code>db.transaction()</code>	Use <code>db.transaction()</code> to create a transaction.
<code>tx.executeSql()</code>	Use <code>tx.executeSql()</code> to execute a SQL statement.
<code>window.localStorage.key(x)</code>	Use <code>window.localStorage.key(x)</code> to get a <code>localStorage</code> key.
<code>window.localStorage.setItem(key,value)</code>	Use <code>window.localStorage.setItem(key,value)</code> to set a <code>localStorage</code> item.
<code>window.localStorage.getItem(key)</code>	Use <code>window.localStorage.getItem(key)</code> to retrieve a <code>localStorage</code> item.
<code>window.localStorage.removeItem(key)</code>	Use <code>window.localStorage.removeItem(key)</code> to remove a <code>localStorage</code> item.
<code>window.localStorage.clear()</code>	Use <code>window.localStorage.clear()</code> to clear out all <code>localStorage</code> items.

12

Files

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Learning about filesystems
- Learning about directories and files
- Learning how to transfer files
- Handling errors

In Chapter 11, you learned how to work with storage options — both local storage and databases. In this chapter, you learn how to work with filesystems, directories, and files. By the end of this chapter, you'll know how to access the filesystem, read and write files, and navigate directories. You'll also be able to transfer files from the device to a remote server.

LEARNING ABOUT FILESYSTEMS

Regardless of manufacturer or operating system, every smartphone or device has a filesystem, and the PhoneGap API provides some access to that filesystem.

At the end of the day, that access is usually fairly limited. For example, it's difficult (if not impossible) to jump outside the sandbox your application lives in. However, let's assume that you're not here to learn about jailbreaking your phone's 16 GB (for example) memory capacity.

What this chapter focuses on is what probably matters most to you: how to access the filesystem so that you can read a file that's already there, or write some data to a file.

PhoneGap's `FileSystem` object represents information about the filesystem. It has the following two properties:

- `name` — This is the name of the filesystem.
- `root` — This is the root directory of the filesystem.

This object is returned as the success callback of the `requestFileSystem()` method. Following is an example:

```
window.requestFileSystem(LocalFileSystem.PERSISTENT, 0, onSuccess, onFail);

function onSuccess(fileSystem){
    alert(fileSystem.name);
    alert(fileSystem.root.name);
}

function onFail(event){
    alert(event.target.error.code);
}
```

The `root` property contains a `DirectoryEntry` object, which you learn about next.

LEARNING ABOUT DIRECTORIES AND FILES

Two kinds of items are found in filesystems: directories and files. Directories are represented by `DirectoryEntry` objects, and files are represented by `FileEntry` objects.

In this section, you learn about the following:

- Using the `DirectoryEntry` object
- Using the `FileEntry` object
- Using flags
- Using `LocalFileSystem`

Using the `DirectoryEntry` Object

This object represents a directory on a filesystem. It has the following properties:

- `isFile` — This is always `false`.
- `isDirectory` — This is always `true`.
- `name` — This is the name of the `DirectoryEntry`, excluding the path leading to it.
- `fullPath` — This is the full absolute path from the root to the `DirectoryEntry` object.

Furthermore, `DirectoryEntry` has the following methods:

- `getMetadata`
- `moveTo`
- `copyTo`
- `toURI`
- `remove`
- `getParent`

- `createReader`
- `getDirectory`
- `getFile`
- `removeRecursively`

getMetadata

This method is used to look up metadata about a directory. It features two callback functions as parameters:

- `successCallback` — This is a callback that is called with a `Metadata` object.
- `errorCallback` — This is a callback that is called if an error occurs retrieving the `Metadata` object.

Following is an example:

```
function onSuccess(metadata) {
    alert("Last Modified: " + metadata.modificationTime);
}

function onFail(error) {
    alert(error.code);
}

// Request the metadata object for this entry
entry.getMetadata(onSuccess, onFail);
```

moveTo

This method is used to move a directory to a different location on the filesystem. Any of the following conditions will create an error:

- If you try to move a directory inside itself, or to any child at any depth.
- If you try to move a directory into its parent if a name that is different from its current one is not provided.
- If you try to move a directory to a path occupied by a file.
- If you try to move a directory to a path occupied by a directory that is not empty.

Following are the parameters for this method:

- `parent` — This is the parent directory to which to move the directory.
- `newName` — This is the new name of the directory. It defaults to the current name if unspecified.
- `successCallback` — This is a callback that is called with the `DirectoryEntry` object of the new directory.
- `errorCallback` — This is a callback that is called if an error occurs when attempting to move the directory.

Following is an example:

```
function success(entry) {
    alert("New Path: " + entry.fullPath);
}

function fail(error) {
    alert(error.code);
}

function moveDirectory(entry) {
    var parent = document.getElementById('parent').value,
        newName = document.getElementById('newName').value,
        parentEntry = new DirectoryEntry({fullPath: parent});

    // move the directory to a new directory and rename it
    entry.moveTo(parentEntry, newName, success, fail);
}
```

copyTo

This method is used to copy a directory to a different location on the filesystem. Any of the following will create an error:

- If you try to copy a directory inside itself at any depth.
- If you try to copy a directory into its parent if a name that is different from its current one is not provided.



NOTE Remember that directory copies are always recursive — that is, they copy all contents of the directory.

Following are the parameters for this method:

- `parent` — This is the parent directory to which to copy the directory.
- `newName` — This is the new name of the directory. It defaults to the current name if unspecified.
- `successCallback` — This is a callback that is called with the `DirectoryEntry` object of the new directory.
- `errorCallback` — This is a callback that is called if an error occurs when attempting to copy the underlying directory. It is invoked with a `FileError` object.

Following is an example:

```
function success(entry) {
    alert("New Path: " + entry.fullPath);
}

function fail(error) {
```

```

        alert(error.code);
    }
    function copyDirectory(entry) {
        var parent = document.getElementById('parent').value,
            newName = document.getElementById('newName').value,
            parentEntry = new DirectoryEntry({fullPath: parent});

        // copy the directory to a new directory and rename it
        entry.copyTo(parentEntry, newName, success, fail);
    }

```



NOTE Note that, in the previous example, the names of the directory to copy (and the name of the new directory) are pulled from DOM elements, specifically text input fields. This is a good way to allow input from users, but you probably want to add some kind of validation in case they decide to pass in a malicious string.

toURI

This method returns a URI that can be used to locate a directory. Following is an example:

```

// Get the URI for this directory
var uri = entry.toURI();
alert(uri);

```

remove

This method deletes a directory. Any of the following will create an error:

- If you try to delete a directory that is not empty.
- If you try to delete the root directory of a filesystem.

The parameters consist of two callback functions:

- **successCallback** — This is a callback that is called after the directory has been deleted. It is invoked with no parameters.
- **errorCallback** — This is a callback that is called if an error occurs when attempting to delete the directory.

Following is an example:

```

function success(entry) {
    alert("Removal succeeded");
}

function fail(error) {
    alert('Error removing directory: ' + error.code);
}

// remove this directory
entry.remove(success, fail);

```

getParent

This method is used to look up the parent `DirectoryEntry` containing the current directory. The parameters consist of two callback functions:

- `successCallback` — This is a callback that is called with the directory's parent `DirectoryEntry` object.
- `errorCallback` — This is a callback that is called if an error occurs when attempting to retrieve the parent `DirectoryEntry` object. It is invoked with a `FileError` object.

Following is an example:

```
function success(parent) {  
    alert("Parent: " + parent.name);  
}  
  
function fail(error) {  
    alert('Failed to get parent: ' + error.code);  
}  
  
// Get the parent DirectoryEntry  
entry.getParent(success, fail);
```

createReader

The way you read entries in a directory is to use the `createReader` method. Following is an example:

```
// create a directory reader  
var directoryReader = entry.createReader();
```

getDirectory

This method creates or looks up an existing directory. If you try to create a directory whose immediate parent does not yet exist, you'll get an error.

Following are the parameters:

- `path` — This is the path to the directory to be looked up or created. This can be a relative or absolute path.
- `options` — These are options to specify whether the directory is created if it doesn't exist.
- `successCallback` — This is a callback that is invoked with a `DirectoryEntry` object.
- `errorCallback` — This is a callback that is called if an error occurs when creating or looking up the directory. It is invoked with a `FileError` object.

Following is an example:

```
function success(parent) {  
    alert("Parent: " + parent.name);  
}
```



```

    }

    function fail(error) {
        alert("Unable to create new directory: " + error.code);
    }

    // Retrieve an existing directory, or create it if it does not already exist
    entry.getDirectory("sampleFiles", {create: true, exclusive: false}, success, fail);

```

getFile

This method creates or looks up a file. If you try to create a file whose immediate parent does not yet exist, you'll get an error.

Following are the parameters:

- **path** — This is the path to the file to be looked up or created. This can be a relative or absolute path.
- **options** — These are options to specify whether the file is created if it doesn't exist.
- **successCallback** — This is a callback that is invoked with a `FileEntry` object.
- **errorCallback** — This is a callback that is called if an error occurs when creating or looking up the file. It is invoked with a `FileError` object.

Following is an example:

```

function success(parent) {
    alert("Parent: " + parent.name);
}

function fail(error) {
    alert("Failed to retrieve file: " + error.code);
}

// Retrieve an existing file, or create it if it does not exist
entry.getFile("sample_data.txt", {create: true, exclusive: false}, success, fail);

```

removeRecursively

This deletes a directory and all of its contents. In the event of an error (for example, if you try to delete a directory that contains a file that cannot be removed), some of the contents of the directory may be deleted. If you try to delete the root directory of a filesystem, you'll get an error.

The parameters consist of two callback functions:

- **successCallback** — This is a callback that is called after the `DirectoryEntry` object has been deleted. It is invoked with no parameters.
- **errorCallback** — This is a callback that is called if an error occurs when attempting to delete the `DirectoryEntry` object. It is invoked with a `FileError` object.

Following is an example:

```
function success(parent) {
    alert("Delete succeeded!");
}

function fail(error) {
    alert("Failed to delete directory or it's contents: " + error.code);
}

// remove the directory and all it's contents
entry.removeRecursively(success, fail);
```

Using the FileEntry Object

This object represents a file on a filesystem. It has the following properties:

- `isFile` — This is always true.
- `isDirectory` — This is always false.
- `name` — This is the name of the `FileEntry` object, excluding the path leading to it.
- `fullPath` — This is the full absolute path from the root to the `FileEntry` object.

This object also has the following methods:

- `getMetadata`
- `moveTo`
- `copyTo`
- `toURI`
- `remove`
- `getParent`
- `createWriter`
- `file`

getMetadata

This method is used to look up metadata about a file. Remember that file metadata is separate from its data — the *data* would be what's stored in the file (words, images, and so on), whereas *metadata* would be more like the creation time for the file, the file's size, and so on.

The parameters consist of two callback functions:

- `successCallback` — This is a callback that is called with a `Metadata` object.
- `errorCallback` — This is a callback that is called if an error occurs while retrieving the `Metadata` object. It is invoked with a `FileError` object.

Following is an example:

```
function success(metadata) {
    alert("Last Modified: " + metadata.modificationTime);
}

function fail(error) {
    alert(error.code);
}

// Request the metadata object for this entry
entry.getMetadata(success, fail);
```

moveTo

This method is used to move a file to a different location on the filesystem. If you try to do any of the following, you'll get an error:

- Try to move a file into its parent if a name that is different from its current one isn't provided.
- Try to move a file to a path occupied by a directory.
- Try to move a file on top of an existing file without first deleting the existing file.

Following are the parameters:

- **parent** — This is the parent directory to which to move the file.
- **newName** — This is the new name of the file. It defaults to the current name if unspecified.
- **successCallback** — This is a callback that is called with the `FileEntry` object of the new file.
- **errorCallback** — This is a callback that is called if an error occurs when attempting to move the file. It is invoked with a `FileError` object.

Following is an example:

```
function success(entry) {
    alert("New Path: " + entry.fullPath);
}

function fail(error) {
    alert(error.code);
}

function moveFile(entry) {
    var parent = document.getElementById('parent').value,
        parentEntry = new DirectoryEntry({fullPath: parent});

    // move the file to a new directory and rename it
    entry.moveTo(parentEntry, "new_file.txt", success, fail);
}
```

copyTo

This method is used to copy a file to a new location on the filesystem. If you try to copy a file into its parent, and a file with that name already exists there, you'll get an error.

Following are the parameters:

- **parent** — This is the parent directory to which to copy the file.
- **newName** — This is the new name of the file. It defaults to the current name if unspecified.
- **successCallback** — This is a callback that is called with the `FileEntry` object of the new file.
- **errorCallback** — This is a callback that is called if an error occurs when attempting to copy the file.

Following is an example:

```
function success(entry) {  
    alert("New Path: " + entry.fullPath);  
}  
  
function fail(error) {  
    alert(error.code);  
}  
  
function copyFile(entry) {  
    var parent = document.getElementById('parent').value,  
        parentEntry = new DirectoryEntry({fullPath: parent});  
  
    // copy the file to a new directory and rename it  
    entry.copyTo(parentEntry, "file.copy.txt", success, fail);  
}
```

toURI

This method is used to return a URI that can be used to locate the file. Following is an example:

```
// Request the metadata object for this entry  
var uri = entry.toURI();  
console.log(uri);
```

remove

This method is used to delete a file. The parameters consist of two callback functions:

- **successCallback** — This is a callback that is called after the file has been deleted. It is invoked with no parameters.
- **errorCallback** — This is a callback that is called if an error occurs when attempting to delete the file. It is invoked with a `FileError` object.

Following is an example:

```
function success(entry) {
    alert("Removal succeeded");
}

function fail(error) {
    alert('Error removing file: ' + error.code);
}

// remove the file
entry.remove(success, fail);
```

getParent

This method is used to look up the parent `DirectoryEntry` object containing the file. The parameters consist of two callback functions:

- **successCallback** — This is a callback that is called with the file's parent `DirectoryEntry` object.
- **errorCallback** — This is a callback that is called if an error occurs when attempting to retrieve the parent `DirectoryEntry` object.

Following is an example:

```
function success(parent) {
    alert("Parent Name: " + parent.name);
}

function fail(error) {
    alert(error.code);
}

// Get the parent DirectoryEntry
entry.getParent(success, fail);
```

createWriter

You can write a file to a directory by using the `createWriter` method. It creates a `FileWriter` object associated with the file. The parameters consist of two callback functions:

- **successCallback** — This is a callback that is called with a `FileWriter` object.
- **errorCallback** — This is a callback that is called if an error occurs while attempting to create the `FileWriter` object.

Following is an example:

```
function success(writer) {
    writer.write("Some text to the file");
}

function fail(error) {
```

```
        alert(error.code);
    }

    // create a FileWriter to write to the file
    entry.createWriter(success, fail);
```

file

This method is used to return a `File` object that represents the current state of the file represented by the `FileEntry` object. The parameters consist of two callback functions:

- `successCallback` — This is a callback that is called with a `File` object.
- `errorCallback` — This is a callback that is called if an error occurs when creating the `File` object (in other words, the underlying file no longer exists).

Following is an example:

```
function success(file) {
    alert("File size: " + file.size);
}

function fail(error) {
    alert("Unable to retrieve file properties: " + error.code);
}

// obtain properties of a file
entry.file(success, fail);
```

Using Flags

This object is used to supply arguments to the `DirectoryEntry` object's `getFile` and `getDirectory` methods, which you can use to look up or create files and directories, respectively.

Following are the properties:

- `create` — This is used to indicate that the file or directory should be created, if it does not exist.
- `exclusive` — This is used with `create`. The `exclusive` flag causes the file or directory creation to fail if the target path already exists.

Following is an example:

```
// Get the images directory, creating it if it doesn't exist.
imagesDir = fileSystem.root.getDirectory("images", {create: true});

// Create the status file, if and only if it doesn't exist.
statusFile = dataDir.getFile("statusfile.txt", {create: true, exclusive: true});
```

Using LocalFileSystem

This object provides a way to obtain root filesystems. It has two methods:

- `requestFileSystem` — This method is used to request a filesystem.
- `resolveLocalFileSystemURI` — This method is used to retrieve a `DirectoryEntry` or `FileEntry` object using a local URI.

In turn, two constants define a local filesystem:

- `LocalFileSystem.PERSISTENT` — This is used for storage that should not be removed by the user agent without permission from the application or user.
- `LocalFileSystem.TEMPORARY` — This is used for storage with no guarantee of persistence.

Following is an example that requests the local filesystem:

```
function onSuccess(fileSystem) {
    console.log(fileSystem.name);
}

// request the persistent file system
window.requestFileSystem(LocalFileSystem.PERSISTENT, 0, onSuccess, onError);
```

Here's an example that resolves the local system URI:

```
function onSuccess(fileEntry) {
    console.log(fileEntry.name);
}

window.resolveLocalFileSystemURI("file:///myconfigfile.txt", onSuccess, onError);
```

READING FILES

You use the `FileReader` object to read a file. The `FileReader` object is a way to read files from the device's filesystem. Files can be read as text, or as a Base64-encoded data string.

This object has the following properties:

- `readyState` — This is one of the three states (`EMPTY`, `LOADING`, or `DONE`) the reader can be in.
- `result` — This is used to get the contents of the file that has been read.
- `error` — This is used for an object containing errors.
- `onloadstart` — This is called when the read starts.
- `onload` — This is called when the read has successfully completed.
- `onabort` — This is called when the read has been aborted (for example, by invoking the `abort()` method).
- `onerror` — This is called when the read has failed.
- `onloadend` — This is called when the request has completed (either in success or failure).

This object has the following methods:

- `abort`
- `readAsDataURL`
- `readAsText`

abort

You can stop reading a file by using the `abort` method, as shown in the following example:

```
function success(file) {
    var reader = new FileReader();
    reader.onloadend = function(event) {
        alert("read success");
        alert(event.target.result);
    };
    reader.readAsText(file);
    reader.abort();
}

function fail(error) {
    console.log(error.code);
}

entry.file(success, fail);
```

readAsDataURL

You can read a file and return the data as Base64-encoded data, which is ideal for images and other binary files. Following is an example of how to do this:

```
function success(file) {
    var reader = new FileReader();
    reader.onloadend = function(evt) {
        alert("read success");
        alert(evt.target.result);
    };
    reader.readAsDataURL(file);
}

function fail(evt) {
    console.log(error.code);
}

entry.file(success, fail);
```

readAsText

You can also read a file and return text, which is best suited for `txt` files, as shown in the following example:


```

function success(file) {
    var reader = new FileReader();
    reader.onloadend = function(evt) {
        alert("read success");
        alert(evt.target.result);
    };
    reader.readAsText(file);
}

var fail = function(event) {
    console.log(error.code);
}

entry.file(sucess, fail);

```

WRITING FILES

You can use the `FileWriter` object to write files. A `FileWriter` object is created for a single file, and, once created, it can be used to write to a file multiple times.

The `FileWriter` object maintains the file's position and length attributes, so you can seek and write anywhere in the file. By default, the `FileWriter` object writes to the beginning of the file, and will overwrite existing data. You set the optional `append` Boolean value to `true` in the `FileWriter`'s constructor to begin writing at the end of the file.

Following are properties for this object:

- `readyState` — This is one of the three states (`INIT`, `WRITING`, or `DONE`) the reader can be in.
- `fileName` — This is the name of the file to be written.
- `length` — This is the length of the file to be written.
- `position` — This is the current position of the file pointer.
- `error` — This is an object containing errors.
- `onwritestart` — This is called when the write starts.
- `onwrite` — This is called when the request has completed successfully.
- `onabort` — This is called when the write has been aborted (for example, by invoking the `abort()` method).
- `onerror` — This is called when the write has failed.
- `onwriteend` — This is called when the request has completed (either in success or failure).

The `FileWriter` object has the following methods:

- `abort` — This is used to abort the writing of a file.
- `seek` — This is used to move the file pointer to the byte specified.
- `truncate` — This is used to shorten the file to the length specified.
- `write` — This is used to write data to the file.

Following is an example:

```
function success(writer) {
    writer.onwrite = function(evt) {
        alert("write success");
    };
    writer.write("some sample text");
}

var fail = function(evt) {
    alert(error.code);
}

entry.createWriter(success, fail);
```

To append data to a file, you would use code similar to the following:

```
function success(writer) {
    writer.onwrite = function(evt) {
        alert("write success");
    };
    writer.seek(writer.length);
    writer.write("appended text");
}

var fail = function(evt) {
    alert(error.code);
}

entry.createWriter(success, fail);
```

LEARNING ABOUT TRANSFERRING FILES

The `FileTransfer` object lets you upload files to a remote server using an HTTP multi-part POST request. Both HTTP and HTTPS protocols are supported. You can specify optional parameters with the `FileUploadOptions` object. When you successfully upload a file, the results are stored in a `FileUploadResult` object. All errors are stored in a `FileTransferError` object.

Following is an example:

```
function success(r) {
    alert("Code = " + r.responseCode);
    alert("Response = " + r.response);
    alert("Sent = " + r.bytesSent);
}

function fail(error) {
    alert("An error has occurred: Code = " + error.code);
}

var options = new FileUploadOptions();
```

```

options.fileKey="file";
options.fileName=fileURI.substr(fileURI.lastIndexOf('/')+1);
options.mimeType="text/plain";

var params = new Object();
params.value1 = "sample text";
params.value2 = "another sample";

options.params = params;

var ft = new FileTransfer();
ft.upload(fileURI, "http://example.com/upload.php", success, fail, options);

```

FileUploadOptions

As mentioned, you can pass along a set of options when you perform a file upload. These options are stored in a `FileUploadOptions` object.

The `FileUploadOptions` object has the following properties:

- `fileKey` — This is the name of the form element. If you don't set it, it defaults to `file`.
- `fileName` — This is the filename you want the file to be saved as on the server. If not set, it defaults to `image.jpg`.
- `mimeType` — This is the MIME type of the data you are uploading. If not set, it defaults to `image/jpeg`.
- `params` — This is a set of optional key/value pairs you can pass along in the HTTP request.

FileUploadResults

When you successfully upload a file, the results are stored in a `FileUploadResult` object. This object has the following properties:

- `bytesSent` — This is the number of bytes sent to the server as part of the upload.
- `responseCode` — This is the HTTP response code returned by the server.
- `response` — This is the HTTP response returned by the server.

HANDLING ERRORS

When you're doing complex operations like reading or writing files to the filesystem, or uploading a file to a remote server, errors will happen, and you'll need a good way to handle them when they do. PhoneGap has the capability to throw a `FileError` or `FileTransferError`.

FileError

A `FileError` object is set when an error occurs in any of the `File` API methods. It contains one property, `code`, which is one of the following predefined codes:

- `FileError.NOT_FOUND_ERR`
- `FileError.SECURITY_ERR`
- `FileError.ABORT_ERR`
- `FileError.NOT_READABLE_ERR`
- `FileError.ENCODING_ERR`
- `FileError.NO_MODIFICATION_ALLOWED_ERR`
- `FileError.INVALID_STATE_ERR`
- `FileError.SYNTAX_ERR`
- `FileError.INVALID_MODIFICATION_ERR`
- `FileError.QUOTA_EXCEEDED_ERR`
- `FileError.TYPE_MISMATCH_ERR`
- `FileError.PATH_EXISTS_ERR`

FileTransferError

A `FileTransferError` object is returned via the error callback when an error occurs. It contains one property, `code`, which is one of the following predefined codes:

- `FileTransferError.FILE_NOT_FOUND_ERR`
- `FileTransferError.INVALID_URL_ERR`
- `FileTransferError.CONNECTION_ERR`

TRY IT OUT Transferring Files to a Remote Server

It's fairly likely that you'll want to transfer a file from the phone the user is holding in his or her hand to a remote server on the Internet. This exercise provides an example of how to take a picture with the camera, and then upload the file to the server. Enter the following code:



Available for
download on
Wrox.com

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

<html>
<head>
  <title>File Transfer Example</title>

  <script type="text/javascript" charset="utf-8"
    src="phonegap.js"></script>
```

```

<script type="text/javascript" charset="utf-8">

document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady() {

    // Take a picture then upload
    navigator.camera.getPicture(uploadPhoto,
        function(message) { alert('get picture
            failed'); },
        { quality: 50,
          destinationType:
            navigator.camera
              .DestinationType.FILE_URI,
          sourceType:
            navigator.camera.PictureSourceType
              .CAMERA}
        );

}

function uploadPhoto(imageURI) {
    var options = new FileUploadOptions();
    options.fileKey="file";
    options.fileName=imageURI.substr(imageURI.lastIndexOf('/')+1);
    options.mimeType="image/jpeg";

    var params = new Object();
    params.value1 = "test value";
    params.value2 = "param value";

    options.params = params;

    var ft = new FileTransfer();
    ft.upload(imageURI, "http://example.com/upload.php", success, fail,
        options);
}

function success(r) {
    alert("Code = " + r.responseCode);
    alert("Response = " + r.response);
    alert("Sent = " + r.bytesSent);
}

function fail(error) {
    alert("An error has occurred: Code = " + error.code);
}

</script>
</head>
<body>
    <h1>Upload file</h1>
</body>
</html>

```

How It Works

As soon as the app is loaded, it prompts the user to take a photo with the default camera. Once the picture is taken by the user, the file URI of that stored image is passed to the `uploadPhoto()` function, which bundles that URI along with some other options, and passes it all up to the remote server.

That's all you need to know about working with the filesystem.

SUMMARY

In this chapter, you learned how to work with the filesystem, directories, and files. Specifically, you learned how to read from the filesystem (including navigating directories and reading file data), writing to the filesystem, and uploading files to a remote server.

In Chapter 13, you learn about working with contacts.

EXERCISES

1. Upload a photo from the photo gallery to a server.
-

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
<code>window.requestFileSystem()</code>	Use <code>window.requestFileSystem()</code> to open a filesystem.
<code>readAsText()</code>	Use <code>readAsText()</code> to read a file as text.
<code>readAsDataURL()</code>	Use <code>readAsDataURL()</code> to read a file as Base64-encoded data.
<code>write()</code>	Use <code>write()</code> to write to a file.
<code>seek()</code> and <code>write()</code>	Use <code>seek()</code> and <code>write()</code> to append to a file.

13

Contacts

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Learning about creating contacts
- Learning about finding contacts
- Handling errors

No matter what kind of device a person owns (iOS, Android, BlackBerry, Palm webOS, and so on), it is bound to have a `Contacts` database built in to it. The `Contacts` database contains all the contacts that a person stores on the phone — friends, family, business colleagues, and others — but that isn't a guarantee that every phone manufacturer will follow a standard when it comes to the contact fields it stores and retrieves.

That being said, the PhoneGap `Contacts` API does a remarkably good job of letting users create, update, and find contacts, all of which you learn about in this chapter.

LEARNING ABOUT CREATING CONTACTS

The easiest way to learn how to use the `Contacts` API is to create a contact. You create a contact with the `contacts.create()` method, like this:

```
var myContact = navigator.contacts.create(properties);
```

When you use this method, it creates a new `Contact` object that you can then manipulate.

Following is an example:

```
var properties = {"displayName": "Joe Smith", "gender": "male"; }  
var myContact = navigator.contacts.create(properties);
```

You could also do this:

```
var myContact = navigator.contacts.create();
myContact.displayName = "Joe Smith";
myContact.gender = "male";
```

Here is a full list of properties you can work with:

- `id` — This is a globally unique identifier (GUID).
- `displayName` — This is the name of this contact, suitable for display to end users.
- `name` — This is an object containing all components of a person's name.
- `nickname` — This is a casual name to by which you can address the contact.
- `phoneNumbers` — This is an array of all the contact's phone numbers.
- `gender` — This is the gender of the contact.
- `emails` — This is an array of all the contact's e-mail addresses.
- `addresses` — This is an array of all the contact's addresses.
- `ims` — This is an array of all the contact's instant messaging (IM) addresses.
- `organizations` — This is an array of all the contact's organizations.
- `birthday` — This is the birthday of the contact.
- `note` — This is a note about the contact.
- `photos` — This is an array of the contact's photos.
- `categories` — This is an array of all the contact's user-defined categories.
- `urls` — This is an array of web pages associated with the contact.

The best way to save a lot of this information (especially, for example, when it comes to the arrays expected for `phoneNumbers`) is to use the `ContactField` object.

The `ContactField` object is a general-purpose wrapper that enables you to create a data field and then add it to a contact. For example, if you want to store a number of e-mails for a contact, you could use something similar to the following:

```
var myContact = navigator.contacts.create();
//store emails
var emails = [2];
emails[0] = new ContactField('work', 'tom@example.com', false);
emails[1] = new ContactField('home', 'tomhome@example.com', false);
contact.emails = emails;
contact.save();
```

The `ContactName` object contains various properties that make up a name, such as the following:

- `formatted` — This is the complete name.
- `familyName` — This is the contact's surname.

- `givenName` — This is the contact's first name.
- `middleName` — This is the contact's middle name.
- `honorificPrefix` — This contains the contact's prefix (such as Dr., Ms., or something similar).
- `honorificSuffix` — This contains the contact's suffix (such as Esq, Jr., or MD).

The `ContactAddress` object has the following properties:

- `pref` — This is set to `true` if this `ContactAddress` contains the user's preferred address.
- `type` — This is a string that tells you what type of address this is (for example, `'home'`).
- `formatted` — This is the full address formatted for display.
- `streetAddress` — This is the full street address.
- `locality` — This is the city or locality.
- `region` — This is the state or region.
- `postalCode` — This is the ZIP code or postal code.
- `country` — This is the country name.

The `ContactOrganization` object contains properties that store a contact's organization (that is, place of employment). Following are the properties for this object:

- `pref` — This is set to `true` if this is the organization preferred by the user.
- `type` — This tells you what type of field this is (for example, `'office'`).
- `name` — This holds the name of the organization.
- `department` — This holds the department the contact works for (for example, Marketing).
- `title` — This holds the contact's title at the organization (such as Vice President or Senior Developer).

Saving a Contact

Once you've created a contact, you can save it, as shown here:

```
var myContact = navigator.contacts.create();
myContact.displayName = "Joe Smith";
myContact.gender = "male";
myContact.save(onSuccess, onFail);

function onSuccess(contact) {
    alert("Save Success");
}

function onError(contactError) {
    alert("Error = " + contactError.code);
}
```

Cloning a Contact

Cloning is an excellent way to make a copy of an existing contact so that you can then manipulate it as needed, as shown in the following example:

```
var myContact = navigator.contacts.create();
myContact.displayName = "Joe Smith";
myContact.gender = "male";

var myClone = myContact.clone();
myClone.displayName = "Tom Smith";
```

Removing a Contact

To delete a contact, use the `remove()` method like this:

```
var myContact = navigator.contacts.create();
myContact.displayName = "Joe Smith";
myContact.gender = "male";

contact.remove(onSuccess, onFail);

function onSuccess() {
    alert("Removal Success");
}

function onError(contactError) {
    alert("Error = " + contactError.code);
}
```

FINDING A CONTACT

Now that you know how to create your own contacts, let's take a look at finding contacts, which is another key function for several types of apps.

To run a search on the Contacts database, use the `contacts.find()` method, as shown here:

```
navigator.contacts.find(contactFields, onSuccess, onError, options);
```

This example shows the following:

- The first parameter passed to this method is a set of contact fields. These fields will be the ones that are returned. For example, if all you want is the person's name, e-mails, and phone numbers, you'd specify those fields here. If you only want the contact's id, pass in an empty parameter. If you want all of the fields returned, pass in `["*"]`.
- The second parameter is the `success` callback function, which will receive the appropriate `Contact` objects holding the contact information you were looking for.
- The third parameter is the `error` callback function, which will contain any errors thrown.

- The fourth parameter is a set of options you can pass to the `find` method. The one you'll use the most is the `filter` option, which is a string that can be used to filter the query to the `Contacts` database.

Following is an example of how you can find contacts:

```
function onSuccess(contacts) {
    document.write(contacts.length + ' contacts found.');
```

```
    for (var i=0; i<contacts.length; i++) {
        for (var j=0; j<contacts[i].emails.length; j++) {
            document.write("Email = " + contacts[i].emails[j].email);
        }
    }
}

function onError(contactError) {
    alert('Oops!');
}

// find all contacts with 'gmail' in any email
var myOptions = new ContactFindOptions();
myOptions.filter="gmail";
var myFields = ["emails"];
navigator.contacts.find(myFields, onSuccess, onError, myOptions);
```

UNDERSTANDING SOME QUIRKS

There are lots of quirks when it comes to working with contacts. Let's take a look at a few, organized by object.

Contacts

On Android 2.x devices, you'll encounter just one little quirk with contacts. The `categories` property is not supported on Android 2.x devices, and will always be returned as `null`.

On Android 1.x devices, you'll find that all of the following properties will return `null` because they are not supported:

- `name`
- `nickname`
- `birthday`
- `photos`
- `categories`
- `urls`

On BlackBerry WebWorks (OS 5.0 and higher) devices, you'll encounter the following quirks:

- `id` — This is assigned by the device when a contact is saved.
- `displayName` — This is stored in BlackBerry `user1` field.
- `nickname` — This is not supported, and will always be returned as `null`.
- `phoneNumbers` — This is partially supported. Phone numbers will be stored in BlackBerry fields `homePhone1` and `homePhone2` if type is 'home'; `workPhone1` and `workPhone2` if type is 'work'; `mobilePhone` if type is 'mobile'; `faxPhone` if type is 'fax'; `pagerPhone` if type is 'pager'; and `otherPhone` if type is none of the other types just described.
- `emails` — This is partially supported. The first three e-mail addresses will be stored in the BlackBerry `email1`, `email2`, and `email3` fields, respectively.
- `addresses` — This is partially supported. The first and second addresses will be stored in the BlackBerry `homeAddress` and `workAddress` fields, respectively.
- `ims` — This is not supported, and will always be returned as `null`.
- `organizations` — This is partially supported. The name and title of the first organization are stored in the BlackBerry `company` and `title` fields, respectively.
- `photos` — This is partially supported. A single thumbnail-sized photo is supported. To set a contact's photo, pass in either a Base64-encoded image, or a URL pointing to the image. The image will be scaled down before saving to the BlackBerry Contacts database. The contact photo is returned as a Base64-encoded image.
- `categories` — Only 'Business' and 'Personal' categories are supported.
- `urls` — The first URL is stored in the BlackBerry `webpage` field.

On iOS devices, you'll run into the following quirks:

- `displayName` — This property is not supported by iOS, and will be returned as `null` unless there is no `ContactName` specified. If there is no `ContactName`, a composite name, nickname, or "" is returned for `displayName`.
- `birthday` — For input, this property must be provided as a JavaScript `Date` object. It is returned as a JavaScript `Date` object.
- `photos` — A returned photo is stored in the application's temporary directory and a file URL for a photo is returned. The contents of the temporary folder are deleted when the application exits.
- `categories` — This is not currently supported, and will always be returned as `null`.

ContactName

On Android devices, note that the `formatted` property is partially supported. It will return the concatenation of `honorificPrefix`, `givenName`, `middleName`, `familyName`, and `honorificSuffix` but will not store the values.

On BlackBerry WebWorks (OS 5.0 and higher) devices, you should note the following:

- `formatted` — This is partially supported. It will return a concatenation of the BlackBerry `firstName` and `lastName` fields.
- `familyName` — This is supported and stored in the BlackBerry `lastName` field.
- `givenName` — This is supported and stored in the BlackBerry `firstName` field.
- `middleName` — This property is not supported, and will always return `null`.
- `honorificPrefix` — This property is not supported, and will always return `null`.
- `honorificSuffix` — This property is not supported, and will always return `null`.

On iOS devices, the `formatted` parameter is partially supported. It will return an iOS `Composite Name`, but will not store it.

ContactOrganization

On Android 2.x devices, the `pref` property is not supported and will always return `false`.

On Android 1.x devices, note the following:

- `pref` — This property is not supported by Android 1.x devices and will always return `false`.
- `type` — This property is not supported by Android 1.x devices and will always return `null`.
- `title` — This property is not supported by Android 1.x devices and will always be returned as `null`.

On BlackBerry WebWorks (OS 5.0 and higher) devices, note the following:

- `pref` — This property is not supported by Blackberry devices and will always return `false`.
- `type` — This property is not supported by Blackberry devices and will always return `null`.
- `name` — This property is partially supported. The first organization name will be stored in the BlackBerry `company` field.
- `department` — This property is not supported and will always be returned as `null`.
- `title` — This property is partially supported. The first organization title will be stored in the BlackBerry `jobTitle` field.

On iOS devices, note the following:

- `pref` — This property is not supported on iOS devices and will always return `false`.
- `type` — This property is not supported on iOS devices and will always return `null`.
- `name` — This property is partially supported. The first organization name will be stored in the iOS `kABPersonOrganizationProperty` field.
- `department` — This property is partially supported. The first department name will be stored in the iOS `kABPersonDepartmentProperty` field.
- `title` — This property is partially supported. The first title will be stored in the iOS `kABPersonJobTitleProperty` field.

HANDLING ERRORS

When an error occurs, a `Contact` error is returned to the error callback that you've created in your app. This object has one property, `code`, which is one of the following predefined codes:

- `ContactError.UNKNOWN_ERROR`
- `ContactError.INVALID_ARGUMENT_ERROR`
- `ContactError.TIMEOUT_ERROR`
- `ContactError.PENDING_OPERATION_ERROR`
- `ContactError.IO_ERROR`
- `ContactError.NOT_SUPPORTED_ERROR`
- `ContactError.PERMISSION_DENIED_ERROR`

TRY IT OUT Finding All Contacts at an Organization

Let's create a fairly simple application tries to find everyone in your `Contact` database who works at a certain organization. Enter the following code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Contact Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        var options = new ContactFindOptions();
        options.filter="Acme Corporation";
        filter = ["displayName","organizations"];
        navigator.contacts.find(filter, onSuccess, onError, options);
      }

      function onSuccess(contacts) {
        var myresults = document.getElementById("results");
        var string = "";
        for (var i=0; i<contacts.length; i++) {
          for (var j=0; j<contacts[i].organizations.length; j++) {
            string += "Name: " + contacts[i].displayName + "<br/>" +
              "Org Name: " + contacts[i].organizations[j].name + "\n" +
              "Department: " + contacts[i].organizations[j].department
              + "<br/>" +
              "Title: " + contacts[i].organizations[j].title);
          }
        }
      }
    </script>
  </head>
  <body>
    <div id="results">
    </div>
  </body>
</html>
```



```

    }

    myresults.innerHTML = string;
  }
  function onError(contactError) {
    alert('Oops!');
  }

  </script>
</head>
<body>
  <h1>Contacts</h1>
  <div id="results"></div>
</body>
</html>

```

How It Works

As soon as the app loads, it runs a search of the `Contacts` database, using the filter "Acme Corporation" on the `displayName` and `organization` fields. Once it retrieves the contact data, it loops through it, and then loops through the `organizations` object that is nested inside the result set. It then prints out the values stored in the object.

That's all you need to know about working with contacts.

SUMMARY

In this chapter, you learned how to work with the contacts, addresses, names, and organizations. You now know how to create new contacts, search for contacts, and work with the various contact fields. You also now know about the different quirks you'll encounter on some of the major devices.

Chapter 14 takes a look at how to capture video, audio, and images.

EXERCISES

1. Find all the contacts in your database who are doctors.
-

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
<code>contact.create()</code>	Use <code>contact.create()</code> to create a contact.
<code>contact.find()</code>	Use <code>contact.find()</code> to find a contact.

14

Capture

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Understanding how to capture video
- Understanding how to capture audio
- Understanding how to capture images
- Understanding how to handle errors

In earlier chapters, you learned how to use the `Media` and `Camera` objects to capture audio and photos, respectively. The PhoneGap API also lets you capture video, audio, and photos with the `Capture` API.

In this chapter, you learn how to use the `Capture` objects and methods.

LEARNING ABOUT CAPTURING VIDEO

To capture video on your device, use the `device.capture.captureVideo` method, as shown here:

```
navigator.device.capture.captureVideo(onSuccess, onError, options);
```

The success callback contains the captured video, and the error callback alerts you to any problems that you may encounter. Specifically, the success callback is invoked with an array of `MediaFile` objects, each describing a captured video clip file.

Using the options Argument

The options argument can contain the following items:

- **limit** — This is the maximum number of video clips the user can capture at a time. It defaults to 1, but must be greater than or equal to 1.
- **duration** — This is the maximum duration of the video clip (in seconds).
- **mode** — This is the selected video capture mode. The value must match a recognized video mode (video/quicktime, video/3gpp, and so on).



NOTE The 3GPP format (sometimes known as 3GP) is a multimedia format defined by the Third Generation Partnership Project.

For example, to capture a 30-second video, you'd do something like this:

```
var options = { duration: 30 };
navigator.device.capture.captureVideo(onSuccess, onError, options);
```

Once you've captured the videos, you probably want to do something interesting with them, so here's a closer look at the success callback:

```
function onSuccess(media_files) {
    var i, len;
    for (i = 0, len = media_files.length; i < len; i += 1) {
        processFiles(media_files[i]);
    }
}
function processFiles(media_file){
    //do something interesting here
}

// start video capture - grab 4 clips
navigator.device.capture.captureVideo(onSuccess,onError, {limit:4});
```

Recognizing Quirks When Capturing Videos

You should be aware of a variety of device-specific quirks when you are capturing videos.

On Android and BlackBerry devices, be aware of the following:

- The **duration** parameter is not supported. Recording lengths cannot be limited programmatically.
- The **mode** parameter is not supported. The video size and format cannot be altered programmatically. However, these parameters can be changed by the device user. By default, videos are recorded in 3GPP (video/3gpp) format.

On iOS devices, be aware of the following:

- The `limit` parameter is not supported. One video is recorded per invocation.
- The `duration` parameter is not supported. Recording lengths cannot be limited programmatically.
- The `mode` parameter is not supported. The video size and format cannot be altered programmatically. By default, videos are recorded in MOV (video/quicktime) format.



NOTE Apple developed the MOV format (files with the `.mov` extension) to work with multimedia files, and, in particular, QuickTime-wrapped files.

LEARNING ABOUT CAPTURING AUDIO

What's the difference between using the `Media` API and the `Capture` API for audio? Well, the `Capture` API enables you to capture more than one audio clip at a time, making it somewhat useful in different contexts.

Capturing audio is similar to capturing video, except that you use the `navigator.device.capture.captureAudio()` method instead. Following is an example:

```
navigator.device.capture.captureAudio(onSuccess, onError, options);
```

As with video, the success callback is invoked with an array of `MediaFile` objects that represent the audio file clips. You can process them in the same way you would video files, as shown here:

```
function onSuccess(media_files) {
    var i, len;
    for (i = 0, len = media_files.length; i < len; i += 1) {
        processFiles(media_files[i]);
    }
}

function processFiles(media_file){
    //do something interesting here
}

// start video capture - capture 2 clips
navigator.device.capture.captureAudio(onSuccess,onError, {limit:2});
```

Using the options Argument

The `options` argument supports these parameters:

- `limit` — This is the number of audio clips to capture. Default is 1.
- `duration` — This is the maximum duration of audio clips (in seconds).
- `mode` — This is the selected audio mode (such as `audio/wav` or `audio/amr`).

Recognizing Quirks When Capturing Audio

Again, you should be aware of a variety of device-specific quirks when you are capturing audio.

On Android and BlackBerry devices, be aware of the following:

- The `duration` parameter is not supported. Recording lengths cannot be limited programmatically.
- The `mode` parameter is not supported. The audio recording format cannot be altered programmatically. Recordings are encoded using Adaptive Multi-Rate (AMR) format (`audio/amr`).

On iOS devices, be aware of the following:

- The `limit` parameter is not supported. One recording can be created for each invocation.
- The `mode` parameter is not supported. The audio recording format cannot be altered programmatically. Recordings are encoded using Waveform Audio (WAV) format (`audio/wav`).

LEARNING ABOUT CAPTURING IMAGES

You already know how to use the `Camera` API to either take a picture with the default camera application, or choose a photo, for example, from the Photo Gallery. The `Capture` API enables you to create an application that will take numerous image captures and then save them.

Capturing an image is simple. Just use the `navigator.device.capture.captureImage()` method, as shown here:

```
navigator.device.capture.captureImage(onSuccess, onError, options);
```

And, just like with audio and video capture, you can then do something useful with the array of `MediaFile` objects that result from the success callback function, as shown here:

```
function onSuccess(media_files) {
    var i, len;
    for (i = 0, len = media_files.length; i < len; i += 1) {
        processFiles(media_files[i]);
    }
}

function processFiles(media_file){
    //do something interesting here
}

// start video capture - grab 2 clips
navigator.device.capture.captureImage(onSuccess,onError, {limit:2});
```

Using the options Argument

The `options` argument supports these parameters:

- `limit` — This is the number of images to capture. The default is 1.
- `mode` — This is the selected image mode (such as `image/jpeg`).

Recognizing Quirks When Capturing Images

You should be aware of a variety of device-specific quirks when you are capturing images.

On Android and BlackBerry devices, be aware of the following:

- The `mode` parameter is not supported. The image size and format cannot be altered programmatically; however, the image size can be altered by the device user. Images are saved in JPEG format (`image/jpeg`).

On iOS devices, be aware of the following:

- The `limit` parameter is not supported. One image is taken per invocation.
- The `mode` parameter is not supported. The image size and format cannot be altered programmatically. Images are saved in JPEG format (`image/jpeg`).

HANDLING ERRORS

The error callback is typically invoked if the following occurs:

- The capture application is busy at the time of invocation.
- Something has gone wrong during the capture operation.
- The user cancels the capture operation.

When an error does occur, a `CaptureError` object is invoked that returns an error code. For example:

```
function onError(error){
  alert('Error code: ' + error.code);
}
```

TRY IT OUT Capture Photo and Upload

For this exercise, let's create a straightforward application that captures an image and then uploads it to a remote server. Enter the following:



```
<!DOCTYPE html>
<html>
  <head>
    <title>Capture Image</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
```

```
<script type="text/javascript" charset="utf-8">

function onSuccess(mediaFiles) {
    var i, len;
    for (i = 0, len = mediaFiles.length; i < len; i += 1) {
        uploadFile(mediaFiles[i]);
    }
}

function onError(error) {
    var msg = 'An error occurred during capture: ' + error.code;
    navigator.notification.alert(msg, null, 'Uh oh!');
}

function captureImage() {
    navigator.device.capture.captureImage(onSuccess, onError, {limit: 2});
}

// Upload files to server
function uploadFile(mediaFile) {
    var status = document.getElementById("status");
    var ft = new FileTransfer(),
        path = mediaFile.fullPath,
        name = mediaFile.name;

    ft.upload(path,
        "http://example.com/upload.php",
        function(result) {

            status.innerHTML = 'Upload success: ' + result.responseCode + '.  

                Bytes sent: ' + result.bytesSent;
        },
        function(error) {
            status.innerHTML = 'Error uploading file ' + path + ':  

                ' + error.code;
        },
        { fileName: name });
}

</script>
</head>
<body>
    <button onclick="captureImage();">Capture Image</button> <br>
    <div id='status'></div>
</body>
</html>
```

Code file [chapter14.html] available for download at Wrox.com.

How It Works

As soon as the app loads, the user can click the Capture Image button. This process enables the user to capture up to two images before the files are uploaded to a remote server using the `uploadFile()` function. Once sent, the user sees a status update showing the confirmation code and the number of bytes sent.

That's all you need to know about working with the Capture API.

SUMMARY

In this chapter, you learned how to work with audio, video, and image capture tools in the PhoneGap Capture API.

In Chapter 15, you're going to take a lot of what you've learned in the first 14 chapters of the book and create a smartphone application.

EXERCISES

1. Create an app that allows audio, video, or image capture options.
-

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
<code>navigator.device.capture.captureAudio()</code>	Use <code>navigator.device.capture.captureAudio()</code> to capture audio.
<code>navigator.device.capture.captureImage()</code>	Use <code>navigator.device.capture.captureImage()</code> to capture image.
<code>navigator.device.capture.captureVideo()</code>	Use <code>navigator.device.capture.captureVideo()</code> to capture video.

15

Creating a Note-Taking Application

WHAT YOU WILL LEARN IN THIS CHAPTER:

- Designing the application
- Building the application

Throughout the previous 14 chapters, you've learned a great deal about PhoneGap and what it can do. You've successfully navigated information about the device; figured out how to use the compass, accelerometer, and geolocation services; learned how to use the camera, video camera, and audio recorder to capture information; and learned how to store that information in the database. Furthermore, you learned about the filesystem, and can work with the contacts database.

In this chapter, you apply what you've learned to create a simple (but useful) note-taking application. Before doing anything else, though, let's first spend some time outlining what this application should do.

DESIGNING THE APPLICATION

The note-taking application you create in this chapter is pretty straightforward, but it has a lot of moving parts that need to work together. The idea behind the application is to be able to capture information in whatever format (photographically, for example, or with an audio recorder), and then send the captured information somewhere for later retrieval.

From the user's perspective, the note taking might involve the following:

- Taking a picture of something.
- Recording a snippet of audio.

- Recording some video.
- Writing down some notes using the device's keyboard.

Once a note has been taken, the user should be allowed to add some metadata (a title and description would be good). Furthermore, the application should probably add its own metadata behind the scenes (timestamp, geolocation data, maybe more). Finally, the application should then store the information locally and back it up to a remote server.

You should capture audio, video, and image data with the `Capture` API because it provides a fairly easy and intuitive way to do this. As you learned in Chapter 14, it's simple for a developer to offer a whole bunch of functionality without too much effort.

You should capture text with a simple text field that uses the default keyboard on the device. You probably want to allow for some flexibility in terms of note length, but in all reality, it just isn't that easy to tap away on a small phone as compared to making an audio or video recording.

Because users have four options for recording notes, the first screen should probably just show a series of buttons that allow them to capture their notes in a certain format, or review a list of previously saved notes.

Once the capture is complete, the next screen should prompt them for a title and description, and then automatically add a timestamp and latitude/longitude information.

Finally, after that information has been entered, the application should store the note data in a local database, and then transmit the information up to a remote server for safekeeping.

So, to summarize, this app uses various parts of the PhoneGap API:

- `Capture`
- `Geolocation`
- `Storage`
- `Files`

BUILDING THE APPLICATION

Building an application of this size requires breaking it down into three or four smaller modules:

- Capture options
- Adding metadata
- Saving/synching
- Adding geolocation

Creating the Capture Options

In a lot of ways, the first screen you'll write for the note-taking application is very similar to the basic capture application you created in Chapter 14.

The idea is to present the users with four buttons that enable them to capture video, audio, images, or text. You add the four buttons and invoke the `capture()` function with each, passing in the type of capture you want (image, audio, video, or text). Because there is no text capture in PhoneGap, what this does is show the previously hidden `notes` DOM element. Also, note that when you deal with media files, you'll be using an array to retrieve results, so plan on looping through that array to get at information.

Here's the code so far:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Note Taker</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

function onSuccess(mediaFiles) {
    var i, len;
    for (i = 0, len = mediaFiles.length; i < len; i += 1) {
        addMetadata(mediaFiles[i]);
    }
}

function onError(error) {
    var msg = 'An error occurred during capture: ' + error.code;
    alert(msg);
}

function capture(type) {
    if (type == "image"){
        navigator.device.capture.captureImage(onSuccess, onError, {limit: 1});
    }else if (type == "audio"){
        navigator.device.capture.captureAudio(onSuccess, onError, {limit: 1});
    }else if (type == "video"){
        navigator.device.capture.captureVideo(onSuccess, onError, {limit: 1});
    }else{
        document.getElementById('notes').style.display = 'block';
    }
}

// Add Metadata
function addMetadata(mediaFile) {
}

</script>
</head>
<body>
    <button onclick="capture('image');">Capture Image</button> <br>
    <button onclick="capture('audio');">Capture Audio</button> <br>
    <button onclick="capture('video');">Capture Video</button> <br>
    <button onclick="capture('text');">Write Text</button> <br>

    <div id='notes' style='display:none'>
        <p>Write notes:<br/>
```

```

        <textarea id='text_notes' cols='20' rows='5'></textarea>
    </p>
</div>
</body>
</html>

```

So far, the interface looks a bit like Figure 15-1. In the screenshot, the Write Text button has been tapped to display the text field.



FIGURE 15-1: Displaying the text field

Of course, if you wanted to add jQuery to the mix, you could simplify your life a little bit, using the `show()` method, as shown here:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Note Taker</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8" src="jquery.js"></script>
    <script type="text/javascript" charset="utf-8">

    function onSuccess(mediaFiles) {
      var i, len;
      for (i = 0, len = mediaFiles.length; i < len; i += 1) {
        showMetadata(mediaFiles[i]);
      }
    }

    function onError(error) {

```

```

        var msg = 'An error occurred during capture: ' + error.code;
        alert(msg);
    }

    function capture(type) {
        if (type == "image"){
            navigator.device.capture.captureImage(onSuccess, onError, {limit: 1});
        }else if (type == "audio"){
            navigator.device.capture.captureAudio(onSuccess, onError, {limit: 1});
        }else if (type == "video"){
            navigator.device.capture.captureVideo(onSuccess, onError, {limit: 1});
        }else{
            $('#notes').show();
        }
    }
}

// Show Metadata
function showMetadata(mediaFile) {
}

</script>
</head>
<body>
    <button onclick="capture('image');">Capture Image</button> <br>
    <button onclick="capture('audio');">Capture Audio</button> <br>
    <button onclick="capture('video');">Capture Video</button> <br>
    <button onclick="capture('text');">Write Text</button> <br>

    <div id='notes' style='display:none'>
        <p>Write notes:<br/>
        <textarea id='text_notes' cols='20' rows='5'></textarea>
        </p>
    </div>
</body>
</html>

```

Adding jQuery is a good idea, not only because it simplifies the life of the developer, but makes for a more consistent user experience for the user of the application. (You can use one function to show a DOM element and not worry about it working on different devices.) If you have less code to worry about, you have less that could possibly go wrong for the user. It's your choice.

Another thing you can do is perhaps make the text area a little wider and taller by adjusting the `cols` and `rows` attributes. That's all cosmetics, but once you see it for real, it's an easy thing to fix. Figure 15-2 shows a much roomier text field.



FIGURE 15-2: Changing the size of the text area

Adding Metadata

So, now you have a situation in which the user could be capturing video/audio/images (and, therefore, ending up inside a success callback function), or jotting some notes down in a text field and then having to tap a button to continue.

Either way, you want the users to end up in the same place: a screen that allows them to add a title and description to the note they’ve just taken. Because you want to keep this all on one page, you’re going to need to go back a bit and think about the components required to make this happen.

The first thing you’ll want to do is wrap your buttons in a DOM element with a unique ID, like this:

```
<div id="buttons">
  <button onclick="capture('image');">Capture Image</button> <br>
  <button onclick="capture('audio');">Capture Audio</button> <br>
  <button onclick="capture('video');">Capture Video</button> <br>
  <button onclick="capture('text');">Write Text</button> <br>
</div>
```

The reasoning behind this is that once you’ve captured some video (for example), or written a text note, the application hides this DOM element so that the user can then add a title and description.

While you’re at it, add that bit to the HTML as well — just a simple form that lets a user enter a title and description:

```
<div id='metadata' style="display:none">
  <p>Title<br/>
  <input type='text' id='title'>
</p>

  <p>Description:<br/>
  <textarea id='description' cols='40' rows='15'></textarea>
</p>
  <button onClick="synch();">save</button>

</div>
```

And, because you have a “save” button on this section, add a “save” button to the previous notes element, as shown here:

```
<div id='notes' style='display:none'>
  <p>Write notes:<br/>
  <textarea id='text_notes' cols='40' rows='15'></textarea>
</p>
  <button onClick="saveNote();">save</button>
</div>
```

Next comes the part where you write the `saveNote()` function. Essentially, what this should do is take you to the same place that the success callback takes you: where you can see the metadata save

form. Oh, yes, you also have to hide the notes and buttons! Note that jQuery is used here, mostly out of convenience.

```
function saveNote(){
    $("#metadata").show();
    $("#buttons").hide();
    $("#notes").hide();
}
```

Figure 15-3 shows the progress thus far.

If the user has opted to capture video, audio, or image data, he or she will have activated the success callback, which invokes the `MediaFile` object and the `showMetadata()` function. Part of this function will be very similar to the `saveNote()` function (that is, show the metadata form, and hide the others). But you must also save the `MediaFile` object's name and full path to the end of the form. Again, jQuery comes to the rescue, enabling you to quickly and easily append to hidden form fields that contain the filename and the file path for later reference, as shown here:

```
function showMetadata(mediaFile) {
    path = mediaFile.fullPath;
    name = mediaFile.name;
    $("#metadata").append("<input type='hidden' id='filename'
        value='"+name+"'>");
    $("#metadata").append("<input type='hidden' id='filepath'
        value='"+path+"'>");
    $("#metadata").show();
    $("#buttons").hide();
    $("#notes").hide();
}
```

At this point, all that's left to do is to synch your information up to the server.

Saving and Synching

In the `metadata` DOM element, there is a button labeled “save” that fires the `synch()` function. You haven't written that function, but what it needs to do is gather the title, description, the text note (if it exists), the filename, and file path (if it exists), and then send it up to the remote server.

Now, for the sake of this discussion, let's make a bunch of assumptions here. First, let's assume that you're running PHP on your server.



FIGURE 15-3: Screen that includes fields for a title and description provided by the user



NOTE The key to this assumption is that you know your way around whatever language you are using.

Next, let's assume that you'll write a script that takes the file you're transferring and deals with the metadata. Each language has its own way of dealing with `POST` data, so the details will be left to you.

So, in effect, when the user taps that final “save” button, you must grab the file and transport it to the server, along with all the metadata that's been saved.

First things first — you must create a database and then put a table into that database to hold your information. The best place to do that is at the very top of the application, as shown here:

```
var db = window.openDatabase("notes", "1.0", "Notes DB", 1000000);
db.transaction(createDB, onErrorDB, onSuccessDB);
```

In turn, the `createDB()` function instantiates the tables, and the two callbacks keep you apprised of what's happening:

```
function createDB(tx) {
    tx.executeSql('DROP TABLE IF EXISTS notes');
    tx.executeSql('CREATE TABLE IF NOT EXISTS notes (id primary key,
        title varchar, description varchar, filepath text,
        note_text text)');
}
function onErrorDB(err) {
    $("#status").append("Error processing SQL: "+err.code+"<br/>");
}
function onSuccessDB() {
    $("#status").append("Saved to DB!<br/>");
}
```

For those status messages to show up, you'll need to add a DOM element with an `id` of `status`:

```
<div id='status'></div>
```

Now, let's go back to the `synch()` function, which is triggered when the final “save” button is tapped by the user. The `synch()` function must read what has been saved in the different form fields, save that information to the local database, and then use the file-transfer mechanism to push the file to the remote server as a backup. After that, it must show the original buttons again and hide all the rest of the DOM elements.

Here's the `synch()` function:

```
function synch(){
    var ft = new FileTransfer();
    var path = $("#filepath").val();
    var name = $("#filename").val();
    db.transaction(insertRow, onErrorDB, onSuccessDB);

    ft.upload(path,
        "http://example.com/upload.php",
        function(result) {
```

```

        $("#status").append('Upload success: ' + result.responseCode + '<br/>');
        $("#status").append(result.bytesSent + ' bytes sent<br/>');
    },
    function(error) {
        $("#status").append('Error uploading file ' + path + ': ' +
            error.code + '<br/>');
    },
    { fileName: name });

$("#metadata").hide();
$("#buttons").show();
$("#notes").hide();
}

```

This code fires off another database transaction, the `insertRow()` function, which looks like this:

```

function insertRow(tx){
    var path = $("#filepath").val();
    var title = $("#title").val();
    var desc = $("#description").val();
    var notes = $("#text_notes").val();
    var randomid = randomUUID();
    tx.executeSql('INSERT INTO notes
        (id,title,description,filepath,note_text)
        VALUES (''+randomid+'',''+title+'',
            ''+desc+'',''+path+'',''+notes+'')');
    alert('INSERT INTO notes (id,title,description,filepath,note_text)
        VALUES (''+randomid+'',''+title+'',
            ''+desc+'',''+path+'',''+notes+'')');
}

```

There's nothing too special here. You're taking the values out of the form fields, storing them in variables, and then pushing those variables into the SQL statement being executed. Of course, there's one more thing to look at there, the `randomUUID()` function:

```

function randomUUID() {
    var s = [], itoh = '0123456789ABCDEF';

    for (var i = 0; i < 32; i++) {
        s[i] = Math.floor(Math.random()*0x10);
    }

    // Conform to RFC-4122, section 4.4
    s[14] = 4; // Set 4 high bits of time_high field to version
    s[19] = (s[19] & 0x3) | 0x8; // Specify 2 high bits of clock sequence

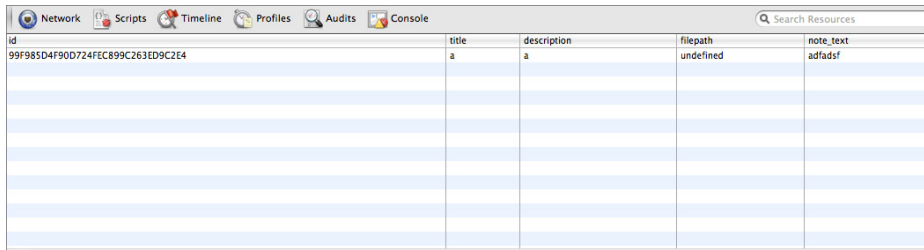
    // Convert to hex chars
    for (var i = 0; i < 36; i++) s[i] = itoh[s[i]];

    return s.join('');
}

```

This is a fairly simple function that creates a 32-digit alphanumeric ID, and it should continue to generate unique strings like that for a good long time.

If you run your app in an HTML5 browser like Chrome, and then look at the Resources tab, you'll see that each time you write a note or save an audio file, you end up with a new record in your database. Figure 15-4 shows what you'd see.



The screenshot shows a web browser's developer tools interface with the 'Resources' tab selected. It displays a table with five columns: 'id', 'title', 'description', 'filepath', and 'note_text'. The first row contains the following data: '99f985d4f90d724fec899c263ed9c2e4', 'a', 'a', 'undefined', and 'adfadsf'. The table has alternating light blue and white rows for its data entries.

id	title	description	filepath	note_text
99f985d4f90d724fec899c263ed9c2e4	a	a	undefined	adfadsf

FIGURE 15-4: View of the application from an HTML5 browser

Adding Geolocation

There is one more thing to do. You must add some geolocation information to the notes you take. The easiest way is to add a call to the geolocation services at the top of the code, and then add a hidden field for latitude/longitude data inside the metadata DOM element:

```
var onGeoSuccess = function(position) {
    $("#metadata").append("<input type='hidden' id='geo'
        value='"+position.coords.latitude+"', "+
        position.coords.longitude+" '>");
};
function onGeoError(error) {
    $("#status").append('Geolocation error: ' + error.code + ' ' +
        'message: ' + error.message + '<br/>');
}

navigator.geolocation.getCurrentPosition(onGeoSuccess, onGeoError);
```

Once you have geolocation information stored there, you have to remember to grab it and put it in the database. This means, however, that you'll want to add a geolocation field to the database, as shown here:

```
function createDB(tx) {
    tx.executeSql('DROP TABLE IF EXISTS notes');
    tx.executeSql('CREATE TABLE IF NOT EXISTS notes (id primary key,
        title varchar, description varchar, filepath text, note_text
        text, geolocation text)');
}
```

Now you can just pick up the geolocation information from the form and dump it to the database, as shown here:

```

function insertRow(tx){
    var path = $("#filepath").val();
    var title = $("#title").val();
    var desc = $("#description").val();
    var notes = $("#text_notes").val();
    var geo = $("#geo").val();
    var randomid = randomUUID();
    tx.executeSql('INSERT INTO notes
        (id,title,description,filepath,note_text,geolocation) VALUES
        (''+randomid+'',''+title+'', '''+desc+'',''+path+'',
        '''+notes+'', '''+geo+'')');
}

```

The Final Code

Here's the entire app in one code listing:

```

<!DOCTYPE html>
<html>
<head>
    <title>Note Taker</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8" src="jqtouch/jquery-
        1.4.2.js"></script>
    <script type="text/javascript" charset="utf-8">

        var db = window.openDatabase("notes", "1.0", "Notes DB", 1000000);
        db.transaction(createDB, onErrorDB, onSuccessDB);

        var onGeoSuccess = function(position) {
            $("#metadata").append("<input type='hidden' id='geo'
                value='"+position.coords.latitude+"','"+
                position.coords.longitude+"'>");
        };

        function onGeoError(error) {
            $("#status").append('Geolocation error: ' + error.code + ' ' +
                'message: ' + error.message + '<br/>');
        }

        navigator.geolocation.getCurrentPosition(onGeoSuccess, onGeoError);

        function onSuccess(mediaFiles) {
            var i, len;
            for (i = 0, len = mediaFiles.length; i < len; i += 1) {
                showMetadata(mediaFiles[i]);
            }
        }

        function onError(error) {
            var msg = 'An error occurred during capture: ' + error.code;

```

```
        alert(msg, null, 'Uh oh!');
    }

function capture(type) {
    if (type == "image"){
        navigator.device.capture.captureImage(onSuccess, onError, {limit: 1});
    }else if (type == "audio"){
        navigator.device.capture.captureAudio(onSuccess, onError, {limit: 1});
    }else if (type == "video"){
        navigator.device.capture.captureVideo(onSuccess, onError, {limit: 1});
    }else{
        $('#notes').show();
    }
}

function saveNote(){
    $('#metadata').show();
    $('#buttons').hide();
    $('#notes').hide();
}

// Show Metadata
function showMetadata(mediaFile) {
    path = mediaFile.fullPath,
    name = mediaFile.name;
    $('#metadata').append("<input type='hidden' id='filename'
        value='"+name+"'>");
    $('#metadata').append("<input type='hidden' id='filepath'
        value='"+path+"'>");
    $('#metadata').show();
    $('#buttons').hide();
    $('#notes').hide();
}

function synch(){
    var ft = new FileTransfer();
    var path = $("#filepath").val();
    var name = $("#filename").val();
    db.transaction(insertRow, onErrorDB, onSuccessDB);

    ft.upload(path,
        "http://example.com/upload.php",
        function(result) {
            $('#status').append('Upload success: ' + result.responseCode +
                '<br/>');
            $('#status').append(result.bytesSent + ' bytes sent');
        },
        function(error) {
            $('#status').append('Error uploading file ' + path + ': ' +
                error.code);
        },
        { fileName: name });

    $('#metadata').hide();
    $('#buttons').show();
}
```

```

        $("#notes").hide();
    }

    function createdB(tx) {
        tx.executeSql('DROP TABLE IF EXISTS notes');
        tx.executeSql('CREATE TABLE IF NOT EXISTS notes (id primary key,
            title varchar, description varchar, filepath text,
            note_text text, geolocation text)');
    }

    function insertRow(tx) {
        var path = $("#filepath").val();
        var title = $("#title").val();
        var desc = $("#description").val();
        var notes = $("#text_notes").val();
        var geo = $("#geo").val();
        var randomid = randomUUID();
        tx.executeSql('INSERT INTO notes (id,title,description,
            filepath,note_text,geolocation) VALUES
            ("'+randomid+'", "' +title+'", "' +desc+'", "' +path+'",
            "' +notes+'", "' +geo+'")');
    }

    function onErrorDB(err) {
        $("#status").append("Error processing SQL: "+err.code+"<br/>");
    }

    function onSuccessDB() {
        $("#status").append("Saved to DB!<br/>");
    }

    function randomUUID() {
        var s = [], itoh = '0123456789ABCDEF';

        for (var i = 0; i <32; i++) s[i] = Math.floor(Math.random()*0x10);

        // Conform to RFC-4122, section 4.4
        s[14] = 4; // Set 4 high bits of time_high field to version
        s[19] = (s[19] & 0x3) | 0x8; // Specify 2 high bits of clock sequence

        // Convert to hex chars
        for (var i = 0; i <36; i++) s[i] = itoh[s[i]];

        return s.join('');
    }
}

</script>
</head>
<body>
    <div id="buttons">
        <button onclick="capture('image');">Capture Image</button> <br>
        <button onclick="capture('audio');">Capture Audio</button> <br>
        <button onclick="capture('video');">Capture Video</button> <br>
    </div>

```

```
        <button onclick="capture('text');">Write Text</button> <br>
    </div>
    <div id='notes' style='display:none'>
        <p>Write notes:<br/>
        <textarea id='text_notes' cols='40' rows='15'></textarea>
        </p>
        <button onClick="saveNote();">save</button>
    </div>

    <div id='metadata' style="display:none">
    <p>Title<br/>
    <input type='text' id='title'>
    </p>

    <p>Description:<br/>
    <textarea id='description' cols='40' rows='15'></textarea>
    </p>
    <button onClick="synch();">save</button>

    </div>

    <div id='status'></div>
</body>
</html>
```

CLEANING UP THE APP

So far, this chapter has focused more on creating the app than worrying about how pretty the code looks. Admittedly, when you look at the final code listing, it looks like a mess. Even though it's only about 160 lines of code, it could prove to be much more difficult to work with in the future.

For your future sanity, you might want to try to look at ways of breaking up this app. An easy way to do this is to sweep all the JavaScript into its own file, and then load that — this would be something akin to a kid just throwing all his toys into the closet and calling it a day.

Another, smarter, approach would be to put all your database code in one file (including the success/error callbacks), all the geolocation code in one file, and so on. This will help you understand what your app is doing, and make it easier to maintain in the long run — particularly if you decide to beef it up with more look and feel.

This discussion has purposely not addressed the look and feel of this sample application. You can use jQTouch, jQuery Mobile CSS, or even SenchaTouch to give it more polish, or you can use your own CSS design skills to give it more punch.

SUMMARY

In this chapter, you learned how to create an application from scratch. This application enables users to capture notes in different formats, save that information to the device, and then transmit it to a remote server.

► WHAT YOU LEARNED IN THIS CHAPTER

TOPIC	KEY CONCEPTS
<code>navigator.geolocation.getCurrentPosition()</code>	Use <code>navigator.geolocation.getCurrentPosition()</code> to get the current position.
<code>window.openDatabase()</code>	Use <code>window.openDatabase()</code> to open a database.
<code>tx.executeSql()</code>	Use <code>tx.executeSql()</code> to execute SQL transactions.
<code>FileTransfer</code>	Use the <code>FileTransfer</code> object to transfer files to a remote server.
<code>navigator.device.capture.captureImage()</code>	Use <code>navigator.device.capture.captureImage()</code> to capture images.
<code>navigator.device.capture.captureAudio()</code>	Use <code>navigator.device.capture.captureAudio()</code> to capture audio.
<code>navigator.device.capture.captureVideo()</code>	Use <code>navigator.device.capture.captureVideo()</code> to capture video.



Answers to Exercises

This appendix provides the answers to Exercises found in chapters throughout this book.

CHAPTER 4 EXERCISE ANSWERS

Following are the answers to the exercises in Chapter 4.

Answer to Question 1

The following code snippet shows how you can open the Google homepage as soon as you detect that the PhoneGap application is online:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap Google Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        // Register the event listeners
        document.addEventListener("online", isOnline, false);
      }

      function isOnline() {
        window.location = "http://www.google.com";
      }

    </script>
```

```
</head>
<body>
</body>
</html>
```

Answer to Question 2

The following code snippet shows how you can display the current time when the device is ready:

```
<!DOCTYPE html>
<html>
  <head>
    <title>PhoneGap Google Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {
        showDateTime();
      }

      function showDateTime() {
        var currentTime = new Date();
        var hours = currentTime.getHours();
        var minutes = currentTime.getMinutes();

        var target = document.getElementById("time");
        target.innerHTML("<b>" + hours + ":" + minutes + "</b>");
      }

    </script>
  </head>
  <body>
    <div id="time"></div>
  </body>
</html>
```

CHAPTER 5 EXERCISE ANSWERS

Following are the answers to the exercises in Chapter 5.

Answer to Question 1

Following is an example of a simple HTML form that allows a user to enter a web address and then press a button to test the network connectivity:

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

```

<html>
  <head>
    <title>Is the Network Reachable?</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      // Wait for PhoneGap to load
      //
      function onLoad() {
        document.addEventListener("deviceready", onDeviceReady, false);
      }

      // PhoneGap is loaded and it is now safe to make calls
      //
      function onDeviceReady() {
        //leave empty
      }

      // Check network status
      //

      function testNetwork(myValue){
        navigator.network.isReachable(myValue, myCallback, {});
      }
      function myCallback(reachability) {
        var networkState = reachability.code || reachability;

        var states = {};
        states[NetworkStatus.NOT_REACHABLE] = 'No network connection';
        states[NetworkStatus.REACHABLE_VIA_CARRIER_DATA_NETWORK] = 'Cellular
          data connection';
        states[NetworkStatus.REACHABLE_VIA_WIFI_NETWORK] = 'WiFi connection';

        alert('Connection type: ' + states[networkState]);
      }

    </script>
  </head>
  <body onload="onLoad()">
    <label>Test the network:</label><br/>
    <input type='text' id='txt1' value='google.com' />
    <input type='submit' value='submit'
      onClick='testNetwork(document.getElementById('txt1').value)' />
  </body>
</html>

```

Answer to Question 2

Following is an example of an informational footer that contains information about the device the person is using:

```

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

```

```
<html>
<head>
  <title>My Device</title>

  <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
  <script type="text/javascript" charset="utf-8">

    // Use an event listener to detect if PhoneGap is ready
    //
    function onLoad() {
      document.addEventListener("deviceready", onDeviceReady, false);
    }

    // okay, PhoneGap is ready
    //
    function onDeviceReady() {
      var myDiv = document.getElementById('footer_text');

      myDiv.innerHTML =   device.name      + ' | ' +
                          device.phonegap + ' | ' +
                          device.platform + ' | ' +
                          device.uuid      + ' | ' +
                          device.version;
    }

    function showHide(){
      var myDiv = document.getElementById('footer_text');
      var myButton = document.getElementById('button');

      if(myDiv.style.display == "block") {
        myDiv.style.display = "none";
        button.innerHTML = "show";
      }
      else {
        myDiv.style.display = "block";
        button.innerHTML = "hide";
      }
    }

  </script>

  <style>
  #footer_text{
    font-size:10px;
    float:left;
    color:#000;
    background-color:#fff;
  }

  button{
    float:left;
    color:#ccc;
    background-color:#fff;
    width:40px;
    border:1px solid #ccc;
```

```

    }
  </style>
</head>
<body onload="onLoad()">
<div id='footer'>
  <div id='footer_text'></div>
  <button id='button' onClick="showHide()">hide</button>
</div>
</body>
</html>

```

Answer to Question 3

Following is an example of a confirmation dialog that prompts the user to push one button to ring a bell, and another button for vibrate:

```

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

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      // Wait for PhoneGap to load
      //
      function onLoad() {
        document.addEventListener("deviceready", onDeviceReady, false);
      }

      // PhoneGap is ready
      //
      function onDeviceReady() {
        // Empty
      }

      // process the confirmation dialog result
      function onConfirm(button) {
        if (button == 'Beep'){
          playBeep();
        }else{
          vibrate();
        }
      }

      // Show a custom confirmation dialog
      //
      function showConfirm() {
        navigator.notification.confirm(
          'Beep or Vibrate?', // message
          onConfirm,          // callback
          'Beep/Vibrate',     // title

```

```
        'Beep,Vibrate' // buttonLabels
    );
}

// Beep twice
//
function playBeep() {
    navigator.notification.beep(2);
}

// Vibrate for 4 seconds
//
function vibrate() {
    navigator.notification.vibrate(4000);
}

</script>
</head>
<body onload="onLoad()" >
    <p><a href="#" onclick="showConfirm(); return false;">Show
        Confirmation</a></p>
</body>
</html>
```

CHAPTER 6 EXERCISE ANSWER

Following is the answer to the exercise in Chapter 6.

Answer to Question 1

Following is some example code to create a beep if the device is held at a certain angle:

```
<!DOCTYPE html>
<html>
<head>
    <title>Acceleration Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

var watchID = null;

// Wait for PhoneGap to load
document.addEventListener("deviceready", onDeviceReady, false);

// PhoneGap is ready, start watching
function onDeviceReady() {
    startWatch();
}

// Start watching the acceleration
function startWatch() {
```



```

        // Update acceleration every 3 seconds
        var options = { frequency: 3000 };

        watchID = navigator.accelerometer.watchAcceleration(onSuccess, onError,
            options);
    }

    // Stop watching the acceleration
    function stopWatch() {
        if (watchID) {
            navigator.accelerometer.clearWatch(watchID);
            watchID = null;
        }
    }

    // onSuccess: Get a snapshot of the current acceleration
    function onSuccess(acceleration) {

        if ((acceleration.x > .5 && acceleration.x < .8) && acceleration.z > 3){
            navigator.notification.beep(1);
        }

    }

    // onError: Failed to get the acceleration
    //
    function onError() {
        alert('oooops!');
    }

</script>
</head>
<body>
    <div id="accelerometer">Waiting for accelerometer...</div>
    <button onclick="stopWatch();">Stop Watching</button>
</body>
</html>

```

CHAPTER 7 EXERCISE ANSWER

Following is the answer to the exercise in Chapter 7.

Answer to Question 1

Following is an example of how to display a special message if the device is oriented north, or not display a special message if it is not oriented north:

```

<!DOCTYPE html>
<html>
  <head>

```

```
<title>Compass Example</title>

<script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
<script type="text/javascript" charset="utf-8">

var watchID = null;
document.addEventListener("deviceready", onDeviceReady, false);

// PhoneGap is ready
function onDeviceReady() {
    startWatch();
}

// Start watching the compass
function startWatch() {

    // Update compass every 2 seconds
    var options = { frequency: 2000 };

    watchID = navigator.compass.watchHeading(onSuccess, onError, options);
}

// Stop watching the compass
function stopWatch() {
    if (watchID) {
        navigator.compass.clearWatch(watchID);
        watchID = null;
    }
}

// onSuccess: Get the current heading
function onSuccess(heading) {
    var heading = document.getElementById('heading');
    heading.innerHTML = 'Heading: ' + heading;
}

var special = document.getElementById('special');
if (heading == 0){
    heading.innerHTML = '<b>NORTH!</b>';
}else{
    heading.innerHTML = '';
}

}

// onError: Failed to get the heading
function onError() {
    alert('oooops!');
}

</script>
</head>
<body>
<div id="heading">Waiting for heading...</div>
<div id="special"></div>
```

```

        <button onclick="startWatch();">Start Watching</button>
        <button onclick="stopWatch();">Stop Watching</button>
    </body>
</html>

```

CHAPTER 8 EXERCISE ANSWERS

Following are the answers to the exercises in Chapter 8.

Answer to Question 1

Following is an example of adding a Stop Watching button to a geolocation application:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Device Properties Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      // Wait for PhoneGap to load
      //
      document.addEventListener("deviceready", onDeviceReady, false);

      var watchID = null;

      // PhoneGap is ready
      //
      function onDeviceReady() {
        // Update every 3 seconds
        var options = { frequency: 3000 };
        watchID = navigator.geolocation.watchPosition(onSuccess, onError, options);
      }

      // onSuccess Geolocation
      //
      function onSuccess(position) {
        var element = document.getElementById('geolocation');
        element.innerHTML = 'Latitude: ' + position.coords.latitude + '<br />' +
          'Longitude: ' + position.coords.longitude + '<br />' +
          '<hr />' + element.innerHTML;
      }

      // onError Callback receives a PositionError object
      //
      function onError(error) {
        alert('code: ' + error.code + '\n' +
          'message: ' + error.message + '\n');
      }

      function stopWatch(){

```

```
        navigator.geolocation.clearWatch(watchID);
    }

    </script>
</head>
<body>
    <p id="geolocation">Watching geolocation...</p>
    <button onClick="stopWatch()">Stop Watching</button>
</body>
</html>
```

Answer to Question 2

Following is an example of an application that displays a special message if the speed drops below a certain threshold:

```
<!DOCTYPE html>
<html>
<head>
    <title>Device Properties Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

        // Wait for PhoneGap to load
        //
        document.addEventListener("deviceready", onDeviceReady, false);

        var watchID = null;

        // PhoneGap is ready
        //
        function onDeviceReady() {
            // Update every 3 seconds
            var options = { frequency: 3000 };
            watchID = navigator.geolocation.watchPosition(onSuccess, onError, options);
        }

        // onSuccess Geolocation
        //
        function onSuccess(position) {
            var element = document.getElementById('geolocation');
            element.innerHTML = 'Latitude: ' + position.coords.latitude + '<br />' +
                'Longitude: ' + position.coords.longitude + '<br />';

            if(position.coords.speed < 2){
                var special = document.getElementById('special');
                special.innerHTML = '<b>Speed is below 2 meters/second!</b>';
            }
        }

        // onError Callback receives a PositionError object
        //
```

```

function onError(error) {
    alert('code: ' + error.code + '\n' +
          'message: ' + error.message + '\n');
}

</script>
</head>
<body>
    <p id="geolocation">Watching geolocation...</p>
    <p id="special"></p>
</body>
</html>

```

CHAPTER 9 EXERCISE ANSWERS

Following are the answers to the exercises in Chapter 9.

Answer to Question 1

Following is an example of an application that will start recording after a 5-second delay:

```

<!DOCTYPE html>
<html>
<head>
    <title>Device Properties Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

        document.addEventListener("deviceready", onDeviceReady, false);

        function recordAudio() {
            var src = "myrecording.mp3";
            var mediaRec = new Media(src, onSuccess, onError);

            // Record audio
            mediaRec.startRecord();

            // Start recording after 5 sec
            var recTime = 0;
            var recInterval = setTimeout(function() {
                recTime = recTime + 1;
                setAudioPosition(recTime + " sec");
                if (recTime >= 10) {
                    clearInterval(recInterval);
                    mediaRec.stopRecord();
                }
            }, 5000);
        }

        function onDeviceReady() {

```

```
        recordAudio();
    }

    function onSuccess() {
        console.log("recordAudio():Audio Success");
    }

    function onError(error) {
        alert('code: ' + error.code + '\n' +
            'message: ' + error.message + '\n');
    }

    function setAudioPosition(position) {
        document.getElementById('audio_position').innerHTML = position;
    }

</script>
</head>
<body>
    <p id="media">Recording audio...</p>
    <p id="audio_position"></p>
</body>
</html>
```

Answer to Question 2

Following is an example of an application that will play a local media file:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Media Example</title>

        <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
        <script type="text/javascript" charset="utf-8">

            document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady() {
    playAudio("/audio_files/audio.mp3"); //change this URL!
}

var my_media = null;
var mediaTimer = null;

// Play audio
//
function playAudio(src) {
    // Create Media object from src
    my_media = new Media(src, onSuccess, onError);

    // Play audio
```

```

my_media.play();

// Update my_media position every second
if (mediaTimer == null) {
    mediaTimer = setInterval(function() {
        // get my_media position
        my_media.getCurrentPosition(
            // success callback
            function(position) {
                if (position > -1) {
                    setAudioPosition((position/1000) + " sec");
                }
            },
            // error callback
            function(e) {
                console.log("Error getting pos=" + e);
                setAudioPosition("Error: " + e);
            }
        );
    }, 1000);
}

function pauseAudio() {
    if (my_media) {
        my_media.pause();
    }
}

function stopAudio() {
    if (my_media) {
        my_media.stop();
    }
    clearInterval(mediaTimer);
    mediaTimer = null;
}

function onSuccess() {
    console.log("playAudio():Audio Success");
}

function onError(error) {
    alert('code: ' + error.code + '\n' +
        'message: ' + error.message + '\n');
}

function setAudioPosition(position) {
    document.getElementById('audio_position').innerHTML = position;
}

</script>

```

```
</head>
<body>
  <a href="#" onclick="playAudio('/audio_files/audio.mp3');">Play Audio</a>
  <a href="#" onclick="pauseAudio();">Pause Playing Audio</a>
  <a href="#" onclick="stopAudio();">Stop Playing Audio</a>
  <p id="audio_position"></p>
</body>
</html>
```

CHAPTER 10 EXERCISE ANSWERS

Following are the answers to the exercises in Chapter 10.

Answer to Question 1

The following example shows how to create a 3-second delay before taking a photo:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Take a Photo</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready",onDeviceReady,false);

      function onDeviceReady() {
      }

      // Called when a photo is successfully retrieved
      //
      function onPhotoSuccess(imageData) {
      var myImage = document.getElementById('myImage');

      myImage.style.display = 'block';

      myImage.src = "data:image/jpeg;base64," + imageData;
      }

      function capturePhoto() {
      navigator.camera.getPicture(onPhotoSuccess, onPhotoFail, { quality: 50 });
      }

      function onPhotoFail(message) {
      alert('Failed because: ' + message);
      }

    </script>
```



```

</head>
<body>
  <button onclick='setTimeout("capturePhoto();",3000);'>3 second
    delay!</button> <br>

  <img style="display:none;width:60px;height:60px;" id="myImage" src="" />
</body>
</html>

```

Answer to Question 2

The following example shows how to add an option that allows the user to choose the quality of a photo taken with a camera:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Take a Photo</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready",onDeviceReady,false);

      function onDeviceReady() {
      }

      // Called when a photo is successfully retrieved
      //
      function onPhotoSuccess(imageData) {
      var myImage = document.getElementById('myImage');

      myImage.style.display = 'block';

      myImage.src = "data:image/jpeg;base64," + imageData;
      }

      function capturePhoto() {
        var dropdown = document.getElementById('quality');
        var myQuality = dropdown.options[dropdown.selectedIndex].value;
        navigator.camera.getPicture(onPhotoSuccess, onPhotoFail, { quality:
          myQuality });
      }

      function onPhotoFail(message) {
        alert('Failed because: ' + message);
      }

    </script>
  </head>

```

```
<body>
  <p>Quality: <select id='quality'>
    <option value='25'>25%</option>
    <option value='50'>50%</option>
    <option value='75'>75%</option>
  </select>
</p>

  <button onclick='capturePhoto()'>Take Picture</button> <br>

  <img style="display:none;width:60px;height:60px;" id="myImage" src="" />
</body>
</html>
```

CHAPTER 11 EXERCISE ANSWERS

Following are the answers to the exercises in Chapter 11.

Answer to Question 1

Following is an example of how to ask the user to enter his or her first and last name:

```
<!DOCTYPE html>
<html>
  <head>
    <title>localStorage Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

      document.addEventListener("deviceready", onDeviceReady, false);

      function onDeviceReady() {

      }

      function storeName(){
        var fName = document.getElementById("fname").value;
        var lName = document.getElementById("lname").value;
        var result = document.getElementById("result");

        window.localStorage.setItem("name", fName + " " +lName);

        result.innerHTML = "Your name is " +
          window.localStorage.getItem("name");

      }

    </script>
```

```

</head>
<body>
  <form>
    <p>What's your first name?<br/>
      <input type='text' id='fname' onKeyUp='storeName()' '>
    </p>
    <p>What's your last name?<br/>
      <input type='text' id='lname' onKeyUp='storeName()' '>

    </p>
  </form>

  <div id='result'></div>
</body>
</html>

```

Answer to Question 2

Following is an example of applying the jQTouch look and feel to the database example:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Database Example</title>
    <style type="text/css" media="screen">@import "jqtouch/jqtouch.css";</style>
    <style type="text/css" media="screen">@import "themes/jqt/theme.css";</style>
    <script src="jqtouch/jquery-1.4.2.js" type="text/javascript"
      charset="utf-8"></script>
    <script src="jqtouch/jqtouch.js" type="application/x-javascript"
      charset="utf-8"></script>
    <script type="text/javascript" charset="utf-8">
      var jQT = new $.jQTouch({
        icon: 'jqtouch.png',
        icon4: 'jqtouch4.png',
        addGlossToIcon: false,
        startupScreen: 'jqt_startup.png',
        statusBar: 'black',
        preloadImages: [
          'themes/jqt/img/activeButton.png',
          'themes/jqt/img/back_button.png',
          'themes/jqt/img/back_button_clicked.png',
          'themes/jqt/img/blueButton.png',
          'themes/jqt/img/button.png',
          'themes/jqt/img/button_clicked.png',
          'themes/jqt/img/grayButton.png',
          'themes/jqt/img/greenButton.png',
          'themes/jqt/img/redButton.png',
          'themes/jqt/img/whiteButton.png',
          'themes/jqt/img/loading.gif'
        ]
      });
    </script>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>

```

```
<script type="text/javascript" charset="utf-8">

document.addEventListener("deviceready", onDeviceReady, false);

function populateDB(tx) {
    tx.executeSql('DROP TABLE IF EXISTS test1');
    tx.executeSql('CREATE TABLE IF NOT EXISTS test1 (id unique, name)');
    tx.executeSql('INSERT INTO test1 (id, name) VALUES (1, "Tony")');
    tx.executeSql('INSERT INTO test1 (id, name) VALUES (2, "Bill")');
    tx.executeSql('INSERT INTO test1 (id, name) VALUES (3, "Thomas")');
}

function queryDB(tx) {
    tx.executeSql('SELECT * FROM test1', [], querySuccess, errorCallback);
}

// Query the success callback
//
function querySuccess(tx, results) {
    //first get the number of rows in the result set
    var len = results.rows.length;
    var status = document.getElementById("status");
    var string = "Rows: " +len+"<br/>";

    for (var i=0;i<len;i++){
        string += "<li>"+results.rows.item(i).name + "</li>";
    }

    status.innerHTML = string;
}

function errorDB(err) {
    alert("Error processing SQL: "+err.code);
}

function successDB() {
    var db = window.openDatabase("Test", "1.0", "Test", 200000);
    db.transaction(queryDB, errorCallback);
}

function onDeviceReady() {
    var db = window.openDatabase("Test", "1.0", "Test", 200000);
    db.transaction(populateDB, errorDB, successDB);
}

</script>
</head>
<body>
    <div id="home" class="current">
        <div class="toolbar">
            <h1>Names</h1>
```

```

    </div>

    <ul class='rounded' id='status'>
    </ul>

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

```

CHAPTER 12 EXERCISE ANSWER

Following is the answer to the exercise in Chapter 12.

Answer to Question 1

Following is an example of how to upload a photo from the photo gallery to a server:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>File Transfer Example</title>

    <script type="text/javascript" charset="utf-8"
        src="phonegap.0.9.4.min.js"></script>
    <script type="text/javascript" charset="utf-8">

document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady() {

    // Retrieve image file location
    navigator.camera.getPicture(uploadPhoto,
                                function(message) { alert('get picture
                                    failed'); },
                                { quality: 50,
                                  destinationType:
                                    navigator.camera
                                      .DestinationType.FILE_URI,
                                  sourceType:
                                    navigator.camera.PictureSourceType
                                      .PHOTOLIBRARY }
                                );
}

function uploadPhoto(imageURI) {
    var options = new FileUploadOptions();
    options.fileKey="file";
    options.fileName=imageURI.substr(imageURI.lastIndexOf('/')+1);

```

```
        options.mimeType="image/jpeg";

        var params = new Object();
        params.value1 = "test value";
        params.value2 = "param value";

        options.params = params;

        var ft = new FileTransfer();
        ft.upload(imageURI, "http://example.com/upload.php", success, fail,
            options);
    }

    function success(r) {
        alert("Code = " + r.responseCode);
        alert("Response = " + r.response);
        alert("Sent = " + r.bytesSent);
    }

    function fail(error) {
        alert("An error has occurred: Code = " + error.code);
    }

</script>
</head>
<body>
    <h1>Example</h1>
    <p>Upload File</p>
</body>
</html>
```

CHAPTER 13 EXERCISE ANSWER

Following is the answer to the exercise in Chapter 13.

Answer to Question 1

The following shows an example of how to find all the contacts in your database who are doctors:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Contact Example</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

    document.addEventListener("deviceready", onDeviceReady, false);

    function onDeviceReady() {
```

```

        var options = new ContactFindOptions();
        options.filter="Dr.";
        filter = ["displayName"];
        navigator.contacts.find(filter, onSuccess, onError, options);
    }

    // onSuccess: Get a snapshot of the current contacts
    //
    function onSuccess(contacts) {
        var myresults = document.getElementById("results");
        var string = "";
        for (var i=0; i<contacts.length; i++) {
            string += contacts[i].name.givenName + " " +
                contacts[i].name.familyName + " " +
                contacts[i].name.honorificSuffix + "<br/>";

        }
        myresults.innerHTML = string;

    // onError: Failed to get the contacts
    //
    function onError(contactError) {
        alert('onError!');
    }

</script>
</head>
<body>
    <h1>Example</h1>
    <div id="results"></div>
</body>
</html>

```

CHAPTER 14 EXERCISE ANSWER

Following is the answer to the exercise in Chapter 14.

Answer to Question 1

Following is an example of an app that allows audio, video, or image capture options:

```

<!DOCTYPE html>
<html>
  <head>
    <title>Capture Files</title>

    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

    function onSuccess(mediaFiles) {
        var i, len;

```

```
        for (i = 0, len = mediaFiles.length; i < len; i += 1) {
            uploadFile(mediaFiles[i]);
        }
    }

function onError(error) {
    var msg = 'An error occurred during capture: ' + error.code;
    navigator.notification.alert(msg, null, 'Uh oh!');
}

function capture(type) {
    if (type == "image"){
        navigator.device.capture.captureImage(onSuccess, onError, {limit: 2});
    }else if (type == "audio"){
        navigator.device.capture.captureAudio(onSuccess, onError, {limit: 2});
    }else{
        navigator.device.capture.captureVideo(onSuccess, onError, {limit: 2});
    }
}

// Upload files to server
function uploadFile(mediaFile) {
    var status = document.getElementById("status");
    var ft = new FileTransfer(),
        path = mediaFile.fullPath,
        name = mediaFile.name;

    ft.upload(path,
        "http://example.com/upload.php",
        function(result) {

            status.innerHTML = 'Upload success: ' + result.responseCode + '.  

                Bytes sent: ' + result.bytesSent;
        },
        function(error) {
            status.innerHTML = 'Error uploading file ' + path + ':  

                ' + error.code;
        },
        { fileName: name });
}

</script>
</head>
<body>
    <button onclick="capture('image');">Capture Image</button> <br>
<button onclick="capture('audio');">Capture Audio</button> <br>
<button onclick="capture('video');">Capture Video</button> <br>
    <div id='status'></div>
</body>
</html>
```


B

Tools for PhoneGap

This appendix lists various tools and frameworks you can use to make mobile application development with PhoneGap more fun and exciting.

APPML

appML (shown in Figure B-1) is an HTML extension specifically designed for web-based apps. It is a merging of jQTouch and iScroll (both of which are described later in this appendix). It is based on jQuery for easy app creation (iOS only) by using tags like `tool`, `carousel`, `scrollable`, `sidebar`, `panel`, and `page`.



FIGURE B-1: appML HTML extension

For more information, see www.appml.org/.

APPMOBI

appMobi solves various challenges faced by PhoneGap developers. With it, you don't have to set up native development environments based on Software Development Kits (SDKs). You get more debugging tools, and you don't have to develop your own web view containers for each supported platform.

For more information, see www.appmobi.com/index.php?q=node/153.

APPMOBI XDK

appMobi XDK is a free development tool that allows you to interactively test, debug, and compile cross-platform HTML5/CSS3/JavaScript apps that use PhoneGap or appMobi device APIs.

For more information, see www.appmobi.com/.

CONVERTIGO MOBILIZER

Convertigo offers a core technology platform called Mobilizer that can create new mobile applications out of any existing web or legacy application in the enterprise. This alleviates the need to rewrite legacy and mainframe applications to fit your modern infrastructure. Instead, you can use them as is.

For more information, see www.convertigo.com/mobilizer.

DOJO MOBILE

The Dojo Mobile framework (Figure B-2) provides a variety of great user interface elements, interactions, and APIs. User interface widgets include buttons, edge-to-edge lists, flippable views, icon containers, menus, progress indicators, scrollable views, side-by-side layouts, toggle buttons, tab bars, and much more. Support is in place for iOS (iPhone and iPad) and Android themes, and is in progress for Blackberry 6, webOS, and neutral themes. CSS3 animations and transition effects are supported.

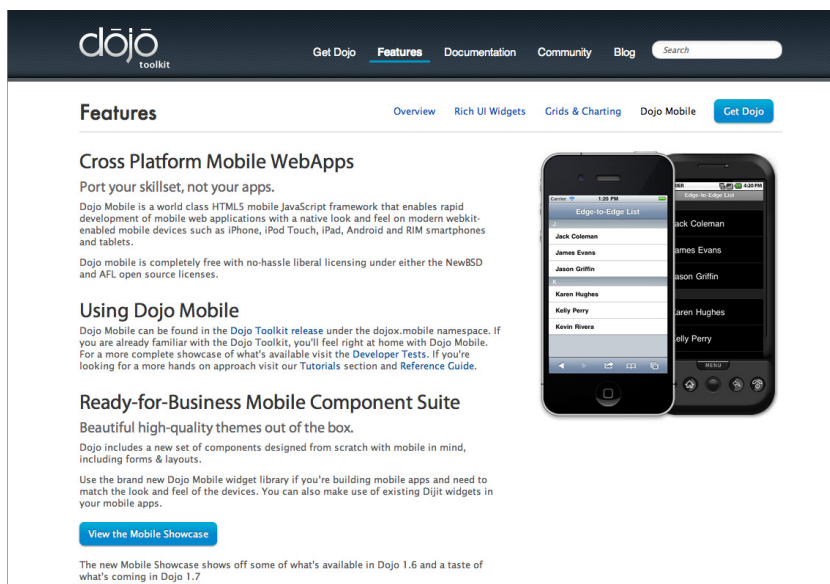


FIGURE B-2: Dojo Mobile framework

For more information, see <http://dojotoolkit.org/features/mobile.php>.

EASY APNS

This tool enables developers to add Apple push notification services using PHP and MySQL to their mobile apps.

For more information, see www.easyapns.com/.

EMBEDJS

EmbedJS is a JavaScript framework for embedded devices such as mobile phones, TVs, and so on. It is based on the Dojo Toolkit. (See the earlier description of Dojo Mobile.) It ships with just the code needed for each device.

For more information, see <http://uxebu.github.com/embedjs/>.

GLOVEBOX

GloveBox from Github Social Coding offers a bare-metal (that is, it provides the essentials, without frills) touch-enabled scroll project for the WebKit browser that includes all JavaScript with no external dependencies.

For more information, see <http://github.com/purplecabbage/GloveBox>.

GWT MOBILE

GWT Mobile is a cross-platform mobile development tool using Google Web Toolkit (GWT) technology. It provides a set of user interface (UI) widgets optimized for mobile devices, an Object Relational Model (ORM) module to persist objects to the browser database, and a wrapper to access PhoneGap functions from GWT. type="note"

For more information, see www.gwtmobile.com/.

HTML5SQL

Anyone who has used the native HTML5 Web SQL specification knows that it contains some tricky callbacks. As shown in Figure B-3, `html5sql` is a lightweight JavaScript module that makes working with the HTML5 Web Database a whole lot easier. Its primary feature is the capability to quickly process a batch of SQL statements within a single database transaction, (which can even be loaded from a separate text file though Ajax. There is also support for the sequential processing of SQL statements within the same transaction. If you need a database for your application, check out `html5sql`.



FIGURE B-3: `html5sql` JavaScript module

For more information, see <http://html5sql.com/>.

IMPACT

As shown in Figure B-4, Impact is a JavaScript Game Engine that enables you to easily develop games with PhoneGap.



FIGURE B-4: Impact JavaScript Game Engine

For more information, see <http://impactjs.com/>.

IScroll

If you've developed on native iPhone, you know that there is no native support for scrolling content inside a fixed-size container. iScroll bills itself as the `overflow:scroll` for the mobile WebKit browser, providing a solution for this problem.

For more information, see <http://cubiq.org/iscroll>.

IWEBKIT

iWebKit is a toolkit you can use to quickly create iPhone and iPad apps and mobile-friendly websites. It uses simple HTML to create projects, and is supposedly easy enough for anyone to master in minutes.

For more information, see <http://snippetspace.com/projects/iwebkit/>.

JO HTML5 FRAMEWORK

Jo is a small HTML5 application framework designed to run on top of PhoneGap. Jo features common mobile UI widgets, and is completely customizable with CSS3. Jo works with PhoneGap, most mobile browsers, desktop browsers, and even Dashboard widgets.

For more information, see <http://joapp.com/>.

JQTOUCH

jQTouch is a jQuery plug-in for mobile web development on the iPhone, iPod Touch, and other devices. It provides a number of very useful libraries for creating touch interfaces with HTML, CSS, and JavaScript that mimic the user experience provided by native iPhone applications.

For more information, see <http://jqtouch.com/>.

JQUERY MOBILE

jQuery Mobile (Figure B-5) is a unified UI system working across all popular mobile device platforms. Built on the rock-solid jQuery and jQuery UI foundation, it is very easy to theme.



FIGURE B-5: jQuery Mobile UI system

For more information, see <http://jquerymobile.com/>.

LAWNCHAIR

Lawnchair (shown in Figure B-6) implements a client-side JavaScript Object Notation (JSON) document store. It is perfect for WebKit mobile apps that need a lightweight, simple, and elegant persistence solution.

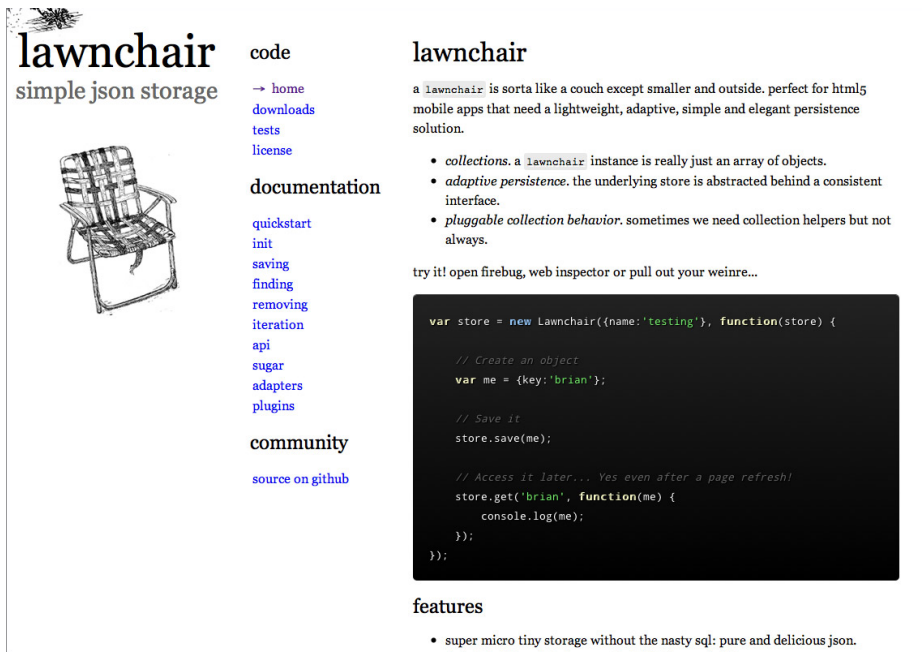


FIGURE B-6: Lawnchair with JSON document store

For more information, see <http://westcoastlogic.com/lawnchair>.

MDS ECLIPSE PLUG-IN FOR ANDROID PHONEGAP

The Mobile Developer Solutions (MDS) plug-in extends the Android Development Tools (ADT) and Eclipse JavaScript Development Tools (JSDT). It includes a project wizard to configure PhoneGap projects, enabling app development and deployment in a few clicks.

For more information, see www.mobiledvelopersolutions.com/.

MOBILEFX STUDIO7

mobileFX Studio7 is an integrated development environment (IDE) for rapid prototyping of mobile applications and two-dimensional (2D) Games. With Studio7, you can build native Java (MIDP/CLDC or RIMLet) applications and games, and HTML5/CSS3 native-looking web applications (currently for iPhone, with Android native/web under development).

For more information, see www.mobilefx.com/web/products/studio/default.asp.

NS BASIC/APP STUDIO

NS Basic/App Studio is a complete, powerful development environment. With it, you can create apps on the desktop, and then download them to your device to run. It also supports development in JavaScript and contains SQLite support. It comes with a royalty-free distribution. It includes lots of sample code and a 200-page handbook.

For more information, see www.nsbasic.com/app.

OSMEK

Osmek is a content management system (CMS) built in the cloud. All integration is done through Osmek's API, so your content is available anywhere, and in just about any format. JSON, JSONP, XML, and HTML support make it perfect for web and mobile app solutions.

For more information, see <http://osmek.com/>.

QOOXD00

qooxdoo is a universal JavaScript framework that enables you to create applications for a wide range of platforms. Its object-oriented programming model lets you build native-like apps for mobile devices.

For more information, see <http://qooxdoo.org/>.

RIPPLE

Ripple is a multi-platform mobile environment emulator that runs in the Chrome web browser and is custom-tailored to HTML5 mobile application testing.

For more information, see <http://ripple.tinyhippos.com/>.

SENCHA TOUCH

Sencha Touch enables you to develop mobile web apps that look and feel native on iPhone, Android, and BlackBerry touch devices. It is a JavaScript framework that features a full set of UI components, a powerful data library for Ajax, stores, and proxies, along with a robust theming system.

For more information, see www.sencha.com/products/touch/.

SONY ERICSSON WEBSDK PACKAGER

The Sony Ericsson WebSDK Packager enables web developers to produce mobile apps that work across different phone platforms. It is an Open Source tool created in collaboration with the PhoneGap Open Source community. Applications are written using HTML, CSS, and JavaScript.

For more information, see <http://developer.sonyericsson.com/wportal/devworld/technology/web/websdk-packager>.

UNIFY

Unify is a tool used to build native-like applications for smartphones, tablets, and desktops. Currently, it supports smartphones based on the iOS, Android, and webOS platforms.

For more information, see <http://unify-project.org/>.

URBAN AIRSHIP

The Urban Airship tool shown in Figure B-7 uses simple drop-in client libraries and a RESTful API that enables developers to integrate real-time push messaging or in-app purchases to their applications.



FIGURE B-7: Urban Airship

For more information, see <http://urbanairship.com/>.

WINK

Wink is a mobile JavaScript framework that you can use to develop mobile web apps on iPhone, iPad, Android, and BlackBerry.

For more information, see www.winktoolkit.org/.

XUI

XUI is a simple JavaScript framework used to build mobile web applications. It supports device browsers such as WebKit, Fennec, and Opera with future support under consideration for Internet Explorer (IE) Mobile and BlackBerry.

For more information, see <http://xuijs.com/>.

ZEPTO.JS

Zepto.js is a minimalist framework for mobile WebKit browsers, with a jQuery-compatible chaining syntax. All APIs provided match their jQuery counterparts.

For more information, see <http://zeptojs.com/>.



PhoneGap.js

Following is the code contained within the PhoneGap.js file:

```
/*
 * PhoneGap is available under *either* the terms of the modified BSD license
 * *or* the * MIT License (2008).
 * See http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (typeof PhoneGap === "undefined") {

/**
 * The order of events during page load and PhoneGap startup is as follows:
 *
 * onDOMContentLoaded      Internal event that is received when the web page
 *                          is loaded and parsed.
 * window.onload           Body onload event.
 * onNativeReady           Internal event that indicates the PhoneGap native
 *                          side is ready.
 * onPhoneGapInit          Internal event that kicks off creation of all
 *                          PhoneGap JavaScript objects (runs constructors).
 * onPhoneGapReady         Internal event fired when all PhoneGap JavaScript
 *                          objects have been created
 * onPhoneGapInfoReady     Internal event fired when device properties are
 *                          available
 * onDeviceReady           User event fired to indicate that PhoneGap is
 *                          ready
 * onResume                User event fired to indicate a start/resume
 *                          lifecycle event
 */
}
```

```
* onPause                User event fired to indicate a pause lifecycle
*                        event
* onDestroy              Internal event fired when app is being destroyed
*                        (User should use window.onunload event,
*                        not this one).
*
* The only PhoneGap events that user code should register for are:
*   onDeviceReady
*   onResume
*
* Listeners can be registered as:
*   document.addEventListener("deviceready", myDeviceReadyListener, false);
*   document.addEventListener("resume", myResumeListener, false);
*   document.addEventListener("pause", myPauseListener, false);
*/

if (typeof(DeviceInfo) !== 'object') {
    var DeviceInfo = {};
}

/**
 * This represents the PhoneGap API itself, and provides a global namespace for
 * accessing
 * information about the state of PhoneGap.
 * @class
 */
var PhoneGap = {
    queue: {
        ready: true,
        commands: [],
        timer: null
    }
};

/**
 * List of resource files loaded by PhoneGap.
 * This is used to ensure JS and other files are loaded only once.
 */
PhoneGap.resources = {base: true};

/**
 * Determine if resource has been loaded by PhoneGap
 *
 * @param name
 * @return
 */
PhoneGap.hasResource = function(name) {
    return PhoneGap.resources[name];
};

/**
 * Add a resource to list of loaded resources by PhoneGap
 *
 * @param name
 */
PhoneGap.addResource = function(name) {
```

```

        PhoneGap.resources[name] = true;
    };

    /**
     * Custom pub-sub channel that can have functions subscribed to it
     * @constructor
     */
    PhoneGap.Channel = function (type)
    {
        this.type = type;
        this.handlers = {};
        this.guid = 0;
        this.fired = false;
        this.enabled = true;
    };

    /**
     * Subscribes the given function to the channel. Any time that
     * Channel.fire is called so too will the function.
     * Optionally specify an execution context for the function
     * and a guid that can be used to stop subscribing to the channel.
     * Returns the guid.
     */
    PhoneGap.Channel.prototype.subscribe = function(f, c, g) {
        // need a function to call
        if (f === null) { return; }

        var func = f;
        if (typeof c === "object" && typeof f === "function") { func =
            PhoneGap.close(c, f); }

        g = g || func.observer_guid || f.observer_guid || this.guid++;
        func.observer_guid = g;
        f.observer_guid = g;
        this.handlers[g] = func;
        return g;
    };

    /**
     * Like subscribe but the function is only called once and then it
     * auto-unsubscribes itself.
     */
    PhoneGap.Channel.prototype.subscribeOnce = function(f, c) {
        var g = null;
        var _this = this;
        var m = function() {
            f.apply(c || null, arguments);
            _this.unsubscribe(g);
        };
        if (this.fired) {
            if (typeof c === "object" && typeof f === "function") { f =
                PhoneGap.close(c, f); }
            f.apply(this, this.fireArgs);
        } else {
            g = this.subscribe(m);
        }
    }

```

```
        return g;
    };

    /**
     * Unsubscribes the function with the given guid from the channel.
     */
    PhoneGap.Channel.prototype.unsubscribe = function(g) {
        if (typeof g === "function") { g = g.observer_guid; }
        this.handlers[g] = null;
        delete this.handlers[g];
    };

    /**
     * Calls all functions subscribed to this channel.
     */
    PhoneGap.Channel.prototype.fire = function(e) {
        if (this.enabled) {
            var fail = false;
            var item, handler, rv;
            for (item in this.handlers) {
                if (this.handlers.hasOwnProperty(item)) {
                    handler = this.handlers[item];
                    if (typeof handler === "function") {
                        rv = (handler.apply(this, arguments) === false);
                        fail = fail || rv;
                    }
                }
            }
            this.fired = true;
            this.fireArgs = arguments;
            return !fail;
        }
        return true;
    };

    /**
     * Calls the provided function only after all of the channels specified
     * have been fired.
     */
    PhoneGap.Channel.join = function(h, c) {
        var i = c.length;
        var f = function() {
            if (!(--i)) {
                h();
            }
        };
    };
    var len = i;
    var j;
    for (j=0; j<len; j++) {
        if (!c[j].fired) {
            c[j].subscribeOnce(f);
        }
        else {
            i--;
        }
    }
}
```



```

        if (!i) {
            h();
        }
    };

    /**
     * Boolean flag indicating if the PhoneGap API is available and initialized.
     * // TODO: Remove this, it is unused here ... -jm
    PhoneGap.available = DeviceInfo.uuid !== undefined;

    /**
     * Add an initialization function to a queue that ensures it will run and
     * initialize application constructors only once PhoneGap has been initialized.
     * @param {Function} func The function callback you want run once PhoneGap is
     *                        initialized
     */
    PhoneGap.addConstructor = function(func) {
        PhoneGap.onPhoneGapInit.subscribeOnce(function() {
            try {
                func();
            } catch(e) {
                console.log("Failed to run constructor: " + e);
            }
        });
    };

    /**
     * Plugins object
     */
    if (!window.plugins) {
        window.plugins = {};
    }

    /**
     * Adds a plugin object to window.plugins.
     * The plugin is accessed using window.plugins.<name>
     *
     * @param name          The plugin name
     * @param obj           The plugin object
     */
    PhoneGap.addPlugin = function(name, obj) {
        if (!window.plugins[name]) {
            window.plugins[name] = obj;
        }
        else {
            console.log("Error: Plugin "+name+" already exists.");
        }
    };

    /**
     * onDOMContentLoaded channel is fired when the DOM content
     * of the page has been parsed.
     */
    PhoneGap.onDOMContentLoaded = new PhoneGap.Channel('onDOMContentLoaded');

    /**

```

```
* onNativeReady channel is fired when the PhoneGap native code
* has been initialized.
*/
PhoneGap.onNativeReady = new PhoneGap.Channel('onNativeReady');

/**
 * onPhoneGapInit channel is fired when the web page is fully loaded and
 * PhoneGap native code has been initialized.
 */
PhoneGap.onPhoneGapInit = new PhoneGap.Channel('onPhoneGapInit');

/**
 * onPhoneGapReady channel is fired when the JS PhoneGap objects have been created.
 */
PhoneGap.onPhoneGapReady = new PhoneGap.Channel('onPhoneGapReady');

/**
 * onPhoneGapInfoReady channel is fired when the PhoneGap device properties
 * has been set.
 */
PhoneGap.onPhoneGapInfoReady = new PhoneGap.Channel('onPhoneGapInfoReady');

/**
 * onPhoneGapConnectionReady channel is fired when the PhoneGap connection
 * properties has been set.
 */
PhoneGap.onPhoneGapConnectionReady = new
    PhoneGap.Channel('onPhoneGapConnectionReady');

/**
 * onResume channel is fired when the PhoneGap native code
 * resumes.
 */
PhoneGap.onResume = new PhoneGap.Channel('onResume');

/**
 * onPause channel is fired when the PhoneGap native code
 * pauses.
 */
PhoneGap.onPause = new PhoneGap.Channel('onPause');

/**
 * onDestroy channel is fired when the PhoneGap native code
 * is destroyed. It is used internally.
 * Window.onunload should be used by the user.
 */
PhoneGap.onDestroy = new PhoneGap.Channel('onDestroy');
PhoneGap.onDestroy.subscribeOnce(function() {
    PhoneGap.shuttingDown = true;
});
PhoneGap.shuttingDown = false;

// _nativeReady is global variable that the native side can set
// to signify that the native code is ready. It is a global since
```

```

// it may be called before any PhoneGap JS is ready.
if (typeof _nativeReady !== 'undefined') { PhoneGap.onNativeReady.fire(); }

/**
 * onDeviceReady is fired only after all PhoneGap objects are created and
 * the device properties are set.
 */
PhoneGap.onDeviceReady = new PhoneGap.Channel('onDeviceReady');

// Array of channels that must fire before "deviceready" is fired
PhoneGap.deviceReadyChannelsArray = [ PhoneGap.onPhoneGapReady,
    PhoneGap.onPhoneGapInfoReady, PhoneGap.onPhoneGapConnectionReady];

// Hashtable of user defined channels that must also fire before "deviceready"
// is fired
PhoneGap.deviceReadyChannelsMap = {};

/**
 * Indicate that a feature needs to be initialized before it is ready to be used.
 * This holds up PhoneGap's "deviceready" event until the feature has been
 * initialized and PhoneGap.initComplete(feature) is called.
 *
 * @param feature {String}      The unique feature name
 */
PhoneGap.waitForInitialization = function(feature) {
    if (feature) {
        var channel = new PhoneGap.Channel(feature);
        PhoneGap.deviceReadyChannelsMap[feature] = channel;
        PhoneGap.deviceReadyChannelsArray.push(channel);
    }
};

/**
 * Indicate that initialization code has completed and the feature is ready to
 * be used.
 *
 * @param feature {String}      The unique feature name
 */
PhoneGap.initializationComplete = function(feature) {
    var channel = PhoneGap.deviceReadyChannelsMap[feature];
    if (channel) {
        channel.fire();
    }
};

/**
 * Create all PhoneGap objects once page has fully loaded and native side is ready.
 */
PhoneGap.Channel.join(function() {

    // Start listening for XHR callbacks
    setTimeout(function() {
        if (PhoneGap.UsePolling) {
            PhoneGap.JSCallbackPolling();

```

```
    }
    else {
        var polling = prompt("usePolling", "gap_callbackServer:");
        PhoneGap.UsePolling = polling;
        if (polling == "true") {
            PhoneGap.UsePolling = true;
            PhoneGap.JSCallbackPolling();
        }
        else {
            PhoneGap.UsePolling = false;
            PhoneGap.JSCallback();
        }
    }
}, 1);

// Run PhoneGap constructors
PhoneGap.onPhoneGapInit.fire();

// Fire event to notify that all objects are created
PhoneGap.onPhoneGapReady.fire();

// Fire onDeviceReady event once all constructors have run and PhoneGap
// info has been received from native side, and any user defined
// initialization channels.
PhoneGap.Channel.join(function() {
    PhoneGap.onDeviceReady.fire();

    // Fire the onresume event, since first one happens before JavaScript
    // is loaded
    PhoneGap.onResume.fire();
}, PhoneGap.deviceReadyChannelsArray);

}, [ PhoneGap.onDOMContentLoaded, PhoneGap.onNativeReady ]);

// Listen for DOMContentLoaded and notify our channel subscribers
document.addEventListener('DOMContentLoaded', function() {
    PhoneGap.onDOMContentLoaded.fire();
}, false);

// Intercept calls to document.addEventListener and watch for deviceready
PhoneGap.m_document_addEventListener = document.addEventListener;

document.addEventListener = function(evt, handler, capture) {
    var e = evt.toLowerCase();
    if (e === 'deviceready') {
        PhoneGap.onDeviceReady.subscribeOnce(handler);
    } else if (e === 'resume') {
        PhoneGap.onResume.subscribe(handler);
        if (PhoneGap.onDeviceReady.fired) {
            PhoneGap.onResume.fire();
        }
    } else if (e === 'pause') {
        PhoneGap.onPause.subscribe(handler);
    }
}
```

```

    else {
        // If subscribing to Android backbutton
        if (e === 'backbutton') {
            PhoneGap.exec(null, null, "App", "overrideBackbutton", [true]);
        }

        PhoneGap.m_document_addEventListener.call(document, evt, handler, capture);
    }
};

// Intercept calls to document.removeEventListener and watch for events that
// are generated by PhoneGap native code
PhoneGap.m_document_removeEventListener = document.removeEventListener;

document.removeEventListener = function(evt, handler, capture) {
    var e = evt.toLowerCase();

    // If unsubscribing to Android backbutton
    if (e === 'backbutton') {
        PhoneGap.exec(null, null, "App", "overrideBackbutton", [false]);
    }

    PhoneGap.m_document_removeEventListener.call(document, evt, handler, capture);
};

/**
 * Method to fire event from native code
 */
PhoneGap.fireEvent = function(type) {
    var e = document.createEvent('Events');
    e.initEvent(type);
    document.dispatchEvent(e);
};

/**
 * If JSON not included, use our own stringify. (Android 1.6)
 * The restriction on ours is that it must be an array of simple types.
 *
 * @param args
 * @return {String}
 */
PhoneGap.stringify = function(args) {
    if (typeof JSON === "undefined") {
        var s = "[";
        var i, type, start, name, nameType, a;
        for (i = 0; i < args.length; i++) {
            if (args[i] !== null) {
                if (i > 0) {
                    s = s + ",";
                }
                type = typeof args[i];
                if ((type === "number") || (type === "boolean")) {
                    s = s + args[i];
                } else if (args[i] instanceof Array) {
                    s = s + "[" + args[i] + "]";
                } else if (args[i] instanceof Object) {

```

```
        start = true;
        s = s + '{';
        for (name in args[i]) {
            if (args[i][name] !== null) {
                if (!start) {
                    s = s + ',';
                }
                s = s + '"' + name + ':';
                nameType = typeof args[i][name];
                if ((nameType === "number") || (nameType ===
                    "boolean")) {
                    s = s + args[i][name];
                } else if ((typeof args[i][name]) === 'function') {
                    // don't copy the functions
                    s = s + '""';
                } else if (args[i][name] instanceof Object) {
                    s = s + PhoneGap.stringify(args[i][name]);
                } else {
                    s = s + '"' + args[i][name] + '"';
                }
                start = false;
            }
        }
        s = s + '>';
    } else {
        a = args[i].replace(/\\/g, '\\\\');
        a = a.replace(/"/g, '\\"');
        s = s + '"' + a + '"';
    }
}

}

s = s + "];
return s;
} else {
    return JSON.stringify(args);
}
};

/**
 * Does a deep clone of the object.
 *
 * @param obj
 * @return {Object}
 */
PhoneGap.clone = function(obj) {
    var i, retVal;
    if(!obj) {
        return obj;
    }

    if(obj instanceof Array){
        retVal = [];
        for(i = 0; i < obj.length; ++i){
            retVal.push(PhoneGap.clone(obj[i]));
        }
    }
```

```

        return retVal;
    }

    if (typeof obj === "function") {
        return obj;
    }

    if (!(obj instanceof Object)){
        return obj;
    }

    if (obj instanceof Date) {
        return obj;
    }

    retVal = {};
    for(i in obj){
        if(!(i in retVal) || retVal[i] !== obj[i]) {
            retVal[i] = PhoneGap.clone(obj[i]);
        }
    }
    return retVal;
};

PhoneGap.callbackId = 0;
PhoneGap.callbacks = {};
PhoneGap.callbackStatus = {
    NO_RESULT: 0,
    OK: 1,
    CLASS_NOT_FOUND_EXCEPTION: 2,
    ILLEGAL_ACCESS_EXCEPTION: 3,
    INSTANTIATION_EXCEPTION: 4,
    MALFORMED_URL_EXCEPTION: 5,
    IO_EXCEPTION: 6,
    INVALID_ACTION: 7,
    JSON_EXCEPTION: 8,
    ERROR: 9
};

/**
 * Execute a PhoneGap command. It is up to the native side whether this action
 * is synch or async.
 * The native side can return:
 *     Synchronous: PluginResult object as a JSON string
 *     Asynchronous: Empty string ""
 * If async, the native side will PhoneGap.callbackSuccess or
 * PhoneGap.callbackError, depending upon the result of the action.
 *
 * @param {Function} success    The success callback
 * @param {Function} fail      The fail callback
 * @param {String} service     The name of the service to use
 * @param {String} action      Action to be run in PhoneGap
 * @param {Array.<String>} [args] Zero or more arguments to pass to the
 *                                method
 */

```

```
PhoneGap.exec = function(success, fail, service, action, args) {
  try {
    var callbackId = service + PhoneGap.callbackId++;
    if (success || fail) {
      PhoneGap.callbacks[callbackId] = {success:success, fail:fail};
    }

    var r = prompt(PhoneGap.stringify(args),
      "gap:"+PhoneGap.stringify([service, action, callbackId, true]));

    // If a result was returned
    if (r.length > 0) {
      eval("var v="+r+";");

      // If status is OK, then return value back to caller
      if (v.status === PhoneGap.callbackStatus.OK) {

        // If there is a success callback, then call it now with
        // returned value
        if (success) {
          try {
            success(v.message);
          } catch (e) {
            console.log("Error in success callback: " + callbackId
              + " = " + e);
          }

          // Clear callback if not expecting any more results
          if (!v.keepCallback) {
            delete PhoneGap.callbacks[callbackId];
          }
        }
        return v.message;
      }

      // If no result
      else if (v.status === PhoneGap.callbackStatus.NO_RESULT) {

        // Clear callback if not expecting any more results
        if (!v.keepCallback) {
          delete PhoneGap.callbacks[callbackId];
        }
      }

      // If error, then display error
      else {
        console.log("Error: Status="+v.status+" Message="+v.message);

        // If there is a fail callback, then call it now with returned
        // value
        if (fail) {
          try {
            fail(v.message);
          }
        }
      }
    }
  }
}
```



```

        catch (e1) {
            console.log("Error in error callback: "+callbackId+" = "+e1);
        }

        // Clear callback if not expecting any more results
        if (!v.keepCallback) {
            delete PhoneGap.callbacks[callbackId];
        }
    }
    return null;
}
} catch (e2) {
    console.log("Error: "+e2);
}
};

/**
 * Called by native code when returning successful result from an action.
 *
 * @param callbackId
 * @param args
 */
PhoneGap.callbackSuccess = function(callbackId, args) {
    if (PhoneGap.callbacks[callbackId]) {
        // If result is to be sent to callback
        if (args.status === PhoneGap.callbackStatus.OK) {
            try {
                if (PhoneGap.callbacks[callbackId].success) {
                    PhoneGap.callbacks[callbackId].success(args.message);
                }
            }
            catch (e) {
                console.log("Error in success callback: "+callbackId+" = "+e);
            }
        }

        // Clear callback if not expecting any more results
        if (!args.keepCallback) {
            delete PhoneGap.callbacks[callbackId];
        }
    }
};

/**
 * Called by native code when returning error result from an action.
 *
 * @param callbackId
 * @param args
 */
PhoneGap.callbackError = function(callbackId, args) {
    if (PhoneGap.callbacks[callbackId]) {

```

```
    try {
      if (PhoneGap.callbacks[callbackId].fail) {
        PhoneGap.callbacks[callbackId].fail(args.message);
      }
    }
    catch (e) {
      console.log("Error in error callback: "+callbackId+" = "+e);
    }

    // Clear callback if not expecting any more results
    if (!args.keepCallback) {
      delete PhoneGap.callbacks[callbackId];
    }
  }
};

/**
 * Internal function used to dispatch the request to PhoneGap. It processes the
 * command queue and executes the next command on the list. If one of the
 * arguments is a JavaScript object, it will be passed on the QueryString of the
 * url, which will be turned into a dictionary on the other end.
 * @private
 */
// TODO: Is this used?
PhoneGap.run_command = function() {
  if (!PhoneGap.available || !PhoneGap.queue.ready) {
    return;
  }
  PhoneGap.queue.ready = false;

  var args = PhoneGap.queue.commands.shift();
  if (PhoneGap.queue.commands.length === 0) {
    clearInterval(PhoneGap.queue.timer);
    PhoneGap.queue.timer = null;
  }

  var uri = [];
  var dict = null;
  var i;
  for (i = 1; i < args.length; i++) {
    var arg = args[i];
    if (arg === undefined || arg === null) {
      arg = '';
    }
    if (typeof(arg) === 'object') {
      dict = arg;
    } else {
      uri.push(encodeURIComponent(arg));
    }
  }
  var url = "gap://" + args[0] + "/" + uri.join("/");
  if (dict !== null) {
    var name;
    var query_args = [];
```

```

        for (name in dict) {
            if (dict.hasOwnProperty(name) && (typeof (name) === 'string')) {
                query_args.push(encodeURIComponent(name) + "=" +
                    encodeURIComponent(dict[name]));
            }
        }
        if (query_args.length > 0) {
            url += "?" + query_args.join("&");
        }
    }
    document.location = url;

};

PhoneGap.JSCallbackPort = null;
PhoneGap.JSCallbackToken = null;

/**
 * This is only for Android.
 *
 * Internal function that uses XHR to call into PhoneGap Java code and retrieve
 * any JavaScript code that needs to be run. This is used for callbacks from
 * Java to JavaScript.
 */
PhoneGap.JSCallback = function() {

    // Exit if shutting down app
    if (PhoneGap.shuttingDown) {
        return;
    }

    // If polling flag was changed, start using polling from now on
    if (PhoneGap.UsePolling) {
        PhoneGap.JSCallbackPolling();
        return;
    }

    var xmlhttp = new XMLHttpRequest();

    // Callback function when XMLHttpRequest is ready
    xmlhttp.onreadystatechange=function(){
        if(xmlhttp.readyState === 4){

            // Exit if shutting down app
            if (PhoneGap.shuttingDown) {
                return;
            }

            // If callback has JavaScript statement to execute
            if (xmlhttp.status === 200) {

                // Need to url decode the response
                var msg = decodeURIComponent(xmlhttp.responseText);
                setTimeout(function() {
                    try {

```

```
        var t = eval(msg);
    }
    catch (e) {
        // If we're getting an error here, seeing the message
        // will help in debugging
        console.log("JSCallback: Message from Server: " + msg);
        console.log("JSCallback Error: "+e);
    }
    }, 1);
    setTimeout(PhoneGap.JSCallback, 1);
}

// If callback ping (used to keep XHR request from timing out)
else if (xmlhttp.status === 404) {
    setTimeout(PhoneGap.JSCallback, 10);
}

// If security error
else if (xmlhttp.status === 403) {
    console.log("JSCallback Error: Invalid token. Stopping
        callbacks.");
}

// If server is stopping
else if (xmlhttp.status === 503) {
    console.log("JSCallback Error: Service unavailable. Stopping
        callbacks.");
}

// If request wasn't GET
else if (xmlhttp.status === 400) {
    console.log("JSCallback Error: Bad request. Stopping callbacks.");
}

// If error, revert to polling
else {
    console.log("JSCallback Error: Request failed.");
    PhoneGap.UsePolling = true;
    PhoneGap.JSCallbackPolling();
}
}
};

if (PhoneGap.JSCallbackPort === null) {
    PhoneGap.JSCallbackPort = prompt("getPort", "gap_callbackServer:");
}
if (PhoneGap.JSCallbackToken === null) {
    PhoneGap.JSCallbackToken = prompt("getToken", "gap_callbackServer:");
}
xmlhttp.open("GET", "http://127.0.0.1:"+PhoneGap.JSCallbackPort+
    "/" +PhoneGap.JSCallbackToken , true);
xmlhttp.send();
};

/**
```

```

    * The polling period to use with JSCallbackPolling.
    * This can be changed by the application. The default is 50ms.
    */
PhoneGap.JSCallbackPollingPeriod = 50;

/**
 * Flag that can be set by the user to force polling to be used or force XHR to
 * be used.
 */
PhoneGap.UsePolling = false;    // T=use polling, F=use XHR

/**
 * This is only for Android.
 *
 * Internal function that uses polling to call into PhoneGap Java code and retrieve
 * any JavaScript code that needs to be run. This is used for callbacks from
 * Java to JavaScript.
 */
PhoneGap.JSCallbackPolling = function() {

    // Exit if shutting down app
    if (PhoneGap.shuttingDown) {
        return;
    }

    // If polling flag was changed, stop using polling from now on
    if (!PhoneGap.UsePolling) {
        PhoneGap.JSCallback();
        return;
    }

    var msg = prompt("", "gap_poll:");
    if (msg) {
        setTimeout(function() {
            try {
                var t = eval(""+msg);
            }
            catch (e) {
                console.log("JSCallbackPolling: Message from Server: " + msg);
                console.log("JSCallbackPolling Error: "+e);
            }
        }, 1);
        setTimeout(PhoneGap.JSCallbackPolling, 1);
    }
    else {
        setTimeout(PhoneGap.JSCallbackPolling, PhoneGap.JSCallbackPollingPeriod);
    }
};

/**
 * Create a UUID
 *
 * @return {String}
 */
PhoneGap.createUUID = function() {

```

```
        return PhoneGap.UUIDcreatePart(4) + '-' +
            PhoneGap.UUIDcreatePart(2) + '-' +
            PhoneGap.UUIDcreatePart(2) + '-' +
            PhoneGap.UUIDcreatePart(2) + '-' +
            PhoneGap.UUIDcreatePart(6);
    };

    PhoneGap.UUIDcreatePart = function(length) {
        var uuidpart = "";
        var i, uuidchar;
        for (i=0; i<length; i++) {
            uuidchar = parseInt((Math.random() * 256),0).toString(16);
            if (uuidchar.length === 1) {
                uuidchar = "0" + uuidchar;
            }
            uuidpart += uuidchar;
        }
        return uuidpart;
    };

    PhoneGap.close = function(context, func, params) {
        if (typeof params === 'undefined') {
            return function() {
                return func.apply(context, arguments);
            };
        } else {
            return function() {
                return func.apply(context, params);
            };
        }
    };

    /**
     * Load a JavaScript file after page has loaded.
     *
     * @param {String} jsfile          The url of the JavaScript file to load.
     * @param {Function} successCallback The callback to call when the file has
     *                                  been loaded.
     */
    PhoneGap.includeJavascript = function(jsfile, successCallback) {
        var id = document.getElementsByTagName("head")[0];
        var el = document.createElement('script');
        el.type = 'text/javascript';
        if (typeof successCallback === 'function') {
            el.onload = successCallback;
        }
        el.src = jsfile;
        id.appendChild(el);
    };

    }

    /**
     * PhoneGap is available under *either* the terms of the modified BSD
     * license *or* the MIT License (2008). See
     * http://opensource.org/licenses/alphabetical for full text.
     */

```

```

*
* Copyright (c) 2005-2010, Nitobi Software Inc.
* Copyright (c) 2010-2011, IBM Corporation
*/

if (!PhoneGap.hasResource("accelerometer")) {
PhoneGap.addResource("accelerometer");

/** @constructor */
var Acceleration = function(x, y, z) {
    this.x = x;
    this.y = y;
    this.z = z;
    this.timestamp = new Date().getTime();
};

/**
 * This class provides access to device accelerometer data.
 * @constructor
 */
var Accelerometer = function() {

    /**
     * The last known acceleration.  type=Acceleration()
     */
    this.lastAcceleration = null;

    /**
     * List of accelerometer watch timers
     */
    this.timers = {};
};

Accelerometer.ERROR_MSG = ["Not running", "Starting", "", "Failed to start"];

/**
 * Asynchronously acquires the current acceleration.
 *
 * @param {Function} successCallback    The function to call when the
 *                                     acceleration data is available
 * @param {Function} errorCallback    The function to call when there is an
 *                                     error getting the acceleration data.
 *                                     (OPTIONAL)
 * @param {AccelerationOptions} options The options for getting the
 *                                     accelerometer data such as timeout.
 *                                     (OPTIONAL)
 */
Accelerometer.prototype.getCurrentAcceleration =
    function(successCallback, errorCallback, options) {

        // successCallback required
        if (typeof successCallback !== "function") {
            console.log("Accelerometer Error: successCallback is not a function");
            return;

```

```
    }

    // errorCallback optional
    if (errorCallback && (typeof errorCallback !== "function")) {
        console.log("Accelerometer Error: errorCallback is not a function");
        return;
    }

    // Get acceleration
    PhoneGap.exec(successCallback, errorCallback, "Accelerometer",
        "getAcceleration", []);
};

/**
 * Asynchronously acquires the acceleration repeatedly at a given interval.
 *
 * @param {Function} successCallback The function to call each time the
 *                                  acceleration data is available
 * @param {Function} errorCallback The function to call when there is an
 *                                  error getting the acceleration data.
 *                                  (OPTIONAL)
 * @param {AccelerationOptions} options The options for getting the
 *                                  accelerometer data such as timeout.
 *                                  (OPTIONAL)
 * @return String The watch id that must be passed to
 *               #clearWatch to stop watching.
 */
Accelerometer.prototype.watchAcceleration = function(successCallback,
    errorCallback, options) {

    // Default interval (10 sec)
    var frequency = (options !== undefined)? options.frequency : 10000;

    // successCallback required
    if (typeof successCallback !== "function") {
        console.log("Accelerometer Error: successCallback is not a function");
        return;
    }

    // errorCallback optional
    if (errorCallback && (typeof errorCallback !== "function")) {
        console.log("Accelerometer Error: errorCallback is not a function");
        return;
    }

    // Make sure accelerometer timeout > frequency + 10 sec
    PhoneGap.exec(
        function(timeout) {
            if (timeout < (frequency + 10000)) {
                PhoneGap.exec(null, null, "Accelerometer", "setTimeout",
                    [frequency + 10000]);
            }
        },
    ),
```



```

        function(e) { }, "Accelerometer", "getTimeout", []);

    // Start watch timer
    var id = PhoneGap.createUUID();
    navigator.accelerometer.timers[id] = setInterval(function() {
        PhoneGap.exec(successCallback, errorCallback, "Accelerometer",
            "getAcceleration", []);
    }, (frequency ? frequency : 1));

    return id;
};

/**
 * Clears the specified accelerometer watch.
 *
 * @param {String} id      The id of the watch returned from
 *                          #watchAcceleration.
 */
Accelerometer.prototype.clearWatch = function(id) {

    // Stop javascript timer & remove from timer list
    if (id && navigator.accelerometer.timers[id] !== undefined) {
        clearInterval(navigator.accelerometer.timers[id]);
        delete navigator.accelerometer.timers[id];
    }
};

PhoneGap.addConstructor(function() {
    if (typeof navigator.accelerometer === "undefined") {
        navigator.accelerometer = new Accelerometer();
    }
});
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("app")) {
    PhoneGap.addResource("app");
    (function() {

    /**
     * Constructor
     * @constructor
     */
    var App = function() {};

    /**

```

```
* Clear the resource cache.
*/
App.prototype.clearCache = function() {
    PhoneGap.exec(null, null, "App", "clearCache", []);
};

/**
 * Load the url into the webview.
 *
 * @param url          The URL to load
 * @param props        Properties that can be passed in to the activity:
 *     wait: int        => wait msec before loading URL
 *     loadingDialog: "Title,Message" => display a native loading dialog
 *     hideLoadingDialogOnPage: boolean => hide loadingDialog when page
 *                                     loaded instead of when
 *                                     deviceready event occurs.
 *     loadInWebView: boolean => cause all links on web page to
 *                               be loaded into existing web
 *                               view, instead of being loaded
 *                               into new browser.
 *     loadUrlTimeoutValue: int => time in msec to wait before
 *                               triggering a timeout error
 *     errorUrl: URL        => URL to load if there's an error
 *                               loading specified URL with
 *                               loadUrl(). Should be a local
 *                               URL such as
 *                               file:///
 *                               android_asset/www/error.html");
 *     keepRunning: boolean => enable app to keep running in
 *                               background
 *
 * Example:
 *     App app = new App();
 *     app.loadUrl("http://server/myapp/index.html", {wait:2000,
 *     loadingDialog:"Wait,Loading App", loadUrlTimeoutValue: 60000});
 */
App.prototype.loadUrl = function(url, props) {
    PhoneGap.exec(null, null, "App", "loadUrl", [url, props]);
};

/**
 * Cancel loadUrl that is waiting to be loaded.
 */
App.prototype.cancelLoadUrl = function() {
    PhoneGap.exec(null, null, "App", "cancelLoadUrl", []);
};

/**
 * Clear web history in this web view.
 * Instead of BACK button loading the previous web page, it will exit the app.
 */
App.prototype.clearHistory = function() {
    PhoneGap.exec(null, null, "App", "clearHistory", []);
};

/**
```

```

    * Override the default behavior of the Android back button.
    * If overridden, when the back button is pressed, the "backKeyDown" JavaScript
    * event will be fired.
    *
    * Note: The user should not have to call this method. Instead, when the user
    *       registers for the "backbutton" event, this is automatically done.
    *
    * @param override      T=override, F=cancel override
    */
App.prototype.overrideBackbutton = function(override) {
    PhoneGap.exec(null, null, "App", "overrideBackbutton", [override]);
};

/**
 * Exit and terminate the application.
 */
App.prototype.exitApp = function() {
    return PhoneGap.exec(null, null, "App", "exitApp", []);
};

PhoneGap.addConstructor(function() {
    navigator.app = new App();
});
})();
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("camera")) {
    PhoneGap.addResource("camera");

    /**
     * This class provides access to the device camera.
     *
     * @constructor
     */
    var Camera = function() {
        this.successCallback = null;
        this.errorCallback = null;
        this.options = null;
    };

    /**
     * Format of image that returned from getPicture.
     *
     * Example: navigator.camera.getPicture(success, fail,

```

```
*           { quality: 80,
*             destinationType: Camera.DestinationType.DATA_URL,
*             sourceType: Camera.PictureSourceType.PHOTOLIBRARY}}
*/
Camera.DestinationType = {
  DATA_URL: 0,           // Return base64 encoded string
  FILE_URI: 1             // Return file uri
                        // (content://media/external/images/
                        // media/2 for Android)
};
Camera.prototype.DestinationType = Camera.DestinationType;

/**
 * Encoding of image returned from getPicture.
 *
 * Example: navigator.camera.getPicture(success, fail,
 *           { quality: 80,
 *             destinationType: Camera.DestinationType.DATA_URL,
 *             sourceType: Camera.PictureSourceType.CAMERA,
 *             encodingType: Camera.EncodingType.PNG})
 */
Camera.EncodingType = {
  JPEG: 0,                // Return JPEG encoded image
  PNG: 1                  // Return PNG encoded image
};
Camera.prototype.EncodingType = Camera.EncodingType;

/**
 * Source to getPicture from.
 *
 * Example: navigator.camera.getPicture(success, fail,
 *           { quality: 80,
 *             destinationType: Camera.DestinationType.DATA_URL,
 *             sourceType: Camera.PictureSourceType.PHOTOLIBRARY}})
 */
Camera.PictureSourceType = {
  PHOTOLIBRARY : 0,       // Choose image from picture library (same as
                        // SAVEDPHOTOALBUM for Android)
  CAMERA : 1,             // Take picture from camera
  SAVEDPHOTOALBUM : 2     // Choose image from picture library (same as
                        // PHOTOLIBRARY for Android)
};
Camera.prototype.PictureSourceType = Camera.PictureSourceType;

/**
 * Gets a picture from source defined by "options.sourceType", and returns the
 * image as defined by the "options.destinationType" option.
 *
 * The defaults are sourceType=CAMERA and destinationType=DATA_URL.
 *
 * @param {Function} successCallback
 * @param {Function} errorCallback
 * @param {Object} options
 */
```

```

Camera.prototype.getPicture = function(successCallback, errorCallback, options) {

    // successCallback required
    if (typeof successCallback !== "function") {
        console.log("Camera Error: successCallback is not a function");
        return;
    }

    // errorCallback optional
    if (errorCallback && (typeof errorCallback !== "function")) {
        console.log("Camera Error: errorCallback is not a function");
        return;
    }

    this.options = options;
    var quality = 80;
    if (options.quality) {
        quality = this.options.quality;
    }

    var maxResolution = 0;
    if (options.maxResolution) {
        maxResolution = this.options.maxResolution;
    }

    var destinationType = Camera.DestinationType.DATA_URL;
    if (this.options.destinationType) {
        destinationType = this.options.destinationType;
    }
    var sourceType = Camera.PictureSourceType.CAMERA;
    if (typeof this.options.sourceType === "number") {
        sourceType = this.options.sourceType;
    }
    var encodingType = Camera.EncodingType.JPEG;
    if (typeof options.encodingType == "number") {
        encodingType = this.options.encodingType;
    }

    var targetWidth = -1;
    if (typeof options.targetWidth == "number") {
        targetWidth = options.targetWidth;
    } else if (typeof options.targetWidth == "string") {
        var width = new Number(options.targetWidth);
        if (isNaN(width) === false) {
            targetWidth = width.valueOf();
        }
    }

    var targetHeight = -1;
    if (typeof options.targetHeight == "number") {
        targetHeight = options.targetHeight;
    } else if (typeof options.targetHeight == "string") {
        var height = new Number(options.targetHeight);
        if (isNaN(height) === false) {

```

```
        targetHeight = height.valueOf();
    }
}

PhoneGap.exec(successCallback, errorCallback, "Camera", "takePicture",
    [quality, destinationType, sourceType, targetWidth,
    targetHeight, encodingType]);
};

PhoneGap.addConstructor(function() {
    if (typeof navigator.camera === "undefined") {
        navigator.camera = new Camera();
    }
});
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("capture")) {
    PhoneGap.addResource("capture");
}

/**
 * Represents a single file.
 *
 * *
 * name {DOMString} name of the file, without path information
 * fullPath {DOMString} the full path of the file, including the name
 * type {DOMString} mime type
 * lastModifiedDate {Date} last modified date
 * size {Number} size of the file in bytes
 */
var MediaFile = function(name, fullPath, type, lastModifiedDate, size){
    this.name = name || null;
    this.fullPath = fullPath || null;
    this.type = type || null;
    this.lastModifiedDate = lastModifiedDate || null;
    this.size = size || 0;
};

/**
 * Launch device camera application for recording video(s).
 *
 * *
 * @param {Function} successCB
 * @param {Function} errorCallback
 */
MediaFile.prototype.getFormatData = function(successCallback, errorCallback){
    PhoneGap.exec(successCallback, errorCallback, "Capture", "getFormatData",
```

```

        [this.fullPath, this.type]);
    };

    /**
     * MediaFileData encapsulates format information of a media file.
     *
     * @param {DOMString} codecs
     * @param {long} bitrate
     * @param {long} height
     * @param {long} width
     * @param {float} duration
     */
    var MediaFileData = function(codecs, bitrate, height, width, duration){
        this.codecs = codecs || null;
        this.bitrate = bitrate || 0;
        this.height = height || 0;
        this.width = width || 0;
        this.duration = duration || 0;
    };

    /**
     * The CaptureError interface encapsulates all errors in the Capture API.
     */
    var CaptureError = function(){
        this.code = null;
    };

    // Capture error codes
    CaptureError.CAPTURE_INTERNAL_ERR = 0;
    CaptureError.CAPTURE_APPLICATION_BUSY = 1;
    CaptureError.CAPTURE_INVALID_ARGUMENT = 2;
    CaptureError.CAPTURE_NO_MEDIA_FILES = 3;
    CaptureError.CAPTURE_NOT_SUPPORTED = 20;

    /**
     * The Capture interface exposes an interface to the camera and microphone of
     * the hosting device.
     */
    var Capture = function(){
        this.supportedAudioModes = [];
        this.supportedImageModes = [];
        this.supportedVideoModes = [];
    };

    /**
     * Launch audio recorder application for recording audio clip(s).
     *
     * @param {Function} successCB
     * @param {Function} errorCallback
     * @param {CaptureAudioOptions} options
     */
    Capture.prototype.captureAudio = function(successCallback, errorCallback, options){
        PhoneGap.exec(successCallback, errorCallback, "Capture", "captureAudio",
            [options]);
    };

```

```
};

/**
 * Launch camera application for taking image(s).
 *
 * @param {Function} successCB
 * @param {Function} errorCallback
 * @param {CaptureImageOptions} options
 */
Capture.prototype.captureImage = function(successCallback, errorCallback, options){
    PhoneGap.exec(successCallback, errorCallback, "Capture", "captureImage",
        [options]);
};

/**
 * Launch camera application for taking image(s).
 *
 * @param {Function} successCB
 * @param {Function} errorCallback
 * @param {CaptureImageOptions} options
 */
Capture.prototype._castMediaFile = function(pluginResult){
    var mediaFiles = [];
    var i;
    for (i = 0; i < pluginResult.message.length; i++) {
        var mediaFile = new MediaFile();
        mediaFile.name = pluginResult.message[i].name;
        mediaFile.fullPath = pluginResult.message[i].fullPath;
        mediaFile.type = pluginResult.message[i].type;
        mediaFile.lastModifiedDate =
            pluginResult.message[i].lastModifiedDate;
        mediaFile.size = pluginResult.message[i].size;
        mediaFiles.push(mediaFile);
    }
    pluginResult.message = mediaFiles;
    return pluginResult;
};

/**
 * Launch device camera application for recording video(s).
 *
 * @param {Function} successCB
 * @param {Function} errorCallback
 * @param {CaptureVideoOptions} options
 */
Capture.prototype.captureVideo = function(successCallback, errorCallback, options){
    PhoneGap.exec(successCallback, errorCallback, "Capture", "captureVideo",
        [options]);
};

/**
 * Encapsulates a set of parameters that the capture device supports.
 */
var ConfigurationData = function(){
```



```

        // The ASCII-encoded string in lower case representing the media type.
        this.type = null;
        // The height attribute represents height of the image or video in pixels.
        // In the case of a sound clip this attribute has value 0.
        this.height = 0;
        // The width attribute represents width of the image or video in pixels.
        // In the case of a sound clip this attribute has value 0
        this.width = 0;
    };

    /**
     * Encapsulates all image capture operation configuration options.
     */
    var CaptureImageOptions = function(){
        // Upper limit of images user can take. Value must be equal or greater than 1.
        this.limit = 1;
        // The selected image mode. Must match with one of the elements in
        // supportedImageModes array.
        this.mode = null;
    };

    /**
     * Encapsulates all video capture operation configuration options.
     */
    var CaptureVideoOptions = function(){
        // Upper limit of videos user can record. Value must be equal or greater
        // than 1.
        this.limit = 1;
        // Maximum duration of a single video clip in seconds.
        this.duration = 0;
        // The selected video mode. Must match with one of the elements in
        // supportedVideoModes array.
        this.mode = null;
    };

    /**
     * Encapsulates all audio capture operation configuration options.
     */
    var CaptureAudioOptions = function(){
        // Upper limit of sound clips user can record. Value must be equal or
        // greater than 1.
        this.limit = 1;
        // Maximum duration of a single sound clip in seconds.
        this.duration = 0;
        // The selected audio mode. Must match with one of the elements in
        // supportedAudioModes array.
        this.mode = null;
    };

    PhoneGap.addConstructor(function(){
        if (typeof navigator.device === "undefined") {
            navigator.device = window.device = new Device();
        }
        if (typeof navigator.device.capture === "undefined") {

```

```
        navigator.device.capture = window.device.capture = new Capture();
    }
});
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("compass")) {
PhoneGap.addResource("compass");

/**
 * This class provides access to device Compass data.
 * @constructor
 */
var Compass = function() {
    /**
     * The last known Compass position.
     */
    this.lastHeading = null;

    /**
     * List of compass watch timers
     */
    this.timers = {};
};

Compass.ERROR_MSG = ["Not running", "Starting", "", "Failed to start"];

/**
 * Asynchronously acquires the current heading.
 *
 * @param {Function} successCallback The function to call when the heading data
 * is available
 * @param {Function} errorCallback The function to call when there is an
 * error getting the heading data. (OPTIONAL)
 * @param {PositionOptions} options The options for getting the heading data
 * such as timeout. (OPTIONAL)
 */
Compass.prototype.getCurrentHeading = function(successCallback, errorCallback,
    options) {

    // successCallback required
    if (typeof successCallback !== "function") {
        console.log("Compass Error: successCallback is not a function");
        return;
    }

    // errorCallback optional
```

```

    if (errorCallback && (typeof errorCallback !== "function")) {
        console.log("Compass Error: errorCallback is not a function");
        return;
    }

    // Get heading
    PhoneGap.exec(successCallback, errorCallback, "Compass", "getHeading", []);
};

/**
 * Asynchronously acquires the heading repeatedly at a given interval.
 *
 * @param {Function} successCallback The function to call each time the
 * heading data is available
 * @param {Function} errorCallback The function to call when there is an
 * error getting the heading data.
 * (OPTIONAL)
 * @param {HeadingOptions} options The options for getting the heading
 * data such as timeout and the frequency
 * of the watch. (OPTIONAL)
 * @return String The watch id that must be passed to
 * #clearWatch to stop watching.
 */
Compass.prototype.watchHeading= function(successCallback, errorCallback, options) {

    // Default interval (100 msec)
    var frequency = (options !== undefined) ? options.frequency : 100;

    // successCallback required
    if (typeof successCallback !== "function") {
        console.log("Compass Error: successCallback is not a function");
        return;
    }

    // errorCallback optional
    if (errorCallback && (typeof errorCallback !== "function")) {
        console.log("Compass Error: errorCallback is not a function");
        return;
    }

    // Make sure compass timeout > frequency + 10 sec
    PhoneGap.exec(
        function(timeout) {
            if (timeout < (frequency + 10000)) {
                PhoneGap.exec(null, null, "Compass", "setTimeout", [frequency +
                    10000]);
            }
        },
        function(e) { }, "Compass", "getTimeout", []);

    // Start watch timer to get headings
    var id = PhoneGap.createUUID();
    navigator.compass.timers[id] = setInterval(
        function() {

```

```
        PhoneGap.exec(successCallback, errorCallback, "Compass",
            "getHeading", []);
    }, (frequency ? frequency : 1));

    return id;
};

/**
 * Clears the specified heading watch.
 *
 * @param {String} id      The ID of the watch returned from #watchHeading.
 */
Compass.prototype.clearWatch = function(id) {

    // Stop javascript timer & remove from timer list
    if (id && navigator.compass.timers[id]) {
        clearInterval(navigator.compass.timers[id]);
        delete navigator.compass.timers[id];
    }
};

PhoneGap.addConstructor(function() {
    if (typeof navigator.compass === "undefined") {
        navigator.compass = new Compass();
    }
});
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("contact")) {
    PhoneGap.addResource("contact");

    /**
     * Contains information about a single contact.
     * @constructor
     * @param {DOMString} id unique identifier
     * @param {DOMString} displayName
     * @param {ContactName} name
     * @param {DOMString} nickname
     * @param {Array.<ContactField>} phoneNumbers array of phone numbers
     * @param {Array.<ContactField>} emails array of email addresses
     * @param {Array.<ContactAddress>} addresses array of addresses
     * @param {Array.<ContactField>} ims instant messaging user ids
     * @param {Array.<ContactOrganization>} organizations
    */
}
```

```

* @param {DOMString} birthday contact's birthday
* @param {DOMString} note user notes about contact
* @param {Array.<ContactField>} photos
* @param {Array.<ContactField>} categories
* @param {Array.<ContactField>} urls contact's web sites
*/
var Contact = function (id, displayName, name, nickname, phoneNumbers, emails,
    addresses,
    ims, organizations, birthday, note, photos, categories, urls) {
    this.id = id || null;
    this.rawId = null;
    this.displayName = displayName || null;
    this.name = name || null; // ContactName
    this.nickname = nickname || null;
    this.phoneNumbers = phoneNumbers || null; // ContactField[]
    this.emails = emails || null; // ContactField[]
    this.addresses = addresses || null; // ContactAddress[]
    this.ims = ims || null; // ContactField[]
    this.organizations = organizations || null; // ContactOrganization[]
    this.birthday = birthday || null;
    this.note = note || null;
    this.photos = photos || null; // ContactField[]
    this.categories = categories || null; // ContactField[]
    this.urls = urls || null; // ContactField[]
};

/**
 * ContactError.
 * An error code assigned by an implementation when an error has occurreds
 * @constructor
 */
var ContactError = function() {
    this.code=null;
};

/**
 * Error codes
 */
ContactError.UNKNOWN_ERROR = 0;
ContactError.INVALID_ARGUMENT_ERROR = 1;
ContactError.TIMEOUT_ERROR = 2;
ContactError.PENDING_OPERATION_ERROR = 3;
ContactError.IO_ERROR = 4;
ContactError.NOT_SUPPORTED_ERROR = 5;
ContactError.PERMISSION_DENIED_ERROR = 20;

/**
 * Removes contact from device storage.
 * @param successCB success callback
 * @param errorCB error callback
 */
Contact.prototype.remove = function(successCB, errorCB) {
    if (this.id === null) {
        var errorObj = new ContactError();

```

```
        errorObj.code = ContactError.UNKNOWN_ERROR;
        errorCB(errorObj);
    }
    else {
        PhoneGap.exec(successCB, errorCB, "Contacts", "remove", [this.id]);
    }
};

/**
 * Creates a deep copy of this Contact.
 * With the contact ID set to null.
 * @return copy of this Contact
 */
Contact.prototype.clone = function() {
    var clonedContact = PhoneGap.clone(this);
    var i;
    clonedContact.id = null;
    clonedContact.rawId = null;
    // Loop through and clear out any id's in phones, emails, etc.
    if (clonedContact.phoneNumbers) {
        for (i = 0; i < clonedContact.phoneNumbers.length; i++) {
            clonedContact.phoneNumbers[i].id = null;
        }
    }
    if (clonedContact.emails) {
        for (i = 0; i < clonedContact.emails.length; i++) {
            clonedContact.emails[i].id = null;
        }
    }
    if (clonedContact.addresses) {
        for (i = 0; i < clonedContact.addresses.length; i++) {
            clonedContact.addresses[i].id = null;
        }
    }
    if (clonedContact.ims) {
        for (i = 0; i < clonedContact.ims.length; i++) {
            clonedContact.ims[i].id = null;
        }
    }
    if (clonedContact.organizations) {
        for (i = 0; i < clonedContact.organizations.length; i++) {
            clonedContact.organizations[i].id = null;
        }
    }
    if (clonedContact.tags) {
        for (i = 0; i < clonedContact.tags.length; i++) {
            clonedContact.tags[i].id = null;
        }
    }
    if (clonedContact.photos) {
        for (i = 0; i < clonedContact.photos.length; i++) {
            clonedContact.photos[i].id = null;
        }
    }
}
```

```

        if (clonedContact.urls) {
            for (i = 0; i < clonedContact.urls.length; i++) {
                clonedContact.urls[i].id = null;
            }
        }
        return clonedContact;
    };

    /**
     * Persists contact to device storage.
     * @param successCB success callback
     * @param errorCB error callback
     */
    Contact.prototype.save = function(successCB, errorCB) {
        PhoneGap.exec(successCB, errorCB, "Contacts", "save", [this]);
    };

    /**
     * Contact name.
     * @constructor
     * @param formatted
     * @param familyName
     * @param givenName
     * @param middle
     * @param prefix
     * @param suffix
     */
    var ContactName = function(formatted, familyName, givenName, middle, prefix,
        suffix) {
        this.formatted = formatted || null;
        this.familyName = familyName || null;
        this.givenName = givenName || null;
        this.middleName = middle || null;
        this.honorificPrefix = prefix || null;
        this.honorificSuffix = suffix || null;
    };

    /**
     * Generic contact field.
     * @constructor
     * @param {DOMString} id unique identifier, should only be set by native code
     * @param type
     * @param value
     * @param pref
     */
    var ContactField = function(type, value, pref) {
        this.id = null;
        this.type = type || null;
        this.value = value || null;
        this.pref = pref || null;
    };

    /**

```

```
* Contact address.
* @constructor
* @param {DOMString} id unique identifier, should only be set by native code
* @param formatted
* @param streetAddress
* @param locality
* @param region
* @param postalCode
* @param country
*/
var ContactAddress = function(pref, type, formatted, streetAddress, locality,
    region, postalCode, country) {
    this.id = null;
    this.pref = pref || null;
    this.type = type || null;
    this.formatted = formatted || null;
    this.streetAddress = streetAddress || null;
    this.locality = locality || null;
    this.region = region || null;
    this.postalCode = postalCode || null;
    this.country = country || null;
};

/**
* Contact organization.
* @constructor
* @param {DOMString} id unique identifier, should only be set by native code
* @param name
* @param dept
* @param title
* @param startDate
* @param endDate
* @param location
* @param desc
*/
var ContactOrganization = function(pref, type, name, dept, title) {
    this.id = null;
    this.pref = pref || null;
    this.type = type || null;
    this.name = name || null;
    this.department = dept || null;
    this.title = title || null;
};

/**
* Represents a group of Contacts.
* @constructor
*/
var Contacts = function() {
    this.inProgress = false;
    this.records = [];
};
/**
```



```

* Returns an array of Contacts matching the search criteria.
* @param fields that should be searched
* @param successCB success callback
* @param errorCallback error callback
* @param {ContactFindOptions} options that can be applied to contact searching
*                                     @return array of Contacts matching search
*                                     criteria
*/
Contacts.prototype.find = function(fields, successCB, errorCallback, options) {
    if (successCB === null) {
        throw new TypeError("You must specify a success callback for the find
            command.");
    }
    if (fields === null || fields === "undefined" || fields.length ===
        "undefined" || fields.length <= 0) {
        if (typeof errorCallback === "function") {
            errorCallback({"code": ContactError.INVALID_ARGUMENT_ERROR});
        }
    } else {
        PhoneGap.exec(successCB, errorCallback, "Contacts", "search", [fields,
            options]);
    }
};

/**
* This function creates a new contact, but it does not persist the contact
* to device storage. To persist the contact to device storage, invoke
* contact.save().
* @param properties an object who's properties will be examined to create a new
* Contact
* @returns new Contact object
*/
Contacts.prototype.create = function(properties) {
    var i;
    var contact = new Contact();
    for (i in properties) {
        if (contact[i] !== 'undefined') {
            contact[i] = properties[i];
        }
    }
    return contact;
};

/**
* This function returns and array of contacts. It is required as we need to
* convert raw
* JSON objects into concrete Contact objects. Currently this method is called
* after
* navigator.contacts.find but before the find methods success call back.
*
* @param jsonArray an array of JSON Objects that need to be converted to
* Contact objects.
* @returns an array of Contact objects
*/

```

```
Contacts.prototype.cast = function(pluginResult) {
    var contacts = [];
    var i;
    for (i=0; i<pluginResult.message.length; i++) {
        contacts.push(navigator.contacts.create(pluginResult.message[i]));
    }
    pluginResult.message = contacts;
    return pluginResult;
};

/**
 * ContactFindOptions.
 * @constructor
 * @param filter used to match contacts against
 * @param multiple boolean used to determine if more than one contact should be
 * returned
 */
var ContactFindOptions = function(filter, multiple) {
    this.filter = filter || '';
    this.multiple = multiple || false;
};

/**
 * Add the contact interface into the browser.
 */
PhoneGap.addConstructor(function() {
    if(typeof navigator.contacts === "undefined") {
        navigator.contacts = new Contacts();
    }
});

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

// TODO: Needs to be commented

if (!PhoneGap.hasResource("crypto")) {
    PhoneGap.addResource("crypto");

    /**
     * @constructor
     */
    var Crypto = function() {
    };

    Crypto.prototype.encrypt = function(seed, string, callback) {
```

```

        this.encryptWin = callback;
        PhoneGap.exec(null, null, "Crypto", "encrypt", [seed, string]);
    };

    Crypto.prototype.decrypt = function(seed, string, callback) {
        this.decryptWin = callback;
        PhoneGap.exec(null, null, "Crypto", "decrypt", [seed, string]);
    };

    Crypto.prototype.getEncryptedString = function(string) {
        this.encryptWin(string);
    };

    Crypto.prototype.getPlainString = function(string) {
        this.decryptWin(string);
    };

    PhoneGap.addConstructor(function() {
        if (typeof navigator.Crypto === "undefined") {
            navigator.Crypto = new Crypto();
        }
    });
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("device")) {
    PhoneGap.addResource("device");

    /**
     * This represents the mobile device, and provides properties for inspecting
     * the model, version, UUID of the phone, etc.
     * @constructor
     */
    var Device = function() {
        this.available = PhoneGap.available;
        this.platform = null;
        this.version = null;
        this.name = null;
        this.uuid = null;
        this.phonegap = null;

        var me = this;
        this.getInfo(
            function(info) {
                me.available = true;
            }
        );
    };

```

```
        me.platform = info.platform;
        me.version = info.version;
        me.name = info.name;
        me.uuid = info.uuid;
        me.phonegap = info.phonegap;
        PhoneGap.onPhoneGapInfoReady.fire();
    },
    function(e) {
        me.available = false;
        console.log("Error initializing PhoneGap: " + e);
        alert("Error initializing PhoneGap: "+e);
    });
};

/**
 * Get device info
 *
 * @param {Function} successCallback The function to call when the heading data
 *                                   is available
 * @param {Function} errorCallback The function to call when there is an error
 *                                   getting the heading data. (OPTIONAL)
 */
Device.prototype.getInfo = function(successCallback, errorCallback) {

    // successCallback required
    if (typeof successCallback !== "function") {
        console.log("Device Error: successCallback is not a function");
        return;
    }

    // errorCallback optional
    if (errorCallback && (typeof errorCallback !== "function")) {
        console.log("Device Error: errorCallback is not a function");
        return;
    }

    // Get info
    PhoneGap.exec(successCallback, errorCallback, "Device", "getDeviceInfo", []);
};

/**
 * DEPRECATED
 * This is only for Android.
 *
 * You must explicitly override the back button.
 */
Device.prototype.overrideBackButton = function() {
    console.log("Device.overrideBackButton() is deprecated. Use
        App.overrideBackbutton(true).");
    navigator.app.overrideBackbutton(true);
};

/**
 * DEPRECATED
 * This is only for Android.
```

```

    *
    * This resets the back button to the default behaviour
    */
Device.prototype.resetBackButton = function() {
    console.log("Device.resetBackButton() is deprecated. Use
        App.overrideBackbutton(false).");
    navigator.app.overrideBackbutton(false);
};

/*
 * DEPRECATED
 * This is only for Android.
 *
 * This terminates the activity!
 */
Device.prototype.exitApp = function() {
    console.log("Device.exitApp() is deprecated. Use App.exitApp().");
    navigator.app.exitApp();
};

PhoneGap.addConstructor(function() {
    if (typeof navigator.device === "undefined") {
        navigator.device = window.device = new Device();
    }
});

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("file")) {
    PhoneGap.addResource("file");

    /**
     * This class provides some useful information about a file.
     * This is the fields returned when navigator.fileMgr.getFileProperties()
     * is called.
     * @constructor
     */
    var FileProperties = function(filePath) {
        this.filePath = filePath;
        this.size = 0;
        this.lastModifiedDate = null;
    };

    /**
     * Represents a single file.
     */

```

```
* @constructor
* @param name {DOMString} name of the file, without path information
* @param fullPath {DOMString} the full path of the file, including the name
* @param type {DOMString} mime type
* @param lastModifiedDate {Date} last modified date
* @param size {Number} size of the file in bytes
*/
var File = function(name, fullPath, type, lastModifiedDate, size) {
    this.name = name || null;
    this.fullPath = fullPath || null;
    this.type = type || null;
    this.lastModifiedDate = lastModifiedDate || null;
    this.size = size || 0;
};

/** @constructor */
var FileError = function() {
    this.code = null;
};

// File error codes
// Found in DOMException
FileError.NOT_FOUND_ERR = 1;
FileError.SECURITY_ERR = 2;
FileError.ABORT_ERR = 3;

// Added by this specification
FileError.NOT_READABLE_ERR = 4;
FileError.ENCODING_ERR = 5;
FileError.NO_MODIFICATION_ALLOWED_ERR = 6;
FileError.INVALID_STATE_ERR = 7;
FileError.SYNTAX_ERR = 8;
FileError.INVALID_MODIFICATION_ERR = 9;
FileError.QUOTA_EXCEEDED_ERR = 10;
FileError.TYPE_MISMATCH_ERR = 11;
FileError.PATH_EXISTS_ERR = 12;

//-----
// File manager
//-----

/** @constructor */
var FileMgr = function() {
};

FileMgr.prototype.getFileProperties = function(filePath) {
    return PhoneGap.exec(null, null, "File", "getFileProperties", [filePath]);
};

FileMgr.prototype.getFileBasePaths = function() {
};

FileMgr.prototype.testSaveLocationExists = function(successCallback,
    errorCallback) {
```

```

        return PhoneGap.exec(successCallback, errorCallback, "File",
            "testSaveLocationExists", []);
    };

    FileMgr.prototype.testFileExists = function(fileName, successCallback,
        errorCallback) {
        return PhoneGap.exec(successCallback, errorCallback, "File",
            "testFileExists", [fileName]);
    };

    FileMgr.prototype.testDirectoryExists = function(dirName, successCallback,
        errorCallback) {
        return PhoneGap.exec(successCallback, errorCallback, "File",
            "testDirectoryExists", [dirName]);
    };

    FileMgr.prototype.getFreeDiskSpace = function(successCallback, errorCallback) {
        return PhoneGap.exec(successCallback, errorCallback, "File",
            "getFreeDiskSpace", []);
    };

    FileMgr.prototype.write = function(fileName, data, position, successCallback,
        errorCallback) {
        PhoneGap.exec(successCallback, errorCallback, "File", "write", [fileName,
            data, position]);
    };

    FileMgr.prototype.truncate = function(fileName, size, successCallback,
        errorCallback) {
        PhoneGap.exec(successCallback, errorCallback, "File", "truncate",
            [fileName, size]);
    };

    FileMgr.prototype.readAsText = function(fileName, encoding, successCallback,
        errorCallback) {
        PhoneGap.exec(successCallback, errorCallback, "File", "readAsText",
            [fileName, encoding]);
    };

    FileMgr.prototype.readAsDataURL = function(fileName, successCallback,
        errorCallback) {
        PhoneGap.exec(successCallback, errorCallback, "File", "readAsDataURL",
            [fileName]);
    };

    PhoneGap.addConstructor(function() {
        if (typeof navigator.fileMgr === "undefined") {
            navigator.fileMgr = new FileMgr();
        }
    });

    //-----
    // File Reader
    //-----
    // TODO: All other FileMgr function operate on the SD card as root. However,

```

```
//      for FileReader & FileWriter the root is not SD card.  Should this be
// changed?

/**
 * This class reads the mobile device file system.
 *
 * For Android:
 *      The root directory is the root of the file system.
 *      To read from the SD card, the file name is "sdcard/my_file.txt"
 * @constructor
 */
var FileReader = function() {
    this.fileName = "";

    this.readyState = 0;

    // File data
    this.result = null;

    // Error
    this.error = null;

    // Event handlers
    this.onloadstart = null;    // When the read starts.
    this.onprogress = null;    // While reading (and decoding) file or
                                // fileBlob data, and reporting partial file
                                // data (progress.loaded/progress.total)
    this.onload = null;        // When the read has successfully completed.
    this.onerror = null;       // When the read has failed (see errors).
    this.onloadend = null;     // When the request has completed (either in
                                // success or failure).
    this.onabort = null;       // When the read has been aborted. For
                                // instance, by invoking the abort() method.
};

// States
FileReader.EMPTY = 0;
FileReader.LOADING = 1;
FileReader.DONE = 2;

/**
 * Abort reading file.
 */
FileReader.prototype.abort = function() {
    var evt;
    this.readyState = FileReader.DONE;
    this.result = null;

    // set error
    var error = new FileError();
    error.code = error.ABORT_ERR;
    this.error = error;

    // If error callback
    if (typeof this.onerror === "function") {
```



```

        this.onerror({"type":"error", "target":this});
    }
    // If abort callback
    if (typeof this.onabort === "function") {
        this.onabort({"type":"abort", "target":this});
    }
    // If load end callback
    if (typeof this.onloadend === "function") {
        this.onloadend({"type":"loadend", "target":this});
    }
};

/**
 * Read text file.
 *
 * @param file      {File} File object containing file properties
 * @param encoding  [Optional] (see
 *                  http://www.iana.org/assignments/character-sets)
 */
FileReader.prototype.readAsText = function(file, encoding) {
    this.fileName = "";
    if (typeof file.fullPath === "undefined") {
        this.fileName = file;
    } else {
        this.fileName = file.fullPath;
    }

    // LOADING state
    this.readyState = FileReader.LOADING;

    // If loadstart callback
    if (typeof this.onloadstart === "function") {
        this.onloadstart({"type":"loadstart", "target":this});
    }

    // Default encoding is UTF-8
    var enc = encoding ? encoding : "UTF-8";

    var me = this;

    // Read file
    navigator.fileMgr.readAsText(this.fileName, enc,

        // Success callback
        function(r) {
            var evt;

            // If DONE (cancelled), then don't do anything
            if (me.readyState === FileReader.DONE) {
                return;
            }

            // Save result
            me.result = r;

            // If onload callback

```

```
        if (typeof me.onload === "function") {
            me.onload({"type": "load", "target": me});
        }

        // DONE state
        me.readyState = FileReader.DONE;

        // If onloadend callback
        if (typeof me.onloadend === "function") {
            me.onloadend({"type": "loadend", "target": me});
        }
    },

    // Error callback
    function(e) {
        var evt;
        // If DONE (cancelled), then don't do anything
        if (me.readyState === FileReader.DONE) {
            return;
        }

        // Save error
        me.error = e;

        // If onerror callback
        if (typeof me.onerror === "function") {
            me.onerror({"type": "error", "target": me});
        }

        // DONE state
        me.readyState = FileReader.DONE;

        // If onloadend callback
        if (typeof me.onloadend === "function") {
            me.onloadend({"type": "loadend", "target": me});
        }
    }
    );
};

/**
 * Read file and return data as a base64 encoded data url.
 * A data url is of the form:
 *   data:[<mediatype>][;base64],<data>
 *
 * @param file      {File} File object containing file properties
 */
FileReader.prototype.readAsDataURL = function(file) {
    this.fileName = "";
    if (typeof file.fullPath === "undefined") {
        this.fileName = file;
    } else {
        this.fileName = file.fullPath;
    }
};
```

```

    }

    // LOADING state
    this.readyState = FileReader.LOADING;

    // If loadstart callback
    if (typeof this.onloadstart === "function") {
        this.onloadstart({"type":"loadstart", "target":this});
    }

    var me = this;

    // Read file
    navigator.fileMgr.readAsDataURL(this.fileName,

        // Success callback
        function(r) {
            var evt;

            // If DONE (cancelled), then don't do anything
            if (me.readyState === FileReader.DONE) {
                return;
            }

            // Save result
            me.result = r;

            // If onload callback
            if (typeof me.onload === "function") {
                me.onload({"type":"load", "target":me});
            }

            // DONE state
            me.readyState = FileReader.DONE;

            // If onloadend callback
            if (typeof me.onloadend === "function") {
                me.onloadend({"type":"loadend", "target":me});
            }
        },

        // Error callback
        function(e) {
            var evt;
            // If DONE (cancelled), then don't do anything
            if (me.readyState === FileReader.DONE) {
                return;
            }

            // Save error
            me.error = e;

            // If onerror callback
            if (typeof me.onerror === "function") {

```

```
        me.onerror({"type":"error", "target":me});
    }

    // DONE state
    me.readyState = FileReader.DONE;

    // If onloadend callback
    if (typeof me.onloadend === "function") {
        me.onloadend({"type":"loadend", "target":me});
    }
    };
};

/**
 * Read file and return data as a binary data.
 *
 * @param file      {File} File object containing file properties
 */
FileReader.prototype.readAsBinaryString = function(file) {
    // TODO - Can't return binary data to browser.
    this.fileName = file;
};

/**
 * Read file and return data as a binary data.
 *
 * @param file      {File} File object containing file properties
 */
FileReader.prototype.readAsArrayBuffer = function(file) {
    // TODO - Can't return binary data to browser.
    this.fileName = file;
};

//-----
// File Writer
//-----

/**
 * This class writes to the mobile device file system.
 *
 * For Android:
 *     The root directory is the root of the file system.
 *     To write to the SD card, the file name is "sdcard/my_file.txt"
 *
 * @constructor
 * @param file {File} File object containing file properties
 * @param append if true write to the end of the file, otherwise overwrite the file
 */
var FileWriter = function(file) {
    this.fileName = "";
    this.length = 0;
    if (file) {
        this.fileName = file.fullPath || file;
        this.length = file.size || 0;
    }
};
```

```

    // default is to write at the beginning of the file
    this.position = 0;

    this.readyState = 0; // EMPTY

    this.result = null;

    // Error
    this.error = null;

    // Event handlers
    this.onwritestart = null;    // When writing starts
    this.onprogress = null;    // While writing the file, and reporting
                                // partial file data
    this.onwrite = null;        // When the write has successfully completed.
    this.onwriteend = null;    // When the request has completed (either in
                                // success or failure).
    this.onabort = null;        // When the write has been aborted. For
                                // instance, by invoking the abort() method.
    this.onerror = null;        // When the write has failed (see errors).
};

// States
FileWriter.INIT = 0;
FileWriter.WRITING = 1;
FileWriter.DONE = 2;

/**
 * Abort writing file.
 */
FileWriter.prototype.abort = function() {
    // check for invalid state
    if (this.readyState === FileWriter.DONE || this.readyState ===
        FileWriter.INIT) {
        throw FileError.INVALID_STATE_ERR;
    }

    // set error
    var error = new FileError(), evt;
    error.code = error.ABORT_ERR;
    this.error = error;

    // If error callback
    if (typeof this.onerror === "function") {
        this.onerror({"type": "error", "target": this});
    }
    // If abort callback
    if (typeof this.onabort === "function") {
        this.onabort({"type": "abort", "target": this});
    }

    this.readyState = FileWriter.DONE;

    // If write end callback

```

```
        if (typeof this.onwriteend == "function") {
            this.onwriteend({"type":"writeend", "target":this});
        }
    };

    /**
     * Writes data to the file
     *
     * @param text to be written
     */
    FileWriter.prototype.write = function(text) {
        // Throw an exception if we are already writing a file
        if (this.readyState === FileWriter.WRITING) {
            throw FileError.INVALID_STATE_ERR;
        }

        // WRITING state
        this.readyState = FileWriter.WRITING;

        var me = this;

        // If onwritestart callback
        if (typeof me.onwritestart === "function") {
            me.onwritestart({"type":"writestart", "target":me});
        }

        // Write file
        navigator.fileMgr.write(this.fileName, text, this.position,

            // Success callback
            function(r) {
                var evt;
                // If DONE (cancelled), then don't do anything
                if (me.readyState === FileWriter.DONE) {
                    return;
                }

                // position always increases by bytes written because file would be
                // extended
                me.position += r;
                // The length of the file is now where we are done writing.
                me.length = me.position;

                // If onwrite callback
                if (typeof me.onwrite === "function") {
                    me.onwrite({"type":"write", "target":me});
                }

                // DONE state
                me.readyState = FileWriter.DONE;

                // If onwriteend callback
                if (typeof me.onwriteend === "function") {
                    me.onwriteend({"type":"writeend", "target":me});
                }
            }
        );
    };
}
```

```

    }
  },

  // Error callback
  function(e) {
    var evt;

    // If DONE (cancelled), then don't do anything
    if (me.readyState === FileWriter.DONE) {
      return;
    }

    // Save error
    me.error = e;

    // If onerror callback
    if (typeof me.onerror === "function") {
      me.onerror({"type":"error", "target":me});
    }

    // DONE state
    me.readyState = FileWriter.DONE;

    // If onwriteend callback
    if (typeof me.onwriteend === "function") {
      me.onwriteend({"type":"writeend", "target":me});
    }
  }
);

};

/**
 * Moves the file pointer to the location specified.
 *
 * If the offset is a negative number the position of the file
 * pointer is rewound. If the offset is greater than the file
 * size the position is set to the end of the file.
 *
 * @param offset is the location to move the file pointer to.
 */
FileWriter.prototype.seek = function(offset) {
  // Throw an exception if we are already writing a file
  if (this.readyState === FileWriter.WRITING) {
    throw FileError.INVALID_STATE_ERR;
  }

  if (!offset) {
    return;
  }

  // See back from end of file.
  if (offset < 0) {
    this.position = Math.max(offset + this.length, 0);
  }

```

```
    }
    // Offset is bigger then file size so set position
    // to the end of the file.
    else if (offset > this.length) {
        this.position = this.length;
    }
    // Offset is between 0 and file size so set the position
    // to start writing.
    else {
        this.position = offset;
    }
};

/**
 * Truncates the file to the size specified.
 *
 * @param size to chop the file at.
 */
FileWriter.prototype.truncate = function(size) {
    // Throw an exception if we are already writing a file
    if (this.readyState === FileWriter.WRITING) {
        throw FileError.INVALID_STATE_ERR;
    }

    // WRITING state
    this.readyState = FileWriter.WRITING;

    var me = this;

    // If onwritestart callback
    if (typeof me.onwritestart === "function") {
        me.onwritestart({"type": "writestart", "target": this});
    }

    // Write file
    navigator.fileMgr.truncate(this.fileName, size,

        // Success callback
        function(r) {
            var evt;
            // If DONE (cancelled), then don't do anything
            if (me.readyState === FileWriter.DONE) {
                return;
            }

            // Update the length of the file
            me.length = r;
            me.position = Math.min(me.position, r);

            // If onwrite callback
            if (typeof me.onwrite === "function") {
                me.onwrite({"type": "write", "target": me});
            }

            // DONE state
```



```

        me.readyState = FileWriter.DONE;

        // If onwriteend callback
        if (typeof me.onwriteend === "function") {
            me.onwriteend({"type":"writeend", "target":me});
        }
    },

    // Error callback
    function(e) {
        var evt;
        // If DONE (cancelled), then don't do anything
        if (me.readyState === FileWriter.DONE) {
            return;
        }

        // Save error
        me.error = e;

        // If onerror callback
        if (typeof me.onerror === "function") {
            me.onerror({"type":"error", "target":me});
        }

        // DONE state
        me.readyState = FileWriter.DONE;

        // If onwriteend callback
        if (typeof me.onwriteend === "function") {
            me.onwriteend({"type":"writeend", "target":me});
        }
    }
);
};

/**
 * Information about the state of the file or directory
 *
 * @constructor
 * {Date} modificationTime (readonly)
 */
var Metadata = function() {
    this.modificationTime=null;
};

/**
 * Supplies arguments to methods that lookup or create files and directories
 *
 * @constructor
 * @param {boolean} create file or directory if it doesn't exist
 * @param {boolean} exclusive if true the command will fail if the file or
 * directory exists
 */
var Flags = function(create, exclusive) {
    this.create = create || false;

```

```
        this.exclusive = exclusive || false;
    };

    /**
     * An interface representing a file system
     *
     * @constructor
     * {DOMString} name the unique name of the file system (readonly)
     * {DirectoryEntry} root directory of the file system (readonly)
     */
    var FileSystem = function() {
        this.name = null;
        this.root = null;
    };

    /**
     * An interface that lists the files and directories in a directory.
     * @constructor
     */
    var DirectoryReader = function(fullPath){
        this.fullPath = fullPath || null;
    };

    /**
     * Returns a list of entries from a directory.
     *
     * @param {Function} successCallback is called with a list of entries
     * @param {Function} errorCallback is called with a FileError
     */
    DirectoryReader.prototype.readEntries = function(successCallback, errorCallback) {
        PhoneGap.exec(successCallback, errorCallback, "File", "readEntries",
            [this.fullPath]);
    };

    /**
     * An interface representing a directory on the file system.
     *
     * @constructor
     * {boolean} isFile always false (readonly)
     * {boolean} isDirectory always true (readonly)
     * {DOMString} name of the directory, excluding the path leading to it (readonly)
     * {DOMString} fullPath the absolute full path to the directory (readonly)
     * {FileSystem} filesystem on which the directory resides (readonly)
     */
    var DirectoryEntry = function() {
        this.isFile = false;
        this.isDirectory = true;
        this.name = null;
        this.fullPath = null;
        this.filesystem = null;
    };

    /**
     * Copies a directory to a new location
     *
     */
```

```

    * @param {DirectoryEntry} parent the directory to which to copy the entry
    * @param {DOMString} newName the new name of the entry, defaults to the
    * current name
    * @param {Function} successCallback is called with the new entry
    * @param {Function} errorCallback is called with a FileError
    */
DirectoryEntry.prototype.copyTo = function(parent, newName, successCallback,
    errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "copyTo",
        [this.fullPath, parent, newName]);
};

/**
 * Looks up the metadata of the entry
 *
 * @param {Function} successCallback is called with a Metadata object
 * @param {Function} errorCallback is called with a FileError
 */
DirectoryEntry.prototype.getMetadata = function(successCallback, errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "getMetadata",
        [this.fullPath]);
};

/**
 * Gets the parent of the entry
 *
 * @param {Function} successCallback is called with a parent entry
 * @param {Function} errorCallback is called with a FileError
 */
DirectoryEntry.prototype.getParent = function(successCallback, errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "getParent",
        [this.fullPath]);
};

/**
 * Moves a directory to a new location
 *
 * @param {DirectoryEntry} parent the directory to which to move the entry
 * @param {DOMString} newName the new name of the entry, defaults to the
 * current name
 * @param {Function} successCallback is called with the new entry
 * @param {Function} errorCallback is called with a FileError
 */
DirectoryEntry.prototype.moveTo = function(parent, newName, successCallback,
    errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "moveTo",
        [this.fullPath, parent, newName]);
};

/**
 * Removes the entry
 *
 * @param {Function} successCallback is called with no parameters
 * @param {Function} errorCallback is called with a FileError
 */

```

```
DirectoryEntry.prototype.remove = function(successCallback, errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "remove",
        [this.fullPath]);
};

/**
 * Returns a URI that can be used to identify this entry.
 *
 * @param {DOMString} mimeType for a FileEntry, the mime type to be used to
 * interpret the file, when loaded through this URI.
 * @return uri
 */
DirectoryEntry.prototype.toURI = function(mimeType) {
    return "file://" + this.fullPath;
};

/**
 * Creates a new DirectoryReader to read entries from this directory
 */
DirectoryEntry.prototype.createReader = function(successCallback, errorCallback) {
    return new DirectoryReader(this.fullPath);
};

/**
 * Creates or looks up a directory
 *
 * @param {DOMString} path either a relative or absolute path from this
 * directory in which to look up or create a directory
 * @param {Flags} options to create or exclusively create the directory
 * @param {Function} successCallback is called with the new entry
 * @param {Function} errorCallback is called with a FileError
 */
DirectoryEntry.prototype.getDirectory = function(path, options,
    successCallback, errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "getDirectory",
        [this.fullPath, path, options]);
};

/**
 * Creates or looks up a file
 *
 * @param {DOMString} path either a relative or absolute path from this
 * directory in which to look up or create a file
 * @param {Flags} options to create or exclusively create the file
 * @param {Function} successCallback is called with the new entry
 * @param {Function} errorCallback is called with a FileError
 */
DirectoryEntry.prototype.getFile = function(path, options, successCallback,
    errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "getFile",
        [this.fullPath, path, options]);
};

/**
 * Deletes a directory and all of it's contents
```

```

*
* @param {Function} successCallback is called with no parameters
* @param {Function} errorCallback is called with a FileError
*/
DirectoryEntry.prototype.removeRecursively = function(successCallback,
    errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "removeRecursively",
        [this.fullPath]);
};

/**
* An interface representing a directory on the file system.
*
* @constructor
* {boolean} isFile always true (readonly)
* {boolean} isDirectory always false (readonly)
* {DOMString} name of the file, excluding the path leading to it (readonly)
* {DOMString} fullPath the absolute full path to the file (readonly)
* {FileSystem} filesystem on which the directory resides (readonly)
*/
var FileEntry = function() {
    this.isFile = true;
    this.isDirectory = false;
    this.name = null;
    this.fullPath = null;
    this.filesystem = null;
};

/**
* Copies a file to a new location
*
* @param {DirectoryEntry} parent the directory to which to copy the entry
* @param {DOMString} newName the new name of the entry, defaults to the
* current name
* @param {Function} successCallback is called with the new entry
* @param {Function} errorCallback is called with a FileError
*/
FileEntry.prototype.copyTo = function(parent, newName, successCallback,
    errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "copyTo",
        [this.fullPath, parent, newName]);
};

/**
* Looks up the metadata of the entry
*
* @param {Function} successCallback is called with a Metadata object
* @param {Function} errorCallback is called with a FileError
*/
FileEntry.prototype.getMetadata = function(successCallback, errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "getMetadata",
        [this.fullPath]);
};

/**

```

```
* Gets the parent of the entry
*
* @param {Function} successCallback is called with a parent entry
* @param {Function} errorCallback is called with a FileError
*/
FileEntry.prototype.getParent = function(successCallback, errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "getParent",
        [this.fullPath]);
};

/**
* Moves a directory to a new location
*
* @param {DirectoryEntry} parent the directory to which to move the entry
* @param {DOMString} newName the new name of the entry, defaults to the
* current name
* @param {Function} successCallback is called with the new entry
* @param {Function} errorCallback is called with a FileError
*/
FileEntry.prototype.moveTo = function(parent, newName, successCallback,
    errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "moveTo",
        [this.fullPath, parent, newName]);
};

/**
* Removes the entry
*
* @param {Function} successCallback is called with no parameters
* @param {Function} errorCallback is called with a FileError
*/
FileEntry.prototype.remove = function(successCallback, errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "remove",
        [this.fullPath]);
};

/**
* Returns a URI that can be used to identify this entry.
*
* @param {DOMString} mimeType for a FileEntry, the mime type to be used to
* interpret the file, when loaded through this URI.
* @return uri
*/
FileEntry.prototype.toURI = function(mimeType) {
    return "file://" + this.fullPath;
};

/**
* Creates a new FileWriter associated with the file that this FileEntry
* represents.
*
* @param {Function} successCallback is called with the new FileWriter
* @param {Function} errorCallback is called with a FileError
*/
```

```

FileEntry.prototype.createWriter = function(successCallback, errorCallback) {
    this.file(function(filePointer) {
        var writer = new FileWriter(filePointer);

        if (writer.fileName === null || writer.fileName === "") {
            if (typeof errorCallback == "function") {
                errorCallback({
                    "code": FileError.INVALID_STATE_ERR
                });
            }
        }

        if (typeof successCallback == "function") {
            successCallback(writer);
        }
    }, errorCallback);
};

/**
 * Returns a File that represents the current state of the file that this
 * FileEntry represents.
 *
 * @param {Function} successCallback is called with the new File object
 * @param {Function} errorCallback is called with a FileError
 */
FileEntry.prototype.file = function(successCallback, errorCallback) {
    PhoneGap.exec(successCallback, errorCallback, "File", "getFileMetadata",
        [this.fullPath]);
};

/** @constructor */
var LocalFileSystem = function() {
};

// File error codes
LocalFileSystem.TEMPORARY = 0;
LocalFileSystem.PERSISTENT = 1;
LocalFileSystem.RESOURCE = 2;
LocalFileSystem.APPLICATION = 3;

/**
 * Requests a filesystem in which to store application data.
 *
 * @param {int} type of file system being requested
 * @param {Function} successCallback is called with the new FileSystem
 * @param {Function} errorCallback is called with a FileError
 */
LocalFileSystem.prototype.requestFileSystem = function(type, size,
    successCallback, errorCallback) {
    if (type < 0 || type > 3) {
        if (typeof errorCallback == "function") {
            errorCallback({
                "code": FileError.SYNTAX_ERR
            });
        }
    }
};

```

```
    }
  }
  else {
    PhoneGap.exec(successCallback, errorCallback, "File",
      "requestFileSystem", [type, size]);
  }
};

/**
 *
 * @param {DOMString} uri referring to a local file in a filesystem
 * @param {Function} successCallback is called with the new entry
 * @param {Function} errorCallback is called with a FileError
 */
LocalFileSystem.prototype.resolveLocalFileSystemURI = function(uri,
  successCallback, errorCallback) {
  PhoneGap.exec(successCallback, errorCallback, "File",
    "resolveLocalFileSystemURI", [uri]);
};

/**
 * This function returns an array of contacts. It is required as we need to
 * convert raw * JSON objects into concrete Contact objects. Currently this
 * method is called after navigator.service.contacts.find
 * but before the find methods success call back.
 *
 * @param a JSON Objects that need to be converted to DirectoryEntry or
 * FileEntry objects.
 * @returns an entry
 */
LocalFileSystem.prototype._castFS = function(pluginResult) {
  var entry = null;
  entry = new DirectoryEntry();
  entry.isDirectory = pluginResult.message.root.isDirectory;
  entry.isFile = pluginResult.message.root.isFile;
  entry.name = pluginResult.message.root.name;
  entry.fullPath = pluginResult.message.root.fullPath;
  pluginResult.message.root = entry;
  return pluginResult;
};

LocalFileSystem.prototype._castEntry = function(pluginResult) {
  var entry = null;
  if (pluginResult.message.isDirectory) {
    console.log("This is a dir");
    entry = new DirectoryEntry();
  }
  else if (pluginResult.message.isFile) {
    console.log("This is a file");
    entry = new FileEntry();
  }
  entry.isDirectory = pluginResult.message.isDirectory;
  entry.isFile = pluginResult.message.isFile;
  entry.name = pluginResult.message.name;
```



```

        entry.fullPath = pluginResult.message.fullPath;
        pluginResult.message = entry;
        return pluginResult;
    };

    LocalFileSystem.prototype._castEntries = function(pluginResult) {
        var entries = pluginResult.message;
        var retVal = [];
        for (var i=0; i<entries.length; i++) {
            retVal.push(window.localFileSystem._createEntry(entries[i]));
        }
        pluginResult.message = retVal;
        return pluginResult;
    };

    LocalFileSystem.prototype._createEntry = function(castMe) {
        var entry = null;
        if (castMe.isDirectory) {
            console.log("This is a dir");
            entry = new DirectoryEntry();
        }
        else if (castMe.isFile) {
            console.log("This is a file");
            entry = new FileEntry();
        }
        entry.isDirectory = castMe.isDirectory;
        entry.isFile = castMe.isFile;
        entry.name = castMe.name;
        entry.fullPath = castMe.fullPath;
        return entry;
    };

    LocalFileSystem.prototype._castDate = function(pluginResult) {
        if (pluginResult.message.modificationTime) {
            var modTime = new Date(pluginResult.message.modificationTime);
            pluginResult.message.modificationTime = modTime;
        }
        else if (pluginResult.message.lastModifiedDate) {
            var file = new File();
            file.size = pluginResult.message.size;
            file.type = pluginResult.message.type;
            file.name = pluginResult.message.name;
            file.fullPath = pluginResult.message.fullPath;
            file.lastModifiedDate = new Date(pluginResult.message.lastModifiedDate);
            pluginResult.message = file;
        }
        return pluginResult;
    };

    /**
     * Add the FileSystem interface into the browser.
     */
    PhoneGap.addConstructor(function() {
        var pgLocalFileSystem = new LocalFileSystem();

```

```
// Needed for cast methods
if(typeof window.localFileSystem == "undefined") window.localFileSystem =
  pgLocalFileSystem;
if(typeof window.requestFileSystem == "undefined") window.requestFileSystem
  = pgLocalFileSystem.requestFileSystem;
if(typeof window.resolveLocalFileSystemURI == "undefined")
  window.resolveLocalFileSystemURI =
    pgLocalFileSystem.resolveLocalFileSystemURI;
});
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("filetransfer")) {
PhoneGap.addResource("filetransfer");

/**
 * FileTransfer uploads a file to a remote server.
 * @constructor
 */
var FileTransfer = function() {};

/**
 * FileUploadResult
 * @constructor
 */
var FileUploadResult = function() {
  this.bytesSent = 0;
  this.responseCode = null;
  this.response = null;
};

/**
 * FileTransferError
 * @constructor
 */
var FileTransferError = function() {
  this.code = null;
};

FileTransferError.FILE_NOT_FOUND_ERR = 1;
FileTransferError.INVALID_URL_ERR = 2;
FileTransferError.CONNECTION_ERR = 3;

/**
 * Given an absolute file path, uploads a file on the device to a remote server
```

```

* using a multipart HTTP request.
* @param filePath {String}      Full path of the file on the device
* @param server {String}        URL of the server to receive the file
* @param successCallback (Function) Callback to be invoked when upload has
*                                completed
* @param errorCallback {Function} Callback to be invoked upon error
* @param options {FileUploadOptions} Optional parameters such as file name and
*                                mimeType
*/
FileTransfer.prototype.upload = function(filePath, server, successCallback,
    errorCallback, options, debug) {

    // check for options
    var fileKey = null;
    var fileName = null;
    var mimeType = null;
    var params = null;
    if (options) {
        fileKey = options.fileKey;
        fileName = options.fileName;
        mimeType = options.mimeType;
        if (options.params) {
            params = options.params;
        }
        else {
            params = {};
        }
    }

    PhoneGap.exec(successCallback, errorCallback, 'FileTransfer', 'upload',
        [filePath, server, fileKey, fileName, mimeType, params, debug]);
};

/**
 * Options to customize the HTTP request used to upload files.
 * @constructor
 * @param fileKey {String}    Name of file request parameter.
 * @param fileName {String}   Filename to be used by the server. Defaults to
 *                             image.jpg.
 * @param mimeType {String}   Mimetype of the uploaded file. Defaults to
 *                             image/jpeg.
 * @param params {Object}     Object with key: value params to send to the
 *                             server.
 */
var FileUploadOptions = function(fileKey, fileName, mimeType, params) {
    this.fileKey = fileKey || null;
    this.fileName = fileName || null;
    this.mimeType = mimeType || null;
    this.params = params || null;
};

/*

```

```
* PhoneGap is available under *either* the terms of the modified BSD
* license *or* the MIT License (2008). See
* http://opensource.org/licenses/alphabetical for full text.
*
* Copyright (c) 2005-2010, Nitobi Software Inc.
* Copyright (c) 2010-2011, IBM Corporation
*/

if (!PhoneGap.hasResource("geolocation")) {
PhoneGap.addResource("geolocation");

/**
 * This class provides access to device GPS data.
 * @constructor
 */
var Geolocation = function() {

    // The last known GPS position.
    this.lastPosition = null;

    // Geolocation listeners
    this.listeners = {};
};

/**
 * Position error object
 *
 * @constructor
 * @param code
 * @param message
 */
var PositionError = function(code, message) {
    this.code = code;
    this.message = message;
};

PositionError.PERMISSION_DENIED = 1;
PositionError.POSITION_UNAVAILABLE = 2;
PositionError.TIMEOUT = 3;

/**
 * Asynchronously acquires the current position.
 *
 * @param {Function} successCallback    The function to call when the position
 *                                     data is available
 * @param {Function} errorCallback    The function to call when there is an
 *                                     error getting the heading position.
 *                                     (OPTIONAL)
 * @param {PositionOptions} options    The options for getting the position
 *                                     data. (OPTIONAL)
 */
Geolocation.prototype.getCurrentPosition = function(successCallback,
    errorCallback, options) {
    if (navigator._geo.listeners.global) {
```

```

        console.log("Geolocation Error: Still waiting for previous
            getCurrentPosition() request.");
    }
    try {
        errorCallback(new PositionError(PositionError.TIMEOUT, "Geolocation
            Error: Still waiting for previous getCurrentPosition()
            request."));
    } catch (e) {
    }
    return;
}
var maximumAge = 10000;
var enableHighAccuracy = false;
var timeout = 10000;
if (typeof options !== "undefined") {
    if (typeof options.maximumAge !== "undefined") {
        maximumAge = options.maximumAge;
    }
    if (typeof options.enableHighAccuracy !== "undefined") {
        enableHighAccuracy = options.enableHighAccuracy;
    }
    if (typeof options.timeout !== "undefined") {
        timeout = options.timeout;
    }
}
navigator._geo.listeners.global = {"success" : successCallback, "fail" :
    errorCallback };
PhoneGap.exec(null, null, "Geolocation", "getCurrentLocation",
    [enableHighAccuracy, timeout, maximumAge]);
};

/**
 * Asynchronously watches the geolocation for changes to geolocation. When a
 * change occurs, the successCallback is called with the new location.
 *
 * @param {Function} successCallback The function to call each time the
 * location data is available
 * @param {Function} errorCallback The function to call when there is an
 * error getting the location data.
 * (OPTIONAL)
 * @param {PositionOptions} options The options for getting the location
 * data such as frequency. (OPTIONAL)
 * @return String The watch id that must be passed to
 * #clearWatch to stop watching.
 */
Geolocation.prototype.watchPosition = function(successCallback, errorCallback,
    options) {
    var maximumAge = 10000;
    var enableHighAccuracy = false;
    var timeout = 10000;
    if (typeof options !== "undefined") {
        if (typeof options.frequency !== "undefined") {
            maximumAge = options.frequency;
        }
        if (typeof options.maximumAge !== "undefined") {

```

```
        maximumAge = options.maximumAge;
    }
    if (typeof options.enableHighAccuracy !== "undefined") {
        enableHighAccuracy = options.enableHighAccuracy;
    }
    if (typeof options.timeout !== "undefined") {
        timeout = options.timeout;
    }
}
var id = PhoneGap.createUUID();
navigator._geo.listeners[id] = {"success" : successCallback, "fail" :
    errorCallback };
PhoneGap.exec(null, null, "Geolocation", "start", [id, enableHighAccuracy,
    timeout, maximumAge]);
return id;
};

/*
 * Native callback when watch position has a new position.
 * PRIVATE METHOD
 *
 * @param {String} id
 * @param {Number} lat
 * @param {Number} lng
 * @param {Number} alt
 * @param {Number} altacc
 * @param {Number} head
 * @param {Number} vel
 * @param {Number} stamp
 */
Geolocation.prototype.success = function(id, lat, lng, alt, altacc, head, vel,
    stamp) {
    var coords = new Coordinates(lat, lng, alt, altacc, head, vel);
    var loc = new Position(coords, stamp);
    try {
        if (lat === "undefined" || lng === "undefined") {
            navigator._geo.listeners[id].fail(new
                PositionError(PositionError.POSITION_UNAVAILABLE,
                    "Lat/Lng are undefined."));
        }
        else {
            navigator._geo.lastPosition = loc;
            navigator._geo.listeners[id].success(loc);
        }
    }
    catch (e) {
        console.log("Geolocation Error: Error calling success callback function.");
    }

    if (id === "global") {
        delete navigator._geo.listeners.global;
    }
};

/**
```

```

* Native callback when watch position has an error.
* PRIVATE METHOD
*
* @param {String} id      The ID of the watch
* @param {Number} code    The error code
* @param {String} msg     The error message
*/
Geolocation.prototype.fail = function(id, code, msg) {
    try {
        navigator._geo.listeners[id].fail(new PositionError(code, msg));
    }
    catch (e) {
        console.log("Geolocation Error: Error calling error callback function.");
    }
};

/**
* Clears the specified heading watch.
*
* @param {String} id      The ID of the watch returned from #watchPosition
*/
Geolocation.prototype.clearWatch = function(id) {
    PhoneGap.exec(null, null, "Geolocation", "stop", [id]);
    delete navigator._geo.listeners[id];
};

/**
* Force the PhoneGap geolocation to be used instead of built-in.
*/
Geolocation.usingPhoneGap = false;
Geolocation.usePhoneGap = function() {
    if (Geolocation.usingPhoneGap) {
        return;
    }
    Geolocation.usingPhoneGap = true;

    // Set built-in geolocation methods to our own implementations
    // (Cannot replace entire geolocation, but can replace individual methods)
    navigator.geolocation.setLocation = navigator._geo.setLocation;
    navigator.geolocation.getCurrentPosition = navigator._geo.getCurrentPosition;
    navigator.geolocation.watchPosition = navigator._geo.watchPosition;
    navigator.geolocation.clearWatch = navigator._geo.clearWatch;
    navigator.geolocation.start = navigator._geo.start;
    navigator.geolocation.stop = navigator._geo.stop;
};

PhoneGap.addConstructor(function() {
    navigator._geo = new Geolocation();

    // No native geolocation object for Android 1.x, so use PhoneGap geolocation
    if (typeof navigator.geolocation === 'undefined') {
        navigator.geolocation = navigator._geo;
        Geolocation.usingPhoneGap = true;
    }
});

```

```
});
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010, IBM Corporation
 */

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the * MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("media")) {
PhoneGap.addResource("media");

/**
 * This class provides access to the device media, interfaces to both sound and
 * video
 *
 * @constructor
 * @param src The file name or url to play
 * @param successCallback The callback to be called when the file is done
 * playing or recording.
 * successCallback() - OPTIONAL
 * @param errorCallback The callback to be called if there is an error.
 * errorCallback(int errorCode) - OPTIONAL
 * @param statusCallback The callback to be called when media status has
 * changed.
 * statusCallback(int statusCode) - OPTIONAL
 * @param positionCallback The callback to be called when media position
 * has changed.
 * positionCallback(long position) - OPTIONAL
 */
var Media = function(src, successCallback, errorCallback, statusCallback,
    positionCallback) {

    // successCallback optional
    if (successCallback && (typeof successCallback !== "function")) {
        console.log("Media Error: successCallback is not a function");
        return;
    }

    // errorCallback optional
    if (errorCallback && (typeof errorCallback !== "function")) {
        console.log("Media Error: errorCallback is not a function");
    }
}
```



```

        return;
    }

    // statusCallback optional
    if (statusCallback && (typeof statusCallback !== "function")) {
        console.log("Media Error: statusCallback is not a function");
        return;
    }

    // statusCallback optional
    if (positionCallback && (typeof positionCallback !== "function")) {
        console.log("Media Error: positionCallback is not a function");
        return;
    }

    this.id = PhoneGap.createUUID();
    PhoneGap.mediaObjects[this.id] = this;
    this.src = src;
    this.successCallback = successCallback;
    this.errorCallback = errorCallback;
    this.statusCallback = statusCallback;
    this.positionCallback = positionCallback;
    this._duration = -1;
    this._position = -1;
};

// Media messages
Media.MEDIA_STATE = 1;
Media.MEDIA_DURATION = 2;
Media.MEDIA_POSITION = 3;
Media.MEDIA_ERROR = 9;

// Media states
Media.MEDIA_NONE = 0;
Media.MEDIA_STARTING = 1;
Media.MEDIA_RUNNING = 2;
Media.MEDIA_PAUSED = 3;
Media.MEDIA_STOPPED = 4;
Media.MEDIA_MSG = ["None", "Starting", "Running", "Paused", "Stopped"];

// TODO: Will MediaError be used?
/**
 * This class contains information about any Media errors.
 * @constructor
 */
var MediaError = function() {
    this.code = null;
    this.message = "";
};

MediaError.MEDIA_ERR_ABORTED = 1;
MediaError.MEDIA_ERR_NETWORK = 2;
MediaError.MEDIA_ERR_DECODE = 3;
MediaError.MEDIA_ERR_NONE_SUPPORTED = 4;

/**

```

```
    * Start or resume playing audio file.
    */
Media.prototype.play = function() {
    PhoneGap.exec(null, null, "Media", "startPlayingAudio", [this.id, this.src]);
};

/**
 * Stop playing audio file.
 */
Media.prototype.stop = function() {
    return PhoneGap.exec(null, null, "Media", "stopPlayingAudio", [this.id]);
};

/**
 * Seek or jump to a new time in the track..
 */
Media.prototype.seekTo = function(milliseconds) {
    PhoneGap.exec(null, null, "Media", "seekToAudio", [this.id, milliseconds]);
};

/**
 * Pause playing audio file.
 */
Media.prototype.pause = function() {
    PhoneGap.exec(null, null, "Media", "pausePlayingAudio", [this.id]);
};

/**
 * Get duration of an audio file.
 * The duration is only set for audio that is playing, paused or stopped.
 *
 * @return      duration or -1 if not known.
 */
Media.prototype.getDuration = function() {
    return this._duration;
};

/**
 * Get position of audio.
 */
Media.prototype.getCurrentPosition = function(success, fail) {
    PhoneGap.exec(success, fail, "Media", "getCurrentPositionAudio", [this.id]);
};

/**
 * Start recording audio file.
 */
Media.prototype.startRecord = function() {
    PhoneGap.exec(null, null, "Media", "startRecordingAudio", [this.id, this.src]);
};

/**
 * Stop recording audio file.
 */
```

```

Media.prototype.stopRecord = function() {
    PhoneGap.exec(null, null, "Media", "stopRecordingAudio", [this.id]);
};

/**
 * Release the resources.
 */
Media.prototype.release = function() {
    PhoneGap.exec(null, null, "Media", "release", [this.id]);
};

/**
 * List of media objects.
 * PRIVATE
 */
PhoneGap.mediaObjects = {};

/**
 * Object that receives native callbacks.
 * PRIVATE
 * @constructor
 */
PhoneGap.Media = function() {};

/**
 * Get the media object.
 * PRIVATE
 *
 * @param id          The media object id (string)
 */
PhoneGap.Media.getMediaObject = function(id) {
    return PhoneGap.mediaObjects[id];
};

/**
 * Audio has status update.
 * PRIVATE
 *
 * @param id          The media object id (string)
 * @param status      The status code (int)
 * @param msg         The status message (string)
 */
PhoneGap.Media.onStatus = function(id, msg, value) {
    var media = PhoneGap.mediaObjects[id];
    // If state update
    if (msg === Media.MEDIA_STATE) {
        if (value === Media.MEDIA_STOPPED) {
            if (media.successCallback) {
                media.successCallback();
            }
        }
        if (media.statusCallback) {
            media.statusCallback(value);
        }
    }
};

```

```
    }
    else if (msg === Media.MEDIA_DURATION) {
        media._duration = value;
    }
    else if (msg === Media.MEDIA_ERROR) {
        if (media.errorCallback) {
            media.errorCallback(value);
        }
    }
    else if (msg == Media.MEDIA_POSITION) {
        media._position = value;
    }
};
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("network")) {
    PhoneGap.addResource("network");
}

/**
 * This class contains information about the current network Connection.
 * @constructor
 */
var Connection = function() {
    this.type = null;
    this._firstRun = true;
    this._timer = null;
    this.timeout = 500;

    var me = this;
    this.getInfo(
        function(type) {
            // Need to send events if we are on or offline
            if (type == "none") {
                // set a timer if still offline at the end of timer send the
                // offline event
                me._timer = setTimeout(function(){
                    me.type = type;
                    PhoneGap.fireEvent('offline');
                    me._timer = null;
                }, me.timeout);
            } else {
                // If there is a current offline event pending clear it
                if (me._timer != null) {
                    clearTimeout(me._timer);
                }
            }
        }
    );
};
```

```

        me._timer = null;
    }
    me.type = type;
    PhoneGap.fireEvent('online');
}

// should only fire this once
if (me._firstRun) {
    me._firstRun = false;
    PhoneGap.onPhoneGapConnectionReady.fire();
}
},
function(e) {
    console.log("Error initializing Network Connection: " + e);
});
};

Connection.UNKNOWN = "unknown";
Connection.ETHERNET = "ethernet";
Connection.WIFI = "wifi";
Connection.CELL_2G = "2g";
Connection.CELL_3G = "3g";
Connection.CELL_4G = "4g";
Connection.NONE = "none";

/**
 * Get connection info
 *
 * @param {Function} successCallback The function to call when the Connection
 *                                data is available
 * @param {Function} errorCallback The function to call when there is an
 *                                error getting the Connection data.
 *                                (OPTIONAL)
 */
Connection.prototype.getInfo = function(successCallback, errorCallback) {
    // Get info
    PhoneGap.exec(successCallback, errorCallback, "Network Status",
        "getConnectionInfo", []);
};

PhoneGap.addConstructor(function() {
    if (typeof navigator.network === "undefined") {
        navigator.network = new Object();
    }
    if (typeof navigator.network.connection === "undefined") {
        navigator.network.connection = new Connection();
    }
});
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD

```

```
* license *or* the MIT License (2008). See
* http://opensource.org/licenses/alphabetical for full text.
*
* Copyright (c) 2005-2010, Nitobi Software Inc.
* Copyright (c) 2010-2011, IBM Corporation
*/

if (!PhoneGap.hasResource("notification")) {
PhoneGap.addResource("notification");

/**
 * This class provides access to notifications on the device.
 * @constructor
 */
var Notification = function() {
};

/**
 * Open a native alert dialog, with a customizable title and button text.
 *
 * @param {String} message      Message to print in the body of the
 *                               alert
 * @param {Function} completeCallback  The callback that is called when user
 *                                     clicks on a button.
 * @param {String} title        Title of the alert dialog (default:
 *                               Alert)
 * @param {String} buttonLabel   Label of the close button (default: OK)
 */
Notification.prototype.alert = function(message, completeCallback, title,
    buttonLabel) {
    var _title = (title || "Alert");
    var _buttonLabel = (buttonLabel || "OK");
    PhoneGap.exec(completeCallback, null, "Notification", "alert",
        [message, _title, _buttonLabel]);
};

/**
 * Open a native confirm dialog, with a customizable title and button text.
 * The result that the user selects is returned to the result callback.
 *
 * @param {String} message      Message to print in the body of the
 *                               alert
 * @param {Function} resultCallback  The callback that is called when user
 *                                     clicks on a button.
 * @param {String} title        Title of the alert dialog (default:
 *                               Confirm)
 * @param {String} buttonLabels  Comma separated list of the labels of
 *                               the buttons (default: 'OK,Cancel')
 */
Notification.prototype.confirm = function(message, resultCallback, title,
    buttonLabels) {
    var _title = (title || "Confirm");
    var _buttonLabels = (buttonLabels || "OK,Cancel");
```

```

        PhoneGap.exec(resultCallback, null, "Notification", "confirm",
            [message,_title,_buttonLabels]);
    };

    /**
     * Start spinning the activity indicator on the statusbar
     */
    Notification.prototype.activityStart = function() {
        PhoneGap.exec(null, null, "Notification", "activityStart", ["Busy","Please
            wait..."]);
    };

    /**
     * Stop spinning the activity indicator on the statusbar, if it's currently
     * spinning
     */
    Notification.prototype.activityStop = function() {
        PhoneGap.exec(null, null, "Notification", "activityStop", []);
    };

    /**
     * Display a progress dialog with progress bar that goes from 0 to 100.
     *
     * @param {String} title      Title of the progress dialog.
     * @param {String} message    Message to display in the dialog.
     */
    Notification.prototype.progressStart = function(title, message) {
        PhoneGap.exec(null, null, "Notification", "progressStart", [title, message]);
    };

    /**
     * Set the progress dialog value.
     *
     * @param {Number} value      0-100
     */
    Notification.prototype.progressValue = function(value) {
        PhoneGap.exec(null, null, "Notification", "progressValue", [value]);
    };

    /**
     * Close the progress dialog.
     */
    Notification.prototype.progressStop = function() {
        PhoneGap.exec(null, null, "Notification", "progressStop", []);
    };

    /**
     * Causes the device to blink a status LED.
     *
     * @param {Integer} count      The number of blinks.
     * @param {String} colour      The colour of the light.
     */
    Notification.prototype.blink = function(count, colour) {

```

```
        // NOT IMPLEMENTED
    };

    /**
     * Causes the device to vibrate.
     *
     * @param {Integer} mills      The number of milliseconds to vibrate for.
     */
    Notification.prototype.vibrate = function(mills) {
        PhoneGap.exec(null, null, "Notification", "vibrate", [mills]);
    };

    /**
     * Causes the device to beep.
     * On Android, the default notification ringtone is played "count" times.
     *
     * @param {Integer} count      The number of beeps.
     */
    Notification.prototype.beep = function(count) {
        PhoneGap.exec(null, null, "Notification", "beep", [count]);
    };

    PhoneGap.addConstructor(function() {
        if (typeof navigator.notification === "undefined") {
            navigator.notification = new Notification();
        }
    });
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD
 * license *or* the MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

if (!PhoneGap.hasResource("position")) {
    PhoneGap.addResource("position");
}

/**
 * This class contains position information.
 * @param {Object} lat
 * @param {Object} lng
 * @param {Object} acc
 * @param {Object} alt
 * @param {Object} altacc
 * @param {Object} head
 * @param {Object} vel
 * @constructor
 */
var Position = function(coords, timestamp) {
    this.coords = coords;
    this.timestamp = (timestamp !== 'undefined') ? timestamp : new
```



```

        Date().getTime();
    };

    /** @constructor */
    var Coordinates = function(lat, lng, alt, acc, head, vel, altacc) {
        /**
         * The latitude of the position.
         */
        this.latitude = lat;
        /**
         * The longitude of the position,
         */
        this.longitude = lng;
        /**
         * The accuracy of the position.
         */
        this.accuracy = acc;
        /**
         * The altitude of the position.
         */
        this.altitude = alt;
        /**
         * The direction the device is moving at the position.
         */
        this.heading = head;
        /**
         * The velocity with which the device is moving at the position.
         */
        this.speed = vel;
        /**
         * The altitude accuracy of the position.
         */
        this.altitudeAccuracy = (altacc !== 'undefined') ? altacc : null;
    };

    /**
     * This class specifies the options for requesting position data.
     * @constructor
     */
    var PositionOptions = function() {
        /**
         * Specifies the desired position accuracy.
         */
        this.enableHighAccuracy = true;
        /**
         * The timeout after which if position data cannot be obtained the
         * errorCallback
         * is called.
         */
        this.timeout = 10000;
    };

    /**
     * This class contains information about any GSP errors.
     * @constructor
     */

```

```
var PositionError = function() {
    this.code = null;
    this.message = "";
};

PositionError.UNKNOWN_ERROR = 0;
PositionError.PERMISSION_DENIED = 1;
PositionError.POSITION_UNAVAILABLE = 2;
PositionError.TIMEOUT = 3;
}

/*
 * PhoneGap is available under *either* the terms of the modified BSD \
 * license *or* the MIT License (2008). See
 * http://opensource.org/licenses/alphabetical for full text.
 *
 * Copyright (c) 2005-2010, Nitobi Software Inc.
 * Copyright (c) 2010-2011, IBM Corporation
 */

/*
 * This is purely for the Android 1.5/1.6 HTML 5 Storage
 * I was hoping that Android 2.0 would deprecate this, but given the fact that
 * most manufacturers ship with Android 1.5 and do not do OTA Updates, this is
 * required
 */

if (!PhoneGap.hasResource("storage")) {
    PhoneGap.addResource("storage");
}

/**
 * SQL result set object
 * PRIVATE METHOD
 * @constructor
 */
var DroidDB_Rows = function() {
    this.resultSet = [];    // results array
    this.length = 0;       // number of rows
};

/**
 * Get item from SQL result set
 *
 * @param row        The row number to return
 * @return           The row object
 */
DroidDB_Rows.prototype.item = function(row) {
    return this.resultSet[row];
};

/**
 * SQL result set that is returned to user.
 * PRIVATE METHOD
```

```

    * @constructor
    */
    var DroidDB_Result = function() {
        this.rows = new DroidDB_Rows();
    };

    /**
     * Storage object that is called by native code when performing queries.
     * PRIVATE METHOD
     * @constructor
     */
    var DroidDB = function() {
        this.queryQueue = {};
    };

    /**
     * Callback from native code when query is complete.
     * PRIVATE METHOD
     *
     * @param id          Query id
     */
    DroidDB.prototype.completeQuery = function(id, data) {
        var query = this.queryQueue[id];
        if (query) {
            try {
                delete this.queryQueue[id];

                // Get transaction
                var tx = query.tx;

                // If transaction hasn't failed
                // Note: We ignore all query results if previous query
                //       in the same transaction failed.
                if (tx && tx.queryList[id]) {

                    // Save query results
                    var r = new DroidDB_Result();
                    r.rows.resultSet = data;
                    r.rows.length = data.length;
                    try {
                        if (typeof query.successCallback === 'function') {
                            query.successCallback(query.tx, r);
                        }
                    } catch (ex) {
                        console.log("executeSql error calling user success
                                callback: "+ex);
                    }

                    tx.queryComplete(id);
                }
            } catch (e) {
                console.log("executeSql error: "+e);
            }
        }
    }
}

```

```
};

/**
 * Callback from native code when query fails
 * PRIVATE METHOD
 *
 * @param reason      Error message
 * @param id          Query id
 */
DroidDB.prototype.fail = function(reason, id) {
    var query = this.queryQueue[id];
    if (query) {
        try {
            delete this.queryQueue[id];

            // Get transaction
            var tx = query.tx;

            // If transaction hasn't failed
            // Note: We ignore all query results if previous query
            //       in the same transaction failed.
            if (tx && tx.queryList[id]) {
                tx.queryList = {};

                try {
                    if (typeof query.errorCallback === 'function') {
                        query.errorCallback(query.tx, reason);
                    }
                } catch (ex) {
                    console.log("executeSql error calling user error callback: "
                                +ex);
                }

                tx.queryFailed(id, reason);
            }

        } catch (e) {
            console.log("executeSql error: " +e);
        }
    }
};

/**
 * SQL query object
 * PRIVATE METHOD
 *
 * @constructor
 * @param tx          The transaction object that this query belongs to
 */
var DroidDB_Query = function(tx) {

    // Set the id of the query
    this.id = PhoneGap.createUUID();

    // Add this query to the queue
```

```

        droiddb.queryQueue[this.id] = this;

        // Init result
        this.resultSet = [];

        // Set transaction that this query belongs to
        this.tx = tx;

        // Add this query to transaction list
        this.tx.queryList[this.id] = this;

        // Callbacks
        this.successCallback = null;
        this.errorCallback = null;
    };

    /**
     * Transaction object
     * PRIVATE METHOD
     * @constructor
     */
    var DroidDB_Tx = function() {

        // Set the id of the transaction
        this.id = PhoneGap.createUUID();

        // Callbacks
        this.successCallback = null;
        this.errorCallback = null;

        // Query list
        this.queryList = {};
    };

    /**
     * Mark query in transaction as complete.
     * If all queries are complete, call the user's transaction success callback.
     *
     * @param id          Query id
     */
    DroidDB_Tx.prototype.queryComplete = function(id) {
        delete this.queryList[id];

        // If no more outstanding queries, then fire transaction success
        if (this.successCallback) {
            var count = 0;
            var i;
            for (i in this.queryList) {
                if (this.queryList.hasOwnProperty(i)) {
                    count++;
                }
            }
            if (count === 0) {

```

```
        try {
            this.successCallback();
        } catch(e) {
            console.log("Transaction error calling user success callback: "
                + e);
        }
    }
}
};

/**
 * Mark query in transaction as failed.
 *
 * @param id          Query id
 * @param reason      Error message
 */
DroidDB_Tx.prototype.queryFailed = function(id, reason) {

    // The sql queries in this transaction have already been run, since
    // we really don't have a real transaction implemented in native code.
    // However, the user callbacks for the remaining sql queries in transaction
    // will not be called.
    this.queryList = {};

    if (this.errorCallback) {
        try {
            this.errorCallback(reason);
        } catch(e) {
            console.log("Transaction error calling user error callback: " + e);
        }
    }
};

/**
 * Execute SQL statement
 *
 * @param sql          SQL statement to execute
 * @param params       Statement parameters
 * @param successCallback Success callback
 * @param errorCallback Error callback
 */
DroidDB_Tx.prototype.executeSql = function(sql, params, successCallback,
    errorCallback) {

    // Init params array
    if (typeof params === 'undefined') {
        params = [];
    }

    // Create query and add to queue
    var query = new DroidDB_Query(this);
    droiddb.queryQueue[query.id] = query;

    // Save callbacks
```

```

        query.successCallback = successCallback;
        query.errorCallback = errorCallback;

        // Call native code
        PhoneGap.exec(null, null, "Storage", "executeSql", [sql, params, query.id]);
    };

    var DatabaseShell = function() {
    };

    /**
     * Start a transaction.
     * Does not support rollback in event of failure.
     *
     * @param process {Function}          The transaction function
     * @param successCallback {Function}
     * @param errorCallback {Function}
     */
    DatabaseShell.prototype.transaction = function(process, errorCallback,
        successCallback) {
        var tx = new DroidDB_Tx();
        tx.successCallback = successCallback;
        tx.errorCallback = errorCallback;
        try {
            process(tx);
        } catch (e) {
            console.log("Transaction error: "+e);
            if (tx.errorCallback) {
                try {
                    tx.errorCallback(e);
                } catch (ex) {
                    console.log("Transaction error calling user error callback: "+e);
                }
            }
        }
    };

    /**
     * Open database
     *
     * @param name          Database name
     * @param version        Database version
     * @param display_name   Database display name
     * @param size           Database size in bytes
     * @return              Database object
     */
    var DroidDB_openDatabase = function(name, version, display_name, size) {
        PhoneGap.exec(null, null, "Storage", "openDatabase", [name, version,
            display_name, size]);
        var db = new DatabaseShell();
        return db;
    };

    /**

```

```
* For browsers with no localStorage we emulate it with SQLite. Follows the w3c
* api.
* TODO: Do similar for sessionStorage.
*/

/**
 * @constructor
 */
var CupcakeLocalStorage = function() {
    try {

        this.db = openDatabase('localStorage', '1.0', 'localStorage',
            2621440);
        var storage = {};
        this.length = 0;
        function setLength (length) {
            this.length = length;
            localStorage.length = length;
        }
        this.db.transaction(
            function (transaction) {
                var i;
                transaction.executeSql('CREATE TABLE IF NOT EXISTS
                    storage (id NVARCHAR(40) PRIMARY KEY, body
                    NVARCHAR(255))');
                transaction.executeSql('SELECT * FROM storage', [],
                    function(tx, result) {
                        for(var i = 0; i < result.rows.length; i++) {
                            storage[result.rows.item(i)['id']] =
                                result.rows.item(i)['body'];
                        }
                        setLength(result.rows.length);
                        PhoneGap.initializationComplete
                            ("cupcakeStorage");
                    }
                ));

            },
            function (err) {
                alert(err.message);
            }
        );
        this.setItem = function(key, val) {
            if (typeof(storage[key])=='undefined') {
                this.length++;
            }
            storage[key] = val;
            this.db.transaction(
                function (transaction) {
                    transaction.executeSql('CREATE TABLE IF NOT
                        EXISTS storage (id NVARCHAR(40) PRIMARY KEY,
                        body NVARCHAR(255))');
                    transaction.executeSql('REPLACE INTO storage (id,
                        body) values(?,?)', [key,val]);
                }
            );
        };
    }
};
```



```

    );
};
this.getItem = function(key) {
    return storage[key];
};
this.removeItem = function(key) {
    delete storage[key];
    this.length--;
    this.db.transaction(
        function (transaction) {
            transaction.executeSql('CREATE TABLE IF NOT
                EXISTS storage (id NVARCHAR(40) PRIMARY KEY,
                body NVARCHAR(255))');
            transaction.executeSql('DELETE FROM storage where
                id=?', [key]);
        }
    );
};
this.clear = function() {
    storage = {};
    this.length = 0;
    this.db.transaction(
        function (transaction) {
            transaction.executeSql('CREATE TABLE IF NOT
                EXISTS storage (id NVARCHAR(40) PRIMARY KEY,
                body NVARCHAR(255))');
            transaction.executeSql('DELETE FROM
                storage', []);
        }
    );
};
this.key = function(index) {
    var i = 0;
    for (var j in storage) {
        if (i==index) {
            return j;
        } else {
            i++;
        }
    }
    return null;
};

} catch(e) {
    alert("Database error "+e+".");
    return;
}

};

PhoneGap.addConstructor(function() {
    var setupDroidDB = function() {
        navigator.openDatabase = window.openDatabase = DroidDB_openDatabase;
        window.droiddb = new DroidDB();
    }

```

```
if (typeof window.openDatabase === "undefined") {
    setupDroidDB();
} else {
    window.openDatabase_orig = window.openDatabase;
    window.openDatabase = function(name, version, desc, size){
        // Some versions of Android will throw a SECURITY_ERR so we need
        // to catch the exception and seutp our own DB handling.
        var db = null;
        try {
            db = window.openDatabase_orig(name, version, desc, size);
        }
        catch (ex) {
            db = null;
        }

        if (db == null) {
            setupDroidDB();
            return DroidDB_openDatabase(name, version, desc, size);
        }
        else {
            return db;
        }
    }
}

if (typeof window.localStorage === "undefined") {
    navigator.localStorage = window.localStorage = new CupcakeLocalStorage();
    PhoneGap.waitForInitialization("cupcakeStorage");
}

});
}
```

D

PhoneGap Plug-ins

An entire community of developers has started creating plug-ins for PhoneGap on Android and iOS device platforms. (There are currently no plug-ins yet for Palm and BlackBerry.) The idea behind a plug-in is simple: Create native and JavaScript code solutions that other developers can “plug in” to their projects and, thus, quickly and easily extend PhoneGap’s capabilities.

Currently, all PhoneGap plug-ins are hosted at <https://github.com/phonegap/phonegap-plugins>, which is shown in Figure D-1.

The screenshot shows the GitHub repository page for `phonegap/phonegap-plugins`. The repository is forked from `purplecabbage/phonegap-plugins`. The page includes tabs for Source, Commits, Network, Pull Requests (1), Issues (8), and Graphs. A pull request #100 from `ecamacho/master` is highlighted, showing the commit details and the changes made. Below the pull request, there is a table of files in the repository:

name	age	message	history
<code>Android/</code>	3 days ago	Updates to the Android status bar plugin [macdonst]	
<code>BlackBerry/</code>	October 06, 2010	added dirs for other devices [purplecabbage]	
<code>Palm/</code>	October 06, 2010	added dirs for other devices [purplecabbage]	
<code>iPhone/</code>	about 18 hours ago	Merge pull request #100 from ecamacho/master [shazron]	
<code>.gitignore</code>	August 27, 2010	Added exclusions to .gitignore [shazron]	
<code>README</code>	February 03, 2010	Added MIT License [purplecabbage]	

Below the table, there is a `README` section with the following text:

```
This code is completely dependent on the PhoneGap project, also hosted on
Github ( github.com/phonegap/phonegap )
```

FIGURE D-1: Site where PhoneGap plug-ins are hosted

This appendix summarizes each of the available plug-ins for each platform.

ANDROID PLUG-INS

The following plug-ins are available for Android:

- Analytics
- BarcodeScanner
- BlueTooth
- ChildBrowser
- ClipboardManager
- ContactView
- FileUploader
- FtpClient
- PayPalPlugin
- PowerManagement
- Share
- SoftKeyboard
- StatusBarNotification
- TTS
- WebIntent

Analytics

Use this plug-in to integrate Google analytics with your Android app. It creates a `window.plugins.analytics` object.

To start, enter the following code:

```
/**
 * Initialize Google Analytics configuration
 *
 * @param accountId      The Google Analytics account id
 * @param successCallback The success callback
 * @param failureCallback The error callback
 */

start(accountId, successCallback, failureCallback);
```

To track a page view, use the following code:

```
/**
 * Track a page view on Google Analytics
 * @param key The name of the tracked item (can be a url or some logical name).
 * The key name will be presented in Google Analytics report.
 * @param successCallback The success callback
```

```

* @param failureCallback    The error callback
*/

trackPageView(key, successCallback, failureCallback);

```

BarcodeScanner

This plug-in creates the `window.plugins.barcodeScanner` object with one method, `scan` (`types`, `success`, `fail` options), where `types` is a comma-separated list of barcode types that the scanner should accept. If you pass `null`, any barcode type will be accepted. The following types are currently available:

- QR_CODE
- DATA_MATRIX
- UPC_E
- UPC_A
- EAN_8
- EAN_13
- CODE_128
- CODE_39
- CODE_93
- CODABAR
- ITF
- RSS14
- PDF417
- RSS_EXPANDED

Following is an example:

```

window.plugins.barcodeScanner.scan( BarcodeScanner.Type.QR_CODE, function(result) {
    alert("We got a barcode: " + result);
}, function(error) {
    alert("Scanning failed: " + error);
}, {yesString: "Install"}
);

```

Bluetooth

Use the `Bluetooth` plug-in to list available Bluetooth devices and connect/disconnect them.

Following is an example:

```

<plugin name="BluetoothPlugin"
    value="com.phonegap.plugin.bluetooth.BluetoothPlugin"/>

```

ChildBrowser

The `ChildBrowser` plug-in enables you to display external web pages within your PhoneGap application. For example, you might want to display links to web pages inside your app, but keep users from leaving your app to access that information.

Following is an example:

```
window.plugins.childBrowser.showWebPage("http://www.google.com");
```

ClipboardManager

The `ClipboardManager` lets you copy and paste. It implements a `windows.plugins.clipboardManager` object with `copy()` and `paste()` methods.

To copy, use the following code:

```
window.plugins.clipboardManager.copy(
    "the text to copy",
    function(r){alert("copy is successful")},
    function(e){alert(e)}
);
```

To paste, use the following code:

```
window.plugins.clipboardManager.paste(
    function(r){alert("The text in the clipboard is " + r)},
    function(e){alert(e)}
);
```

ContactView

This plug-in provides an alternative approach to working with contacts on the Android platform. Instead of pulling the contacts into an HTML web view, users are taken to the native `Contacts` app and then returned to the HTML web view with the information they want.

Following is an example:

```
document.querySelector("#contact-name-to-native")
    .addEventListener("touchstart", function() {
        window.plugins.contactView.show(
            function(contact) {
                document.getElementById("contact-name-from-native").value =
                    contact.name;
                document.getElementById("contact-phone").value =
                    contact.phone;
            },
            function(fail) {
                alert("We were unable to get the contact you selected.");
            }
        );
    }, false);
```

FileUploader

This plug-in creates the `window.plugins.fileUploader` object with two methods — `upload` and `uploadByUri`. The `upload` method takes an absolute path (for example, `/sdcard/media/images/image.jpg`), and `uploadByUri` takes a `content://` URI (for example, `content://media/external/images/media/5`).

Following are the full parameters:

- **Server:** URL of the server that will receive the file.
- **File:** Path or URI of the file to upload.
- **fileKey:** Object with key/value parameters to send to the server.
- **params:** Parameter name of the file.
- **filename:** Filename to send to the server (defaults to `image.jpg`).
- **mimeType:** MIME type of the uploaded file (defaults to `image/jpeg`).
- **callback:** Success callback. Also receives progress messages during upload.
- **fail:** Error callback.

Note that the success callback isn't just called when the upload is complete, but also gets progress events while the file is uploaded. It's passed an object of the form, as shown here:

```
{
  status: 'PROGRESS', //Either FileUploader.Status.PROGRESS or
                    FileUploader.Status.COMPLETE,
  progress: 0,        //The number of bytes uploaded so far.
  total: 1000,         //The total number of bytes to upload.
  result: "OK"         //The string returned from the server. Only sent when
                    status = complete.
}
```

FtpClient

The `FtpClient` plug-in enables you to upload and download files from within your PhoneGap application.

To upload a file, use the following:

```
window.plugins.ftpclient.put("test.txt",
  "ftp://username:password@ftp.server.com/test.txt?type=i", win, fail);
```

To download a file, use the following:

```
window.plugins.ftpclient.get("test.txt",
  "ftp://username:password@ftp.server.com/test.txt?type=i", win, fail);
```

PayPalPlugin

The `PayPalPlugin` lets you integrate PayPal payments directly into your app. For this to work, you not only must install the plug-in, but also configure your PayPal account for a testing/sandbox environment.

PowerManagement

The `PowerManagement` plug-in offers access to the device's power-management interface. You can use it to keep apps running without user intervention.

Share

Use the `Share` plug-in to share information from your app with others via e-mail.

Following is an example:

```
window.plugins.share.show({
  subject: 'I like dogs',
  text: 'http://www.example.com'},
  function() {},
  function() {alert('Share failed')}}
);
```

SoftKeyboard

Use the `SoftKeyBoard` plug-in to show or hide the soft keyboard.

Following is an example:

```
plugins.SoftKeyBoard.show(function () {
  // success
},function () {
  // fail
});
```

StatusBarNotification

The `StatusBarNotification` plug-in lets you display notifications in the status bar of your app.

Following is an example:

```
window.plugins.statusBarNotification.notify("Put your title here", "Put your
message here");
```

TTS

The `TTS` plug-in lets you access the text-to-speech functionality on the device. It creates a window `.plugins.tts` object, and has various methods available to it.

To speak, use the following:

```
window.plugins.tts.speak("The TTS service is ready", win, fail);
```

To play silence for two seconds, use the following:

```
window.plugins.tts.silence(2000, win, fail);
```

WebIntent

The `WebIntent` plug-in creates a `window.plugins.webintent` object with three methods — `startActivity`, `hasExtra`, and `getExtra`.

`startActivity` launches an Android intent. Following is an example:

```
window.plugins.webintent.startActivity({
  action: WebIntent.ACTION_VIEW,
  url: 'geo:0,0?q=' + address},
  function() {},
  function() {alert('Failed to open URL via Android Intent')}});
```

`hasExtra` checks if this app was invoked with the specified `Extra`. Following is an example:

```
window.plugins.webintent.hasExtra(WebIntent.EXTRA_TEXT,
  function(has) {
    // has is true if it has the extra
  }, function() {
    // Something really bad happened.
  }
);
```

`getExtra` gets the `Extra` that this app was invoked with. Following is an example:

```
window.plugins.webintent.getExtra(WebIntent.EXTRA_TEXT,
  function(url) {
    // url is the value of EXTRA_TEXT
  }, function() {
    // There was no extra supplied.
  }
);
```

IOS PLUGINS

The following plug-ins are available for iOS devices (iPhone, iPod Touch, and iPad):

- `AdPlugin`
- `ApplicationPreferences`
- `Badge`

- `BarcodeScanner`
- `BundleFileReader`
- `ChildBrowser`
- `ClipboardPlugin`
- `DatePicker`
- `EmailComposer`
- `FileUploader`
- `GapSocket`
- `Globalization`
- `InAppPurchaseManager`
- `Keychain`
- `LocalNotification`
- `Localizable`
- `MapKitPlug`
- `NativeControls`
- `networkActivityIndicator`
- `PayPalPlugin`
- `PowerManagement`
- `Prompt`
- `SMSComposer`
- `ScreenShot`
- `ShareKitPlugin`
- `SplashScreen`
- `VolumeSlider`

AdPlugin

Using this plug-in requires the use of the iAd network, but it enables you to run iAds in your PhoneGap applications.

ApplicationPreferences

The `ApplicationPreferences` plug-in creates a `window.plugins.applicationPreferences` object with two methods — `get` and `set`.

To get a preference, use the following:

```
window.plugins.applicationPreferences.get('name_identifier', function(result) {
    alert("We got a setting: " + result);
}, function(error) {
    alert("Failed to retrieve a setting: " + error);
});
```

To set a preference, use the following:

```
window.plugins.applicationPreferences.set('name_identifier', 'homer' function() {
    alert("It is saved");
}, function(error) {
    alert("Failed to retrieve a setting: " + error);
});
```

Badge

The Badge plug-in lets you add a number (with red background) to an iPhone app icon. It has a `set` and a `clear` method.

BarcodeScanner

The plug-in creates the `window.plugins.barcodeScanner` object with one method, `scan` (`types`, `success`, `fail` options), where `types` is a comma-separated list of barcode types that the scanner should accept. If you pass `null`, any barcode type will be accepted.

Currently, only `QR_CODE` is supported.

Following is an example:

```
window.plugins.barcodeScanner.scan( BarcodeScanner.Type.QR_CODE, function(result) {
    alert("We got a barcode: " + result);
}, function(error) {
    alert("Scanning failed: " + error);
}, {yesString: "Install"}
);
```

BundleFileReader

Use the `BundleFileReader` plug-in to read files from your app bundle.

Following is an example:

```
plugins.bundleFileReader.readResource(
    'someFile', 'txt', 'www/',
    function(contents) {
        console.log('Contents of www/someFile.txt: ' + contents);
    },
    );
```

```
function() {  
    console.log('File is missing');  
}  
);
```

ChildBrowser

The ChildBrowser plug-in enables you to display external web pages within your PhoneGap application. For example, you might want to display links to web pages inside your app, but keep users from leaving your app to access that information.

Following is an example:

```
window.plugins.childBrowser.showWebPage("http://www.google.com");
```

ClipboardPlugin

Use the ClipboardPlugin to implement copy/paste functionality within your PhoneGap apps.

DatePicker

Add calendar-style date pickers to form elements in your PhoneGap applications. Here's a full example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
    "http://www.w3.org/TR/html4/strict.dtd">  
<html>  
  <head>  
    <meta name="viewport" content="width=default-width; user-scalable=no" />  
    <meta http-equiv="Content-type" content="text/html; charset=utf-8">  
    <title>example</title>  
    <link rel="stylesheet" media="only screen and (max-device-width: 1024px)"  
        href="ipad.css" type="text/css" />  
    <link rel="stylesheet" media="only screen and (max-device-width: 480px)"  
        href="iphone.css" type="text/css" />  
    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>  
    <script type="text/javascript" charset="utf-8"  
        src="DatePicker.js"></script>  
    <script type="text/javascript" charset="utf-8">  
function preventBehavior(e)  
{  
    e.preventDefault();  
}  
document.addEventListener("touchmove", preventBehavior, false);  
  
function onBodyLoad()  
{  
    document.addEventListener("deviceready", onDeviceReady, false);  
}  
  
function onDeviceReady()
```

```

    {
    }

    var cb = function(date) {
        console.log(date.toString());
        document.getElementById("date").innerHTML = date.toString();
    }

    var show = function(mode) {
        plugins.datePicker.show({
            date: new Date(),
            mode: mode, //date or time or blank for both
            allowOldDates: false
        }, cb);
    }

</script>
</head>
<body onload="onBodyLoad()">
    <div>Date: <span id='date'></span></div>
    <a href="#" onclick="show('')">Show Date and Time Picker</a><br/>
    <a href="#" onclick="show('date')">Show Date Picker</a><br/>
    <a href="#" onclick="show('time')">Show Time Picker</a><br/>
</body>
</html>

```

EmailComposer

The `EmailComposer` plug-in enables you to pass address, subject, and message components to the native e-mail functionality of the iPhone or iPad.

FileUploader

This plug-in creates the `window.plugins.fileUploader` object with two methods — `upload` and `uploadByUri`. The `upload` method takes an absolute path (for example, `/sdcard/media/images/image.jpg`), and `uploadByUri` takes a `content://` URI (for example, `content://media/external/images/media/5`).

Following are the full parameters:

- `server` — URL of the server that will receive the file
- `file` — Path or URI of the file to upload
- `fileKey` — Object with key/value parameters to send to the server
- `params` — Parameter name of the file
- `filename` — Filename to send to the server (defaults to `image.jpg`)
- `mimeType` — MIME type of the uploaded file (defaults to `image/jpeg`)
- `callback` — Success callback. Also receives progress messages during upload.
- `fail` — Error callback

Following is an example:

```
window.plugins.fileUploader.uploadByUri('http://example.com/upload',
    'file://path/to/file.jpg', {foo: 'bar'}, 'myPhoto', 'anImage.jpg',
    'image/jpeg',
    function(result) {
        console.log('Done: ' + result);
    },
    function(result) {
        console.log("Error: " + result);
    },
    function(loaded, total) {
        var percent = 100 / total * loaded;
        console.log('Uploaded ' + percent);
    }
);
```

GapSocket

Use the GapSocket plug-in to create sockets and send/receive messages.

To create a socket, use the following:

```
var mySocket = new GapSocket(hostOrIP, port);
```

To be notified when the socket is connected, use the following:

```
mySocket.onopen = function(){ /* do something meaningful with your life! */ };
```

To listen for incoming messages, use the following:

```
mySocket.onmessage = function(msg){alert(msg);};
```

To be notified of an error, use the following:

```
mySocket.onerror = function(msg){alert("Oh Shit! " + msg);};
```

To send data, use the following:

```
mySocket.send("some data here");
```

To be notified when the socket is closed, use the following:

```
mySocket.onclose = function(){ /*. . . */};
```

Globalization

Use the Globalization plug-in to access different locale, date, number, and Daylight Savings settings via your PhoneGap app.

InAppPurchaseManager

Use the `InAppPurchaseManager` plug-in to allow in-app purchases inside your PhoneGap application. The plug-in creates a `window.plugins.inAppPurchaseManager` object with various methods.

To request product data, use the following:

```
requestProductData(productId, successCallback, failCallback);
```

To make a purchase, use the following:

```
makePurchase(productId, quantity);
```

To restore completed transactions, use the following:

```
restoreCompletedTransactions();
```

Keychain

Use the `Keychain` plug-in to access the iOS device's keychain to get, set, and remove keys.

LocalNotification

Use the `LocalNotification` plug-in to send notifications (includes badge updates!) to the user, even when the application is turned off.

Localizable

Use the `Localizable` plug-in to create localized strings for languages you want to support. By creating language directories for each supported language, and then using the plug-in to pull in the keys, you can create localized versions of your app.

The plug-in creates a `window.plugins.localizable` object with a `get` method.

Following is an example:

```
window.plugins.localizable.get('username', function(result) {
    alert("We got a setting: " + result);
});
```

MapKitPlug

The `MapKitPlug` lets you add maps to your PhoneGap apps, including pins and features like zooming, resizing, and more.

NativeControls

The `NativeControls` plug-in exposes various native interface controls to JavaScript, such as tabs and toolbars. You can create tab bars, show and hide tab bars, add tab bar items, and more.

networkActivityIndicator

Use the `networkActivityIndicator` plug-in to determine whether there is network activity detectable by the device.

PayPalPlugin

The `PayPalPlugin` lets you integrate PayPal payments directly into your app. For this to work, you not only must install the plug-in, but also configure your PayPal account for a testing/sandbox environment.

PowerManagement

The `PowerManagement` plug-in offers access to the device's power-management interface. You can use it to keep apps running without user intervention.

Prompt

The `Prompt` plug-in enables you to create a custom notification with buttons that prompt the user to take an action.

Following is an example:

```
window.plugins.Prompt.show(
    "My Prompt Title",
    function (userText) { ... }, // ok callback
    function () { }, // cancel callback
    "Ok", // ok button title (optional)
    "Cancel" // cancel button title (optional)
);
```

SMSComposer

The `SMSComposer` plug-in enables you to send Short Message Service (SMS) messages directly from your PhoneGap applications.

Following is an example:

```
window.plugins.smsComposer.showSMSComposer('3424221122', 'hello');
```

ScreenShot

The `ScreenShot` plug-in enables you to take screenshots from within your application.

ShareKitPlugin

Use the `ShareKitPlugin` to allow users to share content on social networks from within your PhoneGap application. It has five methods:

- `share(message, url)` — Displays the ShareKit Form to share a message and URL on a social network
- `isLoggedInToTwitter(callback)` — Returns whether or not the user is logged in to Twitter
- `isLoggedInToFacebook(callback)` — Returns whether or not the user is logged in to Facebook
- `logoutFromTwitter()` — Logs the user out of Twitter
- `logoutFromFacebook()` — Logs the user out of Facebook

SplashScreen

Use the `SplashScreen` plug-in to manually control the splash screen from within JavaScript.

Following is an example:

```
<script src="phonegap.js"></script>
<script src="SplashScreen.js"></script>
<script type="text/javascript">

    document.addEventListener("deviceready", function(){
        window.plugins.splashScreen.show('Default.png');

        // do some stuff here ...
        // launch your application, render the layout, ...
        // after all, just hide the SplashScreen again:

        window.plugins.splashScreen.hide();

    }, false);

</script>
```

VolumeSlider

The `VolumeSlider` plug-in lets you add a native volume slider interface controller to your PhoneGap application.

Following is an example:

```
function onDeviceReady()
{
    var volumeSlider = window.plugins.volumeSlider;
    volumeSlider.createVolumeSlider(10,350,300,30); // origin x, origin y,
        width, height
    volumeSlider.showVolumeSlider();
}
```


INDEX

Symbols and Numbers

`$()`, jQuery, 15
8900, BlackBerry Bold 8900, 60

A

`abort()`, 169, 171
 FileWriter, 171
Acceleration, 78
Accelerometer, 12
accelerometer, 73–83
 Android, 75–76
 applications, 74–77
 BlackBerry, 77
 frequency, 80–81
 iOS, 80
 iPhone, 75
 methods, 78–81
`Accelerometer.ERROR_MSG`, 265
`Accelerometer.prototype`
 `.getCurrentAcceleration`, 265–266
`Accelerometer.prototype`
 `.watchAcceleration`, 266–268
`accelerometerSuccess`, 78
accuracy, 104
`addEventListener`, 248
 `onLoad()`, 62
addresses
 BlackBerry, 184
 Contacts, 180
AdPlugin, 340
ADT. *See* Android Development Tools
Ajax, 9
 storage options, 12

`alert()`, 65
alerts, 66–67
`allowEdit`
 Android, 11, 136
 BlackBerry, 136
 Camera, 135
 Palm, 136
altitude, 104
`altitudeAccuracy`, 104
Analytics, 334–335
Android, 2, 36–41
 accelerometer, 75–76
 `allowEdit`, 11, 136
 audio capture, 192
 avd, 37
 Back button, 47–48
 Build, 27
 Camera, 130
 Compass, 86, 90
 ContactOrganization, 185
 contacts, 183
 debugging, 41
 `device.name`, 60
 `device.phonegap`, 60
 `device.platform`, 61
 `device.uuid`, 61
 `device.version`, 61
 downloading PhoneGap, 20–22
 droidgap, 40
 duration, 190, 192
 Emulator, 37–39, 41
 formatted, 184
 functionality, 8
 “Hyperspace,” 75–76
 image capture, 193
 “Lighter,” 76
 Menu button, 49
 mode, 192, 193
 “NFB Films,” 3
 notification.alert, 66
 notification.beep, 67
 offline, 53
 online, 52
 “Orbium,” 4–5
 photo gallery, 130
 plug-ins, 334–339
 reachability, 65
 Search button, 51–52
 Sensorfit, 77
 targets, 37
 vibrate, 68
 video capture, 190
Android Development Tools (ADT), 242
animation, 9
answers to exercises, 213–234
Apache Ant, 23
Apple App Store, 114–115
applications
 accelerometer, 74–77
 Camera, 131–133
 Compass, 88–90
 geolocation, 100–103
 Media, 114–117
 note-taking, 197–211
 Capture, 198–201
 final code, 207–210
 geolocation, 206–207
 metadata, 202–203
 `synch()`, 203–206
ApplicationPreferences, 340–341
appML, 235–236
appMobi, 236
appMobi XDK, 236
`App.prototype`
 `.cancelLoadUrl`, 268

- App.prototype
 - .clearCache, 268
- App.prototype.clearHistory, 268–269
- App.prototype.loadUrl, 268
- App.prototype
 - .overrideBackbutton, 269
- audio
 - capture, 191–192
 - Android, 192
 - BlackBerry, 192
 - options, 191
- avd, 37

B

- Back button, Android, 47–48
- backbutton, 46
- Bada, 8
- Badge, 341
- BarcodeScanner, 335, 341
- BBEdition, 27–28
- beeps, 67
- birthday
 - Contacts, 180
 - iOS, 184
- BlackBerry, 2, 41–42
 - accelerometer, 77
 - allowEdit, 136
 - audio capture, 192
 - Build, 27
 - Camera, 130
 - Compass, 87
 - ContactOrganization, 185
 - contacts, 184
 - device.name, 60
 - device.phonegap, 60
 - device.platform, 61
 - deviceready, 48
 - device.uuid, 61
 - device.version, 61
 - downloading PhoneGap, 220–221
 - duration, 190, 192
 - functionality, 8
 - image capture, 193
 - “Jumper,” 77
 - mode, 192, 193
 - notification.alert, 66–67
 - notification.beep, 67

- offline, 53
- online, 52
- quality, 136
- Simulator, 42
- sourceType, 136
- vibrate, 68
- video capture, 190
- WebWorks SDK, 23
- BlackBerry Bold 8900, 60
- Bluetooth, 335
- body
 - HTML, 126
 - onLoad(), 65
- Browserfield, 48
- Build, 25–27
- build, 42
- Build and Run, 32–33
- Bundle identifier, 36
- BundleFileReader, 341–342
- button, 66
- bytesSent, 173

C

- callback, 66
- CAMERA, 11
- Camera, 10–11, 12, 129–142
 - allowEdit, 135
 - Android, 130
 - applications, 131–133
 - BlackBerry, 130
 - “CameraBag,” 132
 - CameraSuccess, 133
 - destinationType, 135
 - Flickr, 132
 - “Instagram,” 131–132
 - iPhone, 130
 - options, 135–137
 - photo gallery, 130–131
 - “QuadCamera,” 132–133
 - quality, 135
 - sourceType, 133, 135
 - “TiltShift,” 133
 - UI, 138–140
 - webOS, 130
- camera, 10–11
- “CameraBag,” 132
- Camera.DestinationType, 270
- Camera.EncodingType, 270
- camera.getPicture(), 11, 133–135

- Camera.PictureSourceType, 270
- Camera.prototype.getPicture, 271–272
- Camera.prototype
 - .PictureSourceType, 270
- CameraSuccess, 133
- cameraSuccess, 134
- capture, 189–196
 - audio, 191–192
 - Android, 192
 - BlackBerry, 192
 - errors, 193–195
 - images, 192–193
 - Android, 193
 - BlackBerry, 193
 - video, 189–191
 - Android, 190
 - BlackBerry, 190
 - iOS, 191
- Capture, 12
 - Media, 191
 - note-taking application, 198–201
- CaptureError, 193
- Capture.prototype
 - .captureAudio, 273–274
- Capture.prototype
 - .captureVideo, 274–275
- Capture.prototype
 - ._castMediaFile, 274
- cascading style sheet (CSS), 2, 6, 13
 - Media, 126
- categories, 183
 - BlackBerry, 184
 - Contacts, 180
 - iOS, 184
- cell tower triangulation, 99
- CELL_2G, 64
- CELL_3G, 64
- CELL_4G, 64
- ChildBrowser, 336
- Chrome, 28
 - Ripple, 243
- Classes, 36
- clear, 152
- clearWatch(), 80, 92
 - Geolocation, 106
- ClipboardManager, 336
- ClipboardPlugin, 342

clone(), 10
 CMS. *See* content management system
 code, 121
 SQLError, 149
 Compass, 12, 85–98
 Android, 86, 90
 applications, 88–90
 BlackBerry, 87
 iPhone, 86–90
 methods, 90–92
 options, 92
 UI, 94–97
 webOS, 87
 compass.clearWatch, 92
 Compass.ERROR_MSG, 276
 compass.getCurrentHeading, 90–91
 compassOptions, 90
 Compass.prototype
 .clearWatch, 279
 Compass.prototype
 .getCurrentHeading, 276–277
 Compass.prototype
 .watchHeading, 277–278
 compass.watchHeading(), 91–92
 confirmation dialogs, 67
 Connection, 12, 63–66
 Connection.prototype
 .getInfo, 319
 Contact, 9
 contacts, 10, 179–188
 Android, 183
 BlackBerry, 184
 cloning, 182
 errors, 186
 find(), 183
 finding, 182–183
 iOS, 184
 remove(), 182
 saving, 181
 ContactAddress, 181
 ContactError, 186
 ContactName, 180–181, 184–185
 ContactOrganization, 181, 185
 Contact.prototype.clone, 280–281
 Contact.prototype.remove, 279–280

Contact.prototype.save, 281
 Contacts, 12
 contacts.create(), 179
 JSON, 10
 contacts.find(), 182–183
 Contacts.prototype
 .cast, 284
 Contacts.prototype
 .create, 283
 Contacts.prototype
 .find, 283
 ContactView, 336
 content management system (CMS), 243
 Convertigo, 236
 Coordinates, 104
 coords, 103
 copyTo
 DirectoryEntry, 160–161
 FileEntry, 166
 country, 181
 create(), 10
 create, 168
 create table, 147
 createReader, 162
 createWriter, 167–168
 Crypto.prototype.decrypt, 285
 Crypto.prototype.encrypt, 284–285
 Crypto.prototype
 getPlainString, 285
 Crypto.prototype
 gotCryptedString, 285
 CSS. *See* cascading style sheet
 cygwin, 24

D

DatabaseShell.prototype
 .transaction, 329
 DATA_URI, 133
 DatePicker, 342–343
 db.transaction(), 151
 debug.apk, 41
 debugging, Android, 41
 Default.png, 36
 delete, 147
 department, 181
 BlackBerry, 185

iOS, 185
 DEPRECATED, 286, 287
 destinationType, 135
 Device, 13
 device, 59–63
 device.name, 59, 60
 device.phonegap, 59, 60
 device.platform, 59, 61
 Device.prototype.exitApp, 287
 Device.prototype.getInfo, 286
 Device.prototype
 .overrideBackButton, 286
 Device.prototype
 .resetBackButton, 287
 deviceready, 33, 48–49
 backbutton, 48
 BlackBerry, 48
 document object model (DOM), 46
 event listener, 46–47
 offline, 53
 onDeviceReady(), 62
 online, 52
 pause, 50
 resume, 51
 searchbutton, 51
 device.uuid, 59, 61
 device.Version, 59
 device.version, 61–63
 “Diary Mobile,” 2–3
 “Digital Compass,” 89–90
 DirectoryEntry, 158–164
 copyTo, 160–161
 createReader, 162
 getDirectory, 162–163
 getFile, 163
 getMetadata, 158
 getParent, 162
 moveTo, 158–159
 remove, 161
 removeRecursively, 163–164
 toURI, 161
 DirectoryEntry.prototype
 .copyTo, 301
 DirectoryEntry.prototype
 .createReader, 302
 DirectoryEntry.prototype
 .getDirectory, 302

- DirectoryEntry.prototype
 - .getFile, 302–303
- DirectoryEntry.prototype
 - .getMetadata, 301
- DirectoryEntry.prototype
 - .getParent, 301
- DirectoryEntry.prototype
 - .moveTo, 301
- DirectoryEntry.prototype
 - .remove, 302
- DirectoryEntry.prototype
 - .removeRecursively, 303
- DirectoryEntry.prototype
 - .toURI, 302
- DirectoryReader.prototype
 - .readEntries, 300
- displayName
 - Contacts, 180
 - iOS, 184
 - window.openDatabase(), 146
- div, 45
 - toolbar, 96
- document object model (DOM), 15, 45
 - deviceready, 46
 - HTML, 47
 - metadata, 203, 206
 - onDeviceReady(), 63
 - status, 151
 - watchAcceleration(), 79–80
 - watchHeading(), 91–92
- document.addEventListener, 47, 254
- document.getElementById(), 15
- document
 - .removeEventListener, 255
- Documents/tmp, 136
- Dojo Mobile, 236–237
- DOM. *See* document object model
- DONE, 169
- “DoodleJump,” 74–75
- downloading PhoneGap, 17–24
 - Android, 20–22
 - BlackBerry, 220–221
 - iOS, 18–20
 - Symbian, 24
 - webOS, 23–24
- Dreamweaver, 27

- DroidDB.prototype
 - .completeQuery, 325–326
- DroidDb.prototype.fail, 326
- DroidDB_Rows.prototype
 - .item, 324
- DroidDB_Tx.prototype
 - .executeSql, 328–329
- DroidDB_Tx.prototype
 - .queryComplete, 327–328
- DroidDB_Tx.prototype
 - .queryFailed, 328
- droidgap, 40
- drop, 147
- duration
 - Android, 190, 192
 - BlackBerry, 190, 192
 - iOS, 191
 - options, 190, 191

E

- Easy APNS, 237
- Eclipse
 - downloading PhoneGap, 20–21
 - JavaScript Development Model (JSDT), 242
- EmailComposer, 343
- emails
 - BlackBerry, 184
 - Contacts, 180
- EmbedJS, 237
- EMPTY, 169
- Emulator, 6, 7
 - Android, 37–39, 41
- error, 151, 169
 - executeSql(), 148
 - FileWriter, 171
- errors, 149–152
 - capture, 193–195
 - contacts, 186
 - files, 173–176
 - Media, 121–123
- errorCallback
 - copyTo, 160, 166
 - createWriter, 167
 - File, 168
 - FileError, 162
 - getDirectory, 162
 - getFile, 163
 - getMetadata, 164
 - getParent, 162, 167

- moveTo, 159, 165
- remove, 161, 166
- removeRecursively, 163
- ETHERNET, 64
- events, 45–57

Android

- Back button, 47–48
- Menu button, 49
- offline, 53
- online, 52
- Search button, 51–52
- BlackBerry
 - deviceready, 48
 - offline, 53
 - online, 52
- deviceready, 48–49
 - backbutton, 48
 - BlackBerry, 48
 - document object model (DOM), 46
 - event listener, 46–47
 - offline, 53
 - online, 52
 - pause, 50
 - resume, 51
 - searchbutton, 51
- document object model (DOM), 45
 - deviceready, 46
 - HTML, 47
- iOS
 - Objective-C, 50
 - offline, 53
 - online, 52
- JavaScript, 45
 - offline, 46, 53
- RIM, 4
- event listener, 46–47
- Events, 13
- exclusive, 168
- executeSql(), 147
 - error, 148
 - success, 148

F

- Facebook, 102
- familyName, 180
 - BlackBerry, 185
- File, 13
 - FileEntry, 168

- files, 157–177
 - DirectoryEntry, 158–164
 - errors, 173–176
 - FileEntry, 164–168
 - LocalFileSystem, 168–169
 - reading, 169–171
 - transferring, 172–173
 - writing, 171–172
- FileEntry, 158, 164–168
 - copyTo, 166
 - createWriter, 167–168
 - File, 168
 - getMetadata, 164–165
 - getParent, 167
 - moveTo, 165
 - remove, 166–167
 - toURI, 166
- FileEntry.prototype
 - .copyTo, 303
- FileEntry.prototype
 - .createWriter, 305
- FileEntry.prototype
 - .file, 305
- FileEntry.prototype
 - .getMetadata, 303–304
- FileEntry.prototype
 - .getParent, 304
- FileEntry.prototype
 - .moveTo, 304
- FileEntry.prototype
 - .remove, 304
- FileEntry.prototype
 - .toURI, 304
- FileError, 174
 - errorCallback, 162
- fileKey, 173
- FileMgr.prototype
 - .getFileBasePaths, 288
- FileMgr.prototype
 - .getFileProperties, 288
- FileMgr.prototype
 - .getFreeDiskSpace, 289
- FileMgr.prototype
 - .readAsDataURL, 289
- FileMgr.prototype
 - .readAsText, 289
- FileMgr.prototype
 - .testDirectoryExists, 289
- FileMgr.prototype
 - .testFileExists, 289

- FileMgr.prototype
 - .testSaveLocationExists, 288–289
- FileMgr.prototype
 - .truncate, 289
- FileMgr.prototype
 - .write, 289
- fileName
 - FileUploadOptions, 173
 - FileWriter, 171
- FileReader, 169–171
- FileReader.prototype
 - .abort, 290–291
- FileReader.prototype
 - .readAsArrayBuffer, 294
- FileReader.prototype
 - .readAsBinaryString, 294
- FileReader.prototype
 - .readAsDataURL, 292–293
- FileReader.prototype
 - .readAsText, 291–292
- FileSystem, 157–158
- FileTransfer, 172–173, 204
- FileTransferError, 172, 174
- FileTransfer.prototype
 - .upload, 309
- FileUploader, 337, 343–344
- FileUploadOptions, 172–173
- FileUploadResult, 172–173
- FILE_URI, 133
 - Documents/tmp, 136
- FileWriter, 171–172
- FileWriter.prototype
 - .abort, 295–296
- FileWriter.prototype.seek, 297–298
- FileWriter.prototype
 - .truncate, 298
- FileWriter.prototype
 - .write, 296
- films, 3
- find(), contacts, 10, 183
- Firefox, 28
- flags, 168
- Flickr, 132
- for, 149
- formatted, 180, 181
 - Android, 184
 - BlackBerry, 185
 - iOS, 185
- “foursquare,” 100–102

- frameworks, 14–15
- Frameworks, 36
- frequency, 80–81
- FtpClient, 337
- fullPath
 - DirectoryEntry, 158
 - FileEntry, 164

G

- GapSocket, 344
- gender, 180
- geolocation, 11–12, 99–111
 - applications, 100–103
 - note-taking application, 206–207
 - user interface (UI), 108–110
- Geolocation, 11–12, 13
 - clearWatch(), 106
 - Coordinates, 104
 - getCurrentPosition(), 105
 - methods, 105–107
 - Position, 103–104
 - PositionError, 104
 - watchPosition(), 105–106
- geolocation.clearWatch, 106
- geolocation
 - .getCurrentPosition(), 103, 105
- Geolocation.prototype
 - .clearWatch, 313
- Geolocation.prototype
 - .fail, 313
- Geolocation.prototype
 - .getCurrentPosition, 310–311
- Geolocation.prototype
 - .success, 312–313
- Geolocation.prototype
 - .watchPosition, 311–312
- Geolocation.usePhoneGap, 313
- getCurrentAcceleration, 78–79
- getCurrentAcceleration(), 79
- getCurrentHeading, 90–91
- getCurrentPosition()
 - Geolocation, 105
 - Media, 118
- getDirectory, 162–163
- getDuration(), 118–119
- getElementById, 63

getFile, 163
 getItem, 152
 getMetadata
 DirectoryEntry, 158
 FileEntry, 164–165
 getParent
 DirectoryEntry, 162
 FileEntry, 167
 getPicture(), 133–135
 github, 333
 givenName, 181
 BlackBerry, 185
 Globalization, 344
 globally unique identifier (GUID), 180
 GloveBox, 237
 Google+, 39
 geolocation, 102–103
 Gowalla, 100–102
 graphics, 9
 high-resolution, 11
 graybutton, 126
 GUID. *See* globally unique identifier
 GWT Mobile, 238

H

hashtables, 253
 heading, 104
 high-resolution graphics, 11
 history, 8–9
 home, 96
 honorificPrefix, 181
 BlackBerry, 185
 honorificSuffix, 181
 BlackBerry, 185
 HTML, 2, 6, 13
 body, 126
 document object model (DOM), 47
 Media, 126
 onDeviceReady(), 33
 watchAcceleration(), 79–80
 watchHeading(), 91
 Xcode, 34
 HTML5, 13
 Jo, 240
 Ripple, 243
 storage options, 12

HTML5SQL, 238
 “Hyperspace,” 75–76

I

icon.png, 36
 id, 45
 BlackBerry, 184
 Contacts, 180
 home, 96
 IDE. *See* integrated development environment
 IE. *See* Internet Explorer
 images
 capture, 192–193
 Android, 193
 BlackBerry, 193
 options, 193
 Impact, 239
 ims, 180
 InAppPurchase
 Manager, 345
 individual, 96
 initialization, 13, 253
 insert, 147
 insertId, 148
 insertRow(), 205
 Inspect Element, 143–144
 “Instagram,” 131–132
 Installer, 19
 integrated development environment (IDE), 27–28
 Internet Explorer (IE), 28
 iOS
 accelerometer, 80
 Build, 27
 ContactOrganization, 185
 contacts, 184
 downloading PhoneGap, 18–20
 duration, 191
 formatted, 185
 getCurrentAcceleration(), 79
 limit, 191, 192, 193
 mode, 191, 192, 193
 Objective-C, 50
 offline, 53
 online, 52
 plug-ins, 339–347
 quality, 136

startAudioRecord(), 120
 video capture, 191
 Xcode, 28

IP address, 99

iPad

 iWebKit, 240
 “Just One More,” 4
 iPhone, 2
 accelerometer, 75
 Camera, 130
 Compass, 86–90
 device.name, 60
 device.phonegap, 60
 device.platform, 61
 device.uuid, 61
 device.version, 61
 “Digital Compass,” 89–90
 functionality, 8
 iWebKit, 240
 “Just One More,” 4
 notification.alert, 66
 notification.beep, 67
 “Orbium,” 4–5
 photo gallery, 130
 reachability.code, 65
 “Spyglass,” 88–89
 “Super Monkey Ball,” 75
 vibrate, 68
 Xcode, 33

iPod, 114

iScroll, 239

isDirectory

 DirectoryEntry, 158
 FileEntry, 164

isFile

 DirectoryEntry, 158
 FileEntry, 164

item(), 149

iWebKit, 240

J

Java Development Kit (JDK), 22–23

 PATH, 22

JavaScript, 2, 6, 13

 alert(), 65

 events, 45

 NS Basic/App Studio, 243

 qoxxdoo, 243

 Xcode, 34

JavaScript Development Tools (JSDT), 242
 JavaScript Game Engine, 239
 JavaScript Object Notation (JSON)
 contacts.create(), 10
 Lawnchair, 241–242
 storage options, 12
 JDK. *See* Java Development Kit
 Jo, 240
 jQTouch, 14–15, 240
 Media, 126
 jqtouch, 94
 jQuery, 9, 13
 \$, 15
 jQuery Mobile, 240–241
 JSDT. *See* JavaScript Development Tools
 JSON. *See* JavaScript Object Notation
 “Jumper,” 77
 “Just One More,” 4

K

key, 152
 Keychain, 345
 key/value pairs, 144

L

latency, 13
 latitude, 104
 Lawnchair, 241–242
 length, 171
 “Lighter,” 76
 limit
 iOS, 191, 192, 193
 options, 190, 191, 193
 LOADING, 169
 load-simulator, 42
 Local Storage, 145–146
 LocalFileSystem, 158, 168–169
 LocalFileSystem.PERSISTENT, 169
 LocalFileSystem.prototype._
 castDate, 307
 LocalFileSystem.prototype._
 castEntries, 307
 LocalFileSystem
 .prototype._castEntry, 306–307

LocalFileSystem.prototype
 ._castFS, 306
 LocalFileSystem
 .prototype
 ._createEntry, 307
 LocalFileSystem.prototype
 .requestFileSystem, 305–306
 LocalFileSystem.prototype
 .resolveLocalFile
 SystemURI, 306
 LocalFileSystem
 .TEMPORARY, 169
 locality, 181
 Localizable, 345
 LocalNotification, 345
 localStorage, 144–145, 152–153
 longitude, 104

M

MAC. *See* Media Access Control
 magnetometer, 85–86
 Mamoru Tokashiki, 116
 MapKitPlug, 345
 MDS. *See* Mobile Developer Solutions
 Media, 13, 113–128
 applications, 114–117
 Capture, 191
 errors, 121–123
 getCurrentPosition(), 118
 getDuration(), 118–119
 iPod, 114
 mediaError, 117
 mediaSuccess, 117
 methods, 117–121
 Pandora, 115–116
 pause(), 119–120
 play(), 119
 release(), 120
 src, 117
 startRecord(), 120–121
 stop(), 121
 UI, 125–127
 W3C, 113
 Media Access Control (MAC), 99
 Media Capture, 10–11, 12
 MediaError, 121–123
 mediaError, 117
 MediaError.MEDIA_ERR_
 ABORTED, 121
 MediaFile.prototype
 .getFormatData, 272–273
 media.pause(), 119–120
 media.play(), 119
 Media.prototype
 .getCurrentPosition, 316
 Media.prototype
 .getDuration, 316
 Media.prototype.pause, 316
 Media.prototype.play, 316
 Media.prototype.release, 317
 Media.prototype.seekTo, 316
 Media.prototype
 .startRecord, 316
 Media.prototype.stop, 316
 Media.prototype
 .stopRecord, 317
 media.release(), 120
 Media.startRecording(), 121
 media.stop(), 121
 media.stopRecording, 121
 mediaSuccess, 117
 Menu button, Android, 49
 menubutton, 46, 49
 message, 66
 MediaError, 121
 SQLException, 149
 metadata, note-taking application, 202–203
 metadata, document object model (DOM), 203, 206
 methods
 accelerometer, 78–81
 Compass, 90–92
 Geolocation, 105–107
 Media, 117–121
 middleName, 181
 BlackBerry, 185
 mimeType, 173
 Mobile Developer Solutions (MDS), 242
 mobileFX Studio7, 242
 Mobilizer, 236
 mode
 Android, 192, 193
 BlackBerry, 192, 193
 iOS, 191, 192, 193
 options, 190, 191, 193

- name, 181
 - BlackBerry, 185
 - DirectoryEntry, 158
 - FileEntry, 164
 - FileSystem, 157
 - iOS, 185
 - window.openDatabase(), 146
- NativeControls, 346
- navigator.device.capture
 - .captureAudio(), 191
- navigator.device.capture
 - .captureImage(), 192
- navigator.device.capture
 - .captureVideo(), 189
- navigator.fileMgr
 - .readAsDataURL, 293–294
- navigator.fileMgr.truncate, 298–299
- navigator.fileMgr.write, 296–297
- navigator.geolocation
 - .getCurrentPosition(), 206
- navigator.network
 - .isReachable, 65
- Network, 13
- networks, 63–66
- networkActivityIndicator, 346
- newName
 - copyTo, 160, 166
 - moveTo, 159, 165
- Nexus One, 60
- “NFB Films,” 3
- nickname
 - BlackBerry, 184
 - Contacts, 180
- NONE, 64
- note, 180
- Notepad, 27
- note-taking application, 197–211
 - Capture, 198–201

- final code, 207–210
- geolocation, 206–207
- metadata, 202–203
- `synch()`, 203–206
- Notification, 13
- notifications, 66–70
 - alerts, 66–67
 - beeps, 67
 - confirmation dialogs, 67
 - vibrations, 68
- `notification.alert`, 66–67
- `notification.beep`, 67
- `notification.confirm`, 67
- `Notification.prototype`
 - `.activityStart`, 321
- `Notification.prototype`
 - `.activityStop`, 321
- `Notification.prototype`
 - `.alert`, 320
- `Notification.prototype`
 - `.beep`, 322
- `Notification.prototype`
 - `.blink`, 321–322
- `Notification.prototype`
 - `.confirm`, 320–321
- `Notification.prototype`
 - `.progressStart`, 321
- `Notification.prototype`
 - `.progressStop`, 321
- `Notification.prototype`
 - `.progressValue`, 321
- `Notification.prototype`
 - `.vibrate`, 322
- `notification.vibrate`, 68
- NS Basic/App Studio, 243

- Objective-C, 8
 - iOS, 50
 - Xcode, 36
- offline, 46, 53
- onabort, 169
 - FileWriter, 171
- onClick, 96
- onclick, 45
- onDestroy, 248, 252
- onDeviceReady, 253
- onDeviceReady(), 47, 247
 - deviceready, 62

- document object model (DOM), 63
- HTML, 33
- `window.openDatabase()`, 146
- `onDOMContentLoaded`, 247, 251
- `onerror`, 169
 - `FileWriter`, 171
- online, 46, 52–53
- `onLoad()`
 - `addEventListener`, 62
 - body, 65
 - onload, 63
- onload, 169
 - `onLoad()`, 63
- onloadend, 169
- onloadstart, 169
- onPause, 248, 252
- `onPhoneGapInfoReady`, 247, 252
- `onPhoneGapInit`, 247, 252
- `onPhoneGapReady`, 247
- `onResume`, 247
- onrollover, 45
- `onSuccess`, 103
- `onwrite`, 171
- `onwriteend`, 171
- `onwritestart`, 171
- Open With Finder, 94–95
- OpenCore, 120
- `openDatabase()`, 151
- operating system, 61–63
- options
 - audio, 191
 - `getDirectory`, 162
 - `getFile`, 163
 - images, 193
 - video, 190
- “Orbium,” 4–5
- organizations
 - BlackBerry, 184
 - Contacts, 180
- Osmek, 243
- `overflow:scroll`, 239

- Palm, 136
- “Pandora,” 115–116
- params, 173
- parent

- copyTo, 160, 166
- moveTo, 159, 165
- parsing, 13
- Passion, 60
- PATH, 22
- path
 - getDirectory, 162
 - getFile, 163
- pause event, 46, 50
- pause(), method, 119–120
- PayPalPlugin, 338, 346
- PhoneGap.addConstructor, 251,
 - 272, 275–276, 279, 284, 285,
 - 287, 289, 307–308, 313–314,
 - 319, 322, 324, 331
- PhoneGap.addPlugin, 251
- PhoneGap.addResource, 272,
 - 276, 279–280, 284, 285, 287,
 - 308, 314, 324
- PhoneGap.callbackError,
 - 259–260
- PhoneGap.callbackStatus(),
 - 257
- PhoneGap.Channel, 249
- PhoneGap.Channel.join,
 - 250–251, 253–254
- PhoneGap.Channel
 - .prototype.fire, 250
- PhoneGap.Channel
 - .prototype.subscribe,
 - 249–250
- PhoneGap.Channel
 - .prototype
 - .unsubscribe, 250
- PhoneGap.clone,
 - 256–257
- PhoneGap.close, 264
- PhoneGap
 - .deviceReadyChannels
 - Map, 253
- PhoneGap.exec, 258–259, 286
- PhoneGap.fireEvent, 255
- PhoneGap.hasResource,
 - 248, 284, 285, 287, 318,
 - 320, 322
- PhoneGap.includeJavascript,
 - 264–265
- PhoneGap.initialization
 - Complete, 253
- PhoneGap.js, 247–332
- PhoneGap.JSCallback,
 - 261–264

- PhoneGap.Media, 317
- PhoneGap.Media
 - getMediaObject, 317
- PhoneGap.mediaObjects, 317
- PhoneGap.Media.onStatus,
 - 317–318
- PhoneGap.onDOMContentLoaded,
 - 254
- PhoneGap.onPhoneGap.Init
 - .fire(), 254
- PhoneGap.resources, 248
- PhoneGap.run_command,
 - 260–261
- PhoneGap.shutting
 - Down, 252
- PhoneGap.stringify,
 - 255–256
- PhoneGap.UUIDcreate
 - Part, 264
- phoneNumbers
 - BlackBerry, 184
 - Contacts, 180
- photo gallery, 130–131
- PHOTOLIBRARY, 11
- photos
 - BlackBerry, 184
 - Contacts, 180
 - iOS, 184
- play(), 119
- plist
 - Bundle identifier, 36
 - Resources, 35
 - Xcode, 35–36
- plug-ins, 333–347
 - Android, 334–339
 - iOS, 339–347
- PluginResult, 257
- Plugins, 36
- plugins.xml, 21
- Position, 103–104
- position, 171
- PositionError, 104
- PositionError.PERMISSION_
 - DENIED, 104
- PositionError.POSITION_
 - UNAVAILABLE, 104
- PositionError.TIMEOUT, 104
- postalCode, 181
- PowerManagement, 338, 346
- pref, 181, 185
- project.properties, 42
- Prompt, 346

- push, 237
- puzzle game, 4–5

Q

- qooxdoo, 243
- “QuadCamera,” 132–133
- quality
 - BlackBerry, 136
 - Camera, 135
 - iOS, 136
 - Palm, 136
- querySuccess(), 149

R

- Radio Frequency Identification (RFID), 99
- randomUUID(), 205
- reachability, 65
- reachability.code, 65
- readAsDataURL, 170
- readAsText, 170–171
- readyState, 169
 - FileWriter, 171
- region, 181
- release(), 120
- remove
 - DirectoryEntry, 161
 - FileEntry, 166–167
- remove(), 10, 182
- removeItem, 152
- removeRecursively, 163–164
- requestFileSystem, 169
- requestFileSystem(), 158
- Research in Motion (RIM), 48
- resolveLocalFile
 - SystemURI, 169
- Resources
 - Default.png, 36
 - icon.png, 36
 - plist, 35
- response, 173
- responseCode, 173
- RESTful, 9
 - Urban Airship, 244
- result, 169
- results.rows.length, 149
- resume, 46, 50–51
- RFID. *See* Radio Frequency Identification

RIM. *See* Research in Motion
 Ripple, 243
 root, 157
 rowAffected, 148
 rows, 148
 item(), 149

S

Safari, 28
 save(), 10
 Screenshot, 346
 SDK. *See* Software Development Kit
 Search button, Android, 51–52
 searchbutton, 46, 51–52
 seek(), 171
 select, 148–149
 Sencha touch, 243
 “Sensorfit,” 77
 Session Storage, 145–146
 sessionStorage, 144–145
 setItem, 152
 Share, 338
 ShareKitPlugin, 347
 Simulator
 BlackBerry, 42
 Symbian, 42
 webOS, 42
 Xcode, 32
 size, 146
 SMSComposer, 346
 SoftKeyBoard, 338
 Software Development Kit (SDK), 17
 appMobi, 236
 webOS, 23
 WebWorks, 23
 Sony Ericsson WebSDK Packager, 243–244
 sourceType
 BlackBerry, 136
 Camera, 133, 135
 Palm, 136
 speed, 104
 SplashScreen, 347
 “Spyglass,” 88–89
 SQL query
 select, 148–149
 SQLTransaction, 147–148
 SQLiteError, 149–150

SQLite, 12
 NS Basic/App Studio, 243
 SQLTransaction, 147–148
 src, 117
 startAudioRecord(), 120
 startRecord(), 120–121
 startRecording(), 121
 status, 151
 StatusBarNotification, 338
 statusmessages, 45
 stop(), 121
 stopAudioRecord(), 121
 stopRecording(), 121
 Storage, 13, 146
 storage options, 143–156
 HTML5, 12
 streetAddress, 181
 success
 executeSql(), 148
 requestFileSystem(), 158
 transaction, 148
 successCallback, 158
 copyTo, 160, 166
 createWriter, 167
 File, 168
 getDirectory, 162
 getFile, 163
 getMetadata, 164
 getParent, 162, 167
 moveTo, 159, 165
 remove, 161, 166
 removeRecursively, 163
 successCompass, 91
 “Super Monkey Ball,” 75
 Symbian
 Build, 27
 downloading PhoneGap, 24
 Simulator, 42
 synch(), 203–206

T

targets, 37
 TextMate, 27
 themes, 94
 “TiltShift,” 133
 timestamp, 103
 title, 66, 181
 Android, 185
 BlackBerry, 185
 iOS, 185

toolbar, 96
 toURI
 DirectoryEntry, 161
 FileEntry, 166
 transaction, 148
 truncate, 171
 TTS, 338–339
 tx.executeSql(), 148
 type, 181, 185

U

UI. *See* user interface
 Unify, 244
 Universally Unique Identifier (UUID), 60, 61
 UNKNOWN, 64
 Urban Airship, 244
 urls
 BlackBerry, 184
 Contacts, 180
 user interface (UI), 13
 Camera, 138–140
 Compass, 94–97
 Media, 125–127
 UUID. *See* Universally Unique Identifier

V

var CaptureAudioOptions, 275
 var CaptureVideoOptions, 275
 var Connection, 318–319
 var Contact, 279
 var ContactError, 279
 var ContactField, 281–282
 var ContactFindOptions, 284
 var ContactName, 281
 var ContactOrganization, 282
 var Contacts, 282–283
 var Coordinates, 323
 var CupcakeLocalStorage, 330–331
 var Device, 285–286
 var DirectoryEntry, 300–301
 var DirectoryReader, 300
 var DroidDB, 325
 var DroidDB_openDatabase, 329

var DroidDb_Query, 326–327
 var DroidDB_Result, 325
 var DroidDB_Rows, 324
 var DroidDB_Tx, 327
 var File, 288
 var FileEntry, 303
 var FileProperties, 287–288
 var FileReader, 290
 var FileSystem, 300
 var FileTransfer, 308
 var FileTransferError, 308–309
 var FileUploadOptions, 309
 var FileUploadResult, 308
 var FileWriter, 294–295
 var Flags, 299–300
 var Geolocation, 310
 var Media, 314–315
 var Metadata, 299
 var Notification, 320
 var Position, 322–323
 var PositionError, 310, 324
 var PositionOptions, 323
 version, 146
 versions, 60
 operating system, 61–63
 vi, 27
 vibrate, 68
 vibrations, 68
 video
 capture, 189–191
 Android, 190
 BlackBerry, 190
 iOS, 191
 content, 4
 options, 190
 Virtual Box, 23
 “Voice Memos,” 114

“Voice Recorder,” 116–117
 voles, 60
 VolumeSlider, 347

W

W3C. *See* World Wide Web Consortium
 watchAcceleration(), 79–80
 watchHeading(), 91–92
 watchPosition(), 105–106
 Web Database, 144
 WebIntent, 339
 webOS
 Build, 27
 Camera, 130
 Compass, 87
 downloading PhoneGap, 23–24
 functionality, 8
 notification.alert, 66–67
 notification.beep, 67
 “Orbium,” 4–5
 SDK, 23
 Simulator, 42
 vibrate, 68
 WebWorks SDK, 23
 whitebutton, 126
 WIFI, 64
 window, 60
 window.localStorage, 152–153
 window.localStorage.clear(), 152
 window.localStorage.getItem(), 152

window.localStorage.key(), 152
 window.localStorage.removeItem(), 152
 window.openDatabase(), 146, 204
 window.plugins, 251
 Windows Phone 7, 8
 Wink, 245
 World Wide Web Consortium (W3C)
 Contact, 9
 Media, 113
 wrappers, 2
 write(), 171

X

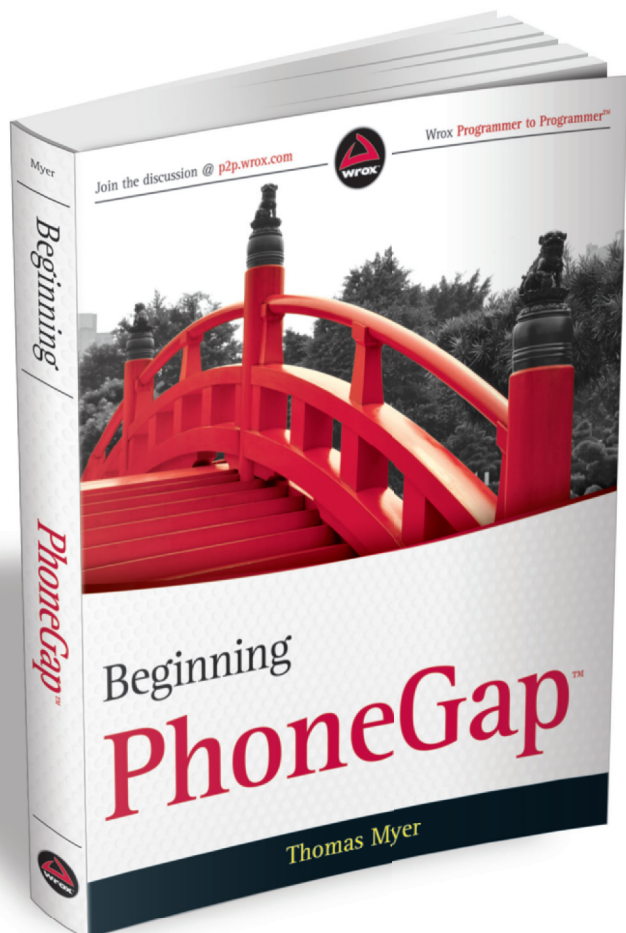
Xcode, 5, 6, 31–36
 Build and Run, 32–33
 downloading PhoneGap, 19
 iOS, 28
 iPhone, 33
 jqtouch, 94
 new project, 20
 Objective-C, 36
 plist, 35–36
 Simulator, 32
 themes, 94
 XmlHttpRequests, 9
 XUI, 245
 xuijs, 14

Z

Zepto.js, 245

Try Safari Books Online FREE for 15 days + 15% off for up to 12 Months*

Read this book for free online—along with thousands of others—
with this 15-day trial offer.



With Safari Books Online, you can experience searchable, unlimited access to thousands of technology, digital media and professional development books and videos from dozens of leading publishers. With one low monthly or yearly subscription price, you get:

- Access to hundreds of expert-led instructional videos on today's hottest topics.
- Sample code to help accelerate a wide variety of software projects
- Robust organizing features including favorites, highlights, tags, notes, mash-ups and more
- Mobile access using any device with a browser
- Rough Cuts pre-published manuscripts

START YOUR FREE TRIAL TODAY!

Visit www.safaribooksonline.com/wrox22 to get started.

*Available to new subscribers only. Discount applies to the Safari Library and is valid for first 12 consecutive monthly billing cycles. Safari Library is not available in all countries.



An Imprint of **WILEY**
Now you know.

Safari 
Books Online

www.it-ebooks.info



Programmer to Programmer™

Connect with Wrox.

Participate

Take an active role online by participating in our P2P forums @ p2p.wrox.com

Wrox Blox

Download short informational pieces and code to keep you up to date and out of trouble

Join the Community

Sign up for our free monthly newsletter at newsletter.wrox.com

Wrox.com

Browse the vast selection of Wrox titles, e-books and blogs and find exactly what you need

User Group Program

Become a member and take advantage of all the benefits

Wrox on **twitter**

Follow @wrox on Twitter and be in the know on the latest news in the world of Wrox

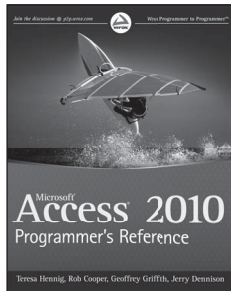
Wrox on **facebook**

Join the Wrox Facebook page at facebook.com/wroxpress and get updates on new books and publications as well as upcoming programmer conferences and user group events

Contact Us.

We love feedback! Have a book idea? Need community support?
Let us know by e-mailing wrox-partnerwithus@wrox.com

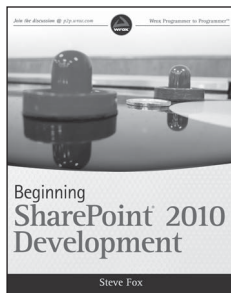
Related Wrox Books



Access 2010 Programmer's Reference

ISBN: 978-0-470-59166-6

Microsoft Access 2010 offers a rich set of features and design tools for storing, analyzing, and viewing data, along with the inherent ability to connect to a large variety of data sources. With this book, you'll discover the benefits of using the newest features to build small- to medium-scale database applications, integrate with other programs, create web applications, and upgrade existing solutions.



Beginning SharePoint 2010 Development

ISBN: 978-0-470-58463-7

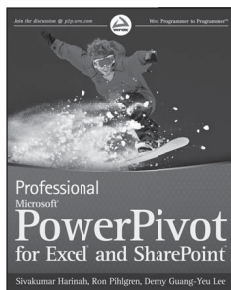
As a first-class platform that has evolved significantly since its previous release, SharePoint 2010 now provides several advancements for the developer (native Visual Studio tools support, services and extensibility enhancements, and APIs), and many new capabilities (improved data programmability, line-of-business interoperability, and sandboxed solutions). With this authoritative guide, industry veteran Steve Fox provides expert guidance on developing applications as he walks you through the fundamentals of programming, explores the developer toolset, and provides practical code examples to teach you how to use many of SharePoint's new developer features. You'll quickly discover how SharePoint's rich platform supports great collaboration, extensibility, and interoperability.



Excel VBA 24-Hour Trainer

ISBN: 978-0-470-89069-1

Virtually every manual task in Excel can be automated with VBA, which increases your productivity and saves enormous amounts of time. This unique book-and-DVD package prepares you to get more out of Excel by using Visual Basic for Applications (VBA) to automate many routine or labor-intensive Excel tasks. Microsoft Excel MVP and author Tom Utts walks through a series of lessons and illustrations, while the accompanying DVD provides demos and screencasts to complement each lesson.



Professional Microsoft PowerPivot for Excel and SharePoint

ISBN: 978-0-470-58737-9

PowerPivot brings the power of Microsoft Business Intelligence to Office 2010! With PowerPivot, Microsoft brings the strength of Microsoft's Business Intelligence (BI) toolset to Excel and PowerPoint users. Authored by members of the Microsoft team behind the creation of PowerPivot, this book shows you how to use PowerPivot for Excel to create compelling BI solutions, perform data analysis, and achieve unique business insight. You'll learn how to use PowerPivot for SharePoint to share your BI solutions and collaborate with others. And your organization will learn how to use SQL Server 2008 R2 management tools to achieve more efficient results.