



Learn by doing: less theory, more results

PhoneGap 3

A guide to building cross-platform apps using the W3C standards-based Cordova/PhoneGap framework

Foreword by Luca Filigheddu, Head of Developer Evangelism EMEA, BlackBerry

Beginner's Guide

Giorgio Natili

[PACKT] open source[®]
community experience distilled
PUBLISHING

PhoneGap 3 Beginner's Guide

A guide to building cross-platform apps using the W3C standards-based Cordova/PhoneGap framework

Giorgio Natili



BIRMINGHAM - MUMBAI

PhoneGap 3 Beginner's Guide

Copyright © 2013 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: September 2011

Second Edition: September 2013

Production Reference: 1170913

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78216-0-984

www.packtpub.com

Cover Image by Abhishek Pandey (abhishek.pandey1210@gmail.com)

Credits

Author

Giorgio Natili

Project Coordinator

Anugya Khurana

Reviewers

Alessandro Crugnola
Michael Kock
Brian Rinaldi
Chris Tagliaferro

Proofreader

Stephen Copestake

Indexer

Hemangini Bari

Acquisition Editor

Usha Iyer

Production Coordinator

Arvindkumar Gupta

Lead Technical Editor

Sweny Sukumaran

Cover Work

Arvindkumar Gupta

Technical Editors

Gauri Dasgupta
Joyslita D'souza
Veronica Fernandes
Dipika Gaonkar
Sampreshita Maheshwari
Nitee Shetty

Foreword

This book is not only just about PhoneGap, it's about mobile development with web standards. In the past years, PhoneGap evolved very quickly, and throughout the book, the author tells the story of the evolution of this framework with passion, keeping the reader focused not only on PhoneGap itself but also on mobile development's best practices. I personally know the author, because I was involved with him in several community-driven meetings; he's passionate about his job and this is pretty clear when reading the book.

This book is relevant for both beginner and intermediate readers; the former will get the most out of the frequently asked questions that are answered, and the latter will get a solid knowledge about the PhoneGap APIs from this book. This book is not only a detailed reference to PhoneGap APIs, but it's actually a practical guide that shows how to use them and how to be very productive.

This book starts with the building blocks of PhoneGap, and, in a step-by-step manner, brings the reader to a deep understanding of the framework architecture and the way to extend it; each chapter contains a practical example tested on Android 4.x, BlackBerry 10.x, iOS 5.x and 6.x, and Windows Phone 8. The examples given in this book will help the readers to create the building blocks of their mobile applications.

It should be no surprise that the author is supportive of initiatives such as this book. By providing an angle enriched by his day-by-day development experience, the author guides the user of PhoneGap through a learning experience, which is complementary to the user documentation provided by the PhoneGap community.

Luca Filigheddu
Head of Developer Evangelism EMEA, BlackBerry

About the Author

Giorgio Natili is an author, educator, community leader, W3C member, and founder of www.gnstudio.com—a boutique Rome-based development and design studio, specializing in engaging and accessible web and mobile experiences. A strong proponent of agile development practices, his areas of expertise include standards-based application development, client-side scripting, gaming, and video streaming. His previous speaking engagements include Adobe Max, 360|Flex, FITC, XP 2010 and 2012, 360|Stack 2013, and several community-driven conferences. Also, he is the founder of the community www.codeinvaders.net and the main organizer of the Mobile Tea, Italy conference.

A very special thanks to Stefano Masciocchi for the design of the itinero app used in the examples of the book.

About the Reviewers

Alessandro Crugnola grew up in a little town near Varese, Italy, where he first studied art in Bologna and then advertising in Perugia. He discovered the joy of programming during his internship.

ActionScript has been his passion for more than 10 years. During this period, he also created an ActionScript editor in Python, when Flash was the only IDE for ActionScript programmers.

In 2008, he started working with Alittleb.it, a company based in Milan, working primarily on Flash apps. At the same time, he also worked for Aviary, on its Flash app for vector manipulation. He later moved to New York in 2011, to start working on the Aviary Android App and SDK.

I'd really like to thank my parents for allowing me to follow my ambitions.

Michael Koch is a web content strategist and developer content specialist with more than 15 years' experience in the technical sector. Michael has an old-school commitment to clear prose and simple instructions, an unhealthy obsession with best practices, and a passion for gadgets and technologies that enrich our daily experiences.

Brian Rinaldi has been a developer since 1996, during which time he picked up on Flash and ColdFusion. Since then, Brian has worked on a number of languages and technologies, recently focusing on HTML and JavaScript development. Brian works for Adobe Systems and is a frequent author and speaker. He maintains a blog at www.remotesynthesis.com as well as a popular site for web and mobile developer tutorials at www.flippinawesome.org. You can follow Brian on Twitter @remotesynth.

Chris Tagliaferro has been working in the web and mobile applications field for the past 10 years. He is currently developing applications using Phonegap, jQuery, and HTML5 powered by Coldfusion as the server-side technology. He has developed several applications that are used company-wide and heads up a small team of developers. He currently works at Dynamics Research Corp. in Andover, MA as a Software Engineer developing cross-platform applications, heavily focused on Phonegap, Coldfusion, jQuery and HTML5.

I'd like to thank Packtpub for giving me the opportunity to work on this book; it's been a great experience. I would also like to thank Dave and Jim at work for giving me the creative freedom to learn more and more as newer technologies come out. And last but not least, my girlfriend Jen for being incredibly supportive with everything that I do.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- ◆ Fully searchable across every book published by Packt
- ◆ Copy and paste, print and bookmark content
- ◆ On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

To my beloved father...

(l'architetto!)

Table of Contents

Preface	1
Chapter 1: Getting Started with PhoneGap	7
PhoneGap and Apache Cordova	7
The evolution of PhoneGap from 1.4 to 3.x	8
A note about operating systems	9
A note about the command-line tool	11
Installing PhoneGap	11
Installing dependencies	12
Setting up your development environment	13
Time for action – setting up Android using PhoneGap 2.x	13
Time for action – setting up iOS using PhoneGap 2.x	15
Time for action – setting up Windows Phone using PhoneGap 2.x	16
Getting Started with Android and Eclipse	17
Time for action - installing ADT into Eclipse	18
Getting started with iOS and Xcode	22
Getting started with Windows Phone and Visual Studio	23
Creating a common code base for multiplatform apps	24
Summary	24
Chapter 2: Building and Debugging on Multiple Platforms	25
Development tools	26
Sublime Text	26
IntelliJ IDEA	30
Adobe Brackets	32
Time for action – configuring the cloud service in Bracket	33
Eclipse	35
Native, web, and hybrid apps	35
Working with desktop browsers	36
WebKit debugging (Chrome, Safari, and Opera)	37

Table of Contents

Gecko debugging (Firefox)	43
Internet Explorer 10	46
Mobile debugging workflow	48
Remote debugging	48
Using iWebInspector (OS X only)	49
Time for action – configuring iWebInspector for iOS debugging	49
Debugging with weinre	50
Time for action – configuring Node.js and weinre	50
Wireless debugging with Adobe Edge Inspect	51
Time for action – integrating Edge Inspect and weinre	51
iOS 6 remote debugging	52
Mimicking mobile counterparts	52
Summary	54
Chapter 3: Getting Started with Mobile Applications	55
Mobile-centric HTML/CSS/JavaScript	56
The viewport meta tag	56
Unwanted telephone number linking	57
Autocorrect	57
CSS media queries and mobile properties	57
JavaScript for mobile 101	58
querySelector and querySelectorAll	59
addEventListener	59
Screen orientation	60
Device orientation	60
Shake gestures	60
Media capture API	61
Data URI	61
Performance best-practices	62
Understanding screen size and pixel density	64
Time for action – scaling UI images according to pixel density	64
Writing effective JavaScript	65
Loose coupling	66
Event handling best practice	67
Choosing web app templates	68
HTML5 Mobile Boilerplate	68
Foundation	68
Bootstrap	70
jQuery Mobile	70
Which is the right one?	71
Setting up your project using cordova-cli	71
Time for action – installing cordova-cli using npm	71

Table of Contents

Your first application – "Hello World"	72
Time for action – creating your first cross-platform app	73
Add interactivity to your app	74
Time for action – programmatically opening a modal window using Bootstrap	74
Achieving a native look and feel on iOS	77
Time for action – setting up a native-like CSS for your app	77
Summary	79
Chapter 4: Architecting Your Mobile App	81
Fine-tuning your development environment	81
Speeding up folder access with jump (OS X)	82
Creating a server alias with serve	83
Customizing your shell with iTerm2 (OS X)	84
Time for action – customizing the shell	84
Let LiveReload refresh pages for you (OS X)	86
Time for action – enabling Live Reload	86
Reviewing the JavaScript guidelines	86
Exploring the sample app	87
The navigation flow	88
The app architecture	89
Communication between modules	91
The anatomy of a module	91
Building the app's core	93
Bootstrap loader	95
Time for action – Require.js Bootstrap	95
Mustache templates	96
Time for action – creating a Mustache template	97
Template initialization	98
Time for action – loading and parsing a template	98
The splash screen	98
Time for action – adding a splash screen to your app	99
Summary	100
Chapter 5: Improving the User Interface and Device Interaction	101
Exploring JavaScript compression	102
Google Closure Compiler	102
Time for action – compressing files using the Closure Compiler	104
UglifyJS2	105
Time for action – using UglifyJS with the Closure Compiler	106
Optimization with Require.js	107
Time for action – optimizing JavaScript with Require.js	107
Comparing compression tools	108

Table of Contents

Using template engine compression	109
Time for action – compiling a template using pistachio	110
Handling a retina display user interface	111
Time for action – user interface elements and retina display	112
Creating fluid, multiple app views	113
PhoneGap lifecycle events	114
Time for action – accessing the device API	115
App views	117
Time for action – creating the templates	119
Navigation between views	120
Using hardware-accelerated transitions	123
Alice.js	123
Getting started with the PhoneGap APIs	124
Exploring the Connection API	126
Summary	127
Chapter 6: Using Device Storage and the Contacts API	129
Application data storage	129
Exploring the PhoneGap LocalStorage API	130
Time for action – reading and writing data on the LocalStorage	132
Exploring the PhoneGap SQL storage	136
Database storage with PhoneGap	137
Time for action – populating a local database	138
Database limitations	141
The Contacts API	142
The ContactName object	143
The ContactField object	143
The ContactAddress object	144
The ContactOrganization object	144
The Contact object	145
Filtering contact data	147
Time for action – filtering device contacts	147
Summary	149
Chapter 7: Accessing Device Sensors	151
What are device sensors?	151
Sensors and human-computer interaction	155
Accelerometer	156
Detecting shakes	157
Time for action – detecting shakes in your app	159
Device orientation events	165
Handling orientation with JavaScript	167

Table of Contents

Time for action – handling device orientation with JavaScript	167
Compass	170
Creating a compass	172
Time for action – using the Compass API	173
Summary	175
Chapter 8: Using Location Data with PhoneGap	177
An introduction to Geolocation	177
The PhoneGap Geolocation API	179
Time for action – showing device position with Google Maps	181
Other Geolocation data	185
Time for action – discovering places with Google Places	187
Summary	195
Chapter 9: Manipulating Files	197
Understanding the Files API	197
Reading directories and files	200
Time for Action – listing folders and files recursively	201
Writing and reading a file's data	206
Time for Action – reading and rendering an Image	209
Transferring files	212
Time for Action – downloading and saving a file	214
Summary	217
Chapter 10: Capturing and Manipulating Device Media	219
Camera API or Capture API?	220
Accessing the camera using the Camera API	220
Time for action – accessing the device camera	223
Controlling the camera popover	227
Time for action – controlling the position of the camera roll	229
The Capture API	230
Time for action – manipulating images with a canvas	234
Summary	236
Chapter 11: Working with PhoneGap Plugins	237
Introduction to plugins	237
Getting started with plugins	239
Using plugins with Plugman	240
The anatomy of a plugin	241
Working with plugins	245
The Push Notifications plugin	245
Time for action – using push notifications on Android	247
Summary	251

Table of Contents

Appendix A: Localizing Your App	253
Time for action – rendering localized messages	257
Summary	261
Appendix B: Publishing Your App	263
Publishing on Google Play	264
Publishing on the BlackBerry World	265
Publishing on the Apple App Store	267
Publishing on the Windows Phone Store	269
Summary	270
Appendix C: Pop Quiz Answers	271
Chapter 2, Building and Debugging on Multiple Platforms	271
Chapter 3, Getting Started with Mobile Applications	271
Chapter 4, Architecting Your Mobile App	271
Chapter 5, Improving the User Interface and Device Interaction	272
Chapter 6, Using Device Storage and the Contacts API	272
Chapter 7, Accessing Device Sensors	272
Chapter 8, Using Location Data with PhoneGap	272
Chapter 9, Manipulating Files	273
Chapter 10, Capturing and Manipulating Device Media	273
Chapter 11, Working with PhoneGap Plugins	273
Index	275

Preface

PhoneGap Beginner's Guide will help you break into the world of mobile application development. You will learn how to set up and configure your mobile development environment, implement the most common features of modern mobile apps, and build rich, native-style experiences. Most of the samples deal with real use case scenarios, based upon the code of an open source application available through the most popular app stores.

What this book covers

Chapter 1, Getting Started with PhoneGap, covers how to set up dependencies and mobile SDKs in your development environment.

Chapter 2, Building and Debugging on Multiple Platforms, deals with choosing the development tool that best fits your needs and it will provide an overview of several tools and some debugging techniques.

Chapter 3, Getting Started with Mobile Applications, covers how to improve the performance of a mobile app. You will see how to define the building blocks of a modern hybrid app built using web standards and PhoneGap.

Chapter 4, Architecting Your Mobile App, deals with creating the backbones of your mobile app, focusing on the architecture of the app and on bootstrapping the app.

Chapter 5, Improving the User Interface and Device Interaction, deals with consolidating the app architecture, learning how to implement the navigation mechanics, and how to handle the user interface depending on the network status.

Chapter 6, Using Device Storage and the Contacts API, helps you to understand the offline storage capabilities of PhoneGap and how to interact with the Contacts API.

Chapter 7, Accessing Device Sensors, deals with introducing device sensors, and explaining their power and limitations with regard to the effective use of the APIs provided by the PhoneGap framework.

Chapter 8, Using Location Data with PhoneGap, covers how to get the Geolocation information from a device and how to integrate the external Geolocation service in your app.

Chapter 9, Manipulating Files, covers how to manipulate files on a device. With this knowledge you will be able to store information and read files on a device.

Chapter 10, Capturing and Manipulating Device Media, covers how to access the device camera and all the other capturing tools available in the device.

Chapter 11, Working with PhoneGap Plugins, covers how to extend the PhoneGap capabilities using native code. You will see how PhoneGap's apps architecture allows developers to extend the framework capabilities with the help of custom plugins.

Appendix A, Localizing Your App, covers how to create a localized app using PhoneGap. You will be introduced to the Globalization API, a very powerful tool that allows you to work in conjunction with other JavaScript libraries.

Appendix B, Publishing Your App, covers how you can publish your app on different app stores and discusses the common issues faced.

What you need for this book

A personal computer with an Internet connection and a command line tool to use with the command line utilities distributed with PhoneGap. Windows users should install Cygwin, available at <http://www.cygwin.com/>, to have a Linux look and feel environment; OS X and Linux users just need the default command line tool.

Who this book is for

This book is for web developers who want to start to be productive in the mobile market quickly. In fact by using PhoneGap it's possible to deploy native applications based upon web standards. The book assumes a very small knowledge of HTML/CSS/JavaScript and of mobile platforms, such as Android, BlackBerry, iOS, and Windows Phone, and takes the reader step-by-step into a deep overview of PhoneGap and its API.

Conventions

In this book, you will find several headings appearing frequently.

To give clear instructions of how to complete a procedure or task, we use:

Time for action – heading

- 1.** Action 1
- 2.** Action 2
- 3.** Action 3

Instructions often need some extra explanation so that they make sense, so they are followed with:

What just happened?

This heading explains the working of tasks or instructions that you have just completed.

You will also find some other learning aids in the book, including:

Pop quiz – heading

These are short multiple-choice questions intended to help you test your own understanding.

Have a go hero – heading

These practical challenges give you ideas for experimenting with what you have learned.

You will also find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "The root folder contains the `AndroidManifest.xml` file; it's important that the package name and the activity name defined in the files match the arguments used when launching the `./create` command"

A block of code is set as follows:

```
({
  baseUrl: 'js/',
  paths: {
    mustache: 'libs/mustache',
    alice: 'libs/alice.min',
    text: 'libs/require/plugins/text'
  },
  name: 'main',
  out: 'js/main-built.js'
})
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
function processImages() {  
  
    var pixelRatio = window.devicePixelRatio;  
    if(window.devicePixelRatio > 1) {  
        var matches = document.querySelectorAll("img.highRes");  
        for(var i = 0; i < matches.length; i++) {  
  
            matches[i].width = (matches[i].width / pixelRatio);  
  
        }  
    }  
}
```

Any command-line input or output is written as follows:

```
$ cordova create ~/PhoneGapProjects/PGGettinStarted/ch06/storage  
SampleStorage
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "On Mac the command-line tool is named **Terminal** and you can find it by navigating to **Applications | Utilities | Terminal**."



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important to us so we can develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title through the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website, or added to any list of existing errata, under the Errata section of that title.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

Getting Started with PhoneGap

PhoneGap is an application framework that enables you to build native applications using HTML and JavaScript. Think of PhoneGap as a web view container that is 100% width and 100% height, with a JavaScript programming interface that allows you to access operating system features. The main issue developers used to face with Apache Cordova/PhoneGap was the setup of the development environment and the dependencies with other IDEs. Since version 2.x, however, things have changed dramatically and now the setup is pretty straightforward. This chapter covers setting up your development environment.

In this chapter we will:

- ◆ Get an overview of the history of Apache Cordova/PhoneGap
- ◆ Learn how to configure your development environment configuring all the dependencies
- ◆ Learn how to create a new project using Eclipse, Xcode, and Visual Studio
- ◆ Learn how to create a common code base without using the CLI (Command Line Interface) tool released with the version 2.0

PhoneGap and Apache Cordova

PhoneGap was originally developed by *Nitobi*, a company acquired by Adobe in 2011. After it was acquired, Nitobi donated the PhoneGap code base to the **Apache Software Foundation (ASF)** under the project name **Cordova**, which is the name of the street in Vancouver where Nitobi's offices were located and where the company created the first version of the framework.

One of biggest advantages of moving the code base to the ASF is that big organizations can easily contribute to the project (many companies are not only comfortable with the Apache organization and license, but already have a Contributor License Agreement with Apache); furthermore, the project is now under an open and transparent governance: its community!

PhoneGap is a free and open licensed distribution of Apache Cordova. Picture Cordova to be the engine upon which PhoneGap and its related services (debug, emulate and build services) are built. For existing PhoneGap developers nothing has changed; but for those who are interested in contributing to the project, Apache Cordova is a great chance to join a vibrant open source community.

Adobe continues to play a major role in the project, investing in its ongoing development, and the company decided to keep the PhoneGap name to describe its own distribution of the Cordova project. Other contributors to the Apache Cordova project includes Google, RIM, Microsoft, IBM, Nokia, Intel, and Hewlett-Packard.

The evolution of PhoneGap from 1.4 to 3.x

PhoneGap has evolved very quickly since January 2012, with ten releases in the first nine months of 2012. Projects such as **NodeJS** and **Gnome**, among others, use odd MINOR version numbers to denote development releases. It's a very good practice that tells developers whether a release has major or minor improvements.

Apache Cordova follows the guidelines defined in the **Semantic Versioning** specification (more information is available at <http://semver.org/>), which seeks to address one of a developer's nightmares: dependency hell. The Semantic Versioning specification codifies the long-standing de facto version schema of X.Y.Z where X denotes MAJOR changes, Y represents MINOR updates, and Z is a PATCH to the minor update stream.

Since the 1.4 release, the project has been known as Apache Cordova. This release is generally considered the first stable release of the framework, with a fairly complete and up-to-date documentation. The 1.5 release fixed a long list of bugs, but the initial reaction of the community was not very favorable because the documentation was outdated and some changes to the main files caused broken build issues to apps developed with earlier releases. The 1.6 release brought some improvements to the plugin architecture, the Camera and Compass APIs, and the project template files. As is often the case with a maturing community, the release was not perfect but there was a significant improvement in the overall quality compared to the previous release. The 1.7 and 1.8 releases were bug fixes and added support for Bada 2.0. The community reaction was positive also because of the speed of the releases. The 1.9 release addressed even more bug fixes and added support for the new features of the iOS and Android platforms.

Apache Cordova 2.x added the following features and support:

- ◆ The definition of a unique JavaScript file to use across all platforms thanks to the unification of the JavaScript layer of the Cordova application framework
- ◆ The introduction of a command line tool (CLI) through which common operations, such as project creation, debug, and emulation, can be performed in a standard way (Android, iOS, and BlackBerry)
- ◆ The capability to embed PhoneGap applications into larger native iOS and Android applications using Cordova WebView
- ◆ Support for the Windows phone platform
- ◆ The porting of the **Web Inspector Remote (Weinre)** to nodejs and the introduction of a node module that facilitates installation using **Node package manager (npm)**
- ◆ An improved plugin documentation and support thanks to a **plugman** Node.js application able to install compliant plugins
- ◆ Several improvements to the process of creating iOS apps
- ◆ The introduction of the long-awaited `InAppBrowser` API (formerly called `ChildBrowser`)
- ◆ The standardization of the commands available for each platform (i.e., `build`, `run`, and so on)
- ◆ Full support for Windows Phone 8

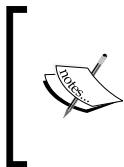
With the 2.x release, Apache Cordova and PhoneGap have become a mature, stable, and powerful tool in the mobile developer's toolkit. The upcoming PhoneGap 3.x will no longer support BlackBerry 7 and will add support for Ubuntu and Firefox OS. Since PhoneGap 3.x the APIs work like plugins and can be installed and uninstalled using the updated cordova cli utility, enabling the PhoneGap core to be slim and better performing. PhoneGap 3.x also comes with a better set of command-line tools; for instance, the **phonegap npm** module allows you to build your app using the command line and the online PhoneGap Build service. For a good overview of the major changes in PhoneGap 3.x refer to the blog post at <http://phonegap.com/blog/2013/06/20/coming-soon-phonegap30/>.

A note about operating systems

We touched on this in the preface but it's worth emphasizing again: PhoneGap plays by the rules. If a vendor releases an SDK for a single operating system only, then you will have to use that OS to build and deploy your applications.

In detail, for each PhoneGap platform:

- ◆ You can develop **Android** apps on any of the major desktop operating systems—Windows, Mac OS X, or Linux
- ◆ You can develop **Symbian Web Runtime** apps on any OS but you can only run the simulator from Windows
- ◆ You can develop apps for **BlackBerry** on any of the major desktop operating systems—the SDK can be installed on Windows or Mac OS X (to run the emulator you need to install the virtual machine distributed with the SDK)
- ◆ The **Windows Phone 7** SDK runs on Windows Vista SP2 and on Windows 7
- ◆ The **iOS SDK** requires OS X 10.7 or later (and, according to the OS X EULA, a Mac computer as well)



You can emulate apps in the desktop browser with **Ripple** (a Chrome extension that is currently incubated in the Apache Software Foundation <http://incubator.apache.org/projects/ripple.html>) or with the online emulation service available at <http://emulate.phonegap.com>.

Practically speaking, your best bet for mobile development is to get a Mac and install Windows on a separate partition that you can boot into, or run it in a virtual environment using **Parallels** or **VMWare Fusion**. According to Apple's legal terms, you cannot run Mac OS X on non-Apple hardware; if you stick with a Windows PC, you will be able to build for every platform except iOS (if you want to ignore Apple's legal terms refer to this tutorial <http://lifehacker.com/5938332/how-to-run-mac-os-x-on-any-windows-pc-using-virtualbox> and run OSX on a Windows PC).

Anyway, with the new CLI utilities it is getting to be pretty straightforward to build an app for all major mobile platforms. Mobile developers are well aware of the problems involved in building cross-platform apps; not surprisingly, the <http://build.phonegap.com> service is starting to become pretty popular due to the fact that it lets the developer use his/her favorite operating system. After registering with this service, it's possible to build a cross-platform app starting from a common code base. You can upload the code base or pull it from a GitHub repository. When compiling the app it's possible to enable or disable the debug and to activate the **hydration** services. Hydration is able to push updates directly to the application installed on a device. This is accomplished by compiling a native binary that acts as a container for the mobile application. Once a new build is uploaded the end user (for example, tester) will be notified upon restart of the app. At the end of this book I have included a section dealing with the distribution process for mobile applications.

A note about the command-line tool

Across this book you will widely use command-line tool, if you are on a Mac or on a Linux machine the commands you can type in the tool are almost the same. On Mac the command-line tool is named **Terminal** and you can find it by navigating to **Applications | Utilities | Terminal**. On Linux you refer again to the command-line tool with the word "Terminal" but its location varies accordingly to the Linux distribution you are using. In Gnome, the classic desktop environment for Ubuntu 11.04, you can find the **Terminal** by navigating to **Applications | Accessories | Terminal**. In Xfce you can find the **Terminal** by navigating to **Applications | System | Terminal**.

On a Windows machine the command-line tool is named MS-DOS prompt and can be opened by typing `cmd` and pressing *Enter*, in the start menu. The commands syntax is different from the Mac or Linux one, the most relevant differences are summarized in the following table:

Command	Terminal	MS-DOS prompt
Copy Files	<code>cp</code>	<code>copy</code>
Edit text files	<code>vi</code>	<code>edit</code>
Compare files	<code>diff</code>	<code>fc</code>
Browse files	<code>ls</code>	<code>dir</code>
Clear screen	<code>clear</code>	<code>cls</code>
Create a symbolic link	<code>ln</code>	<code>mklink</code>

Installing PhoneGap

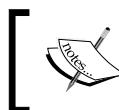
The installation process of PhoneGap has been confusing for a long time because there were a lot of dependencies. These dependencies were due to the fact that in order to compile an app for different platforms, you not only have to have the platform-specific SDKs but also the platform-specific tool: in order to build for Android Eclipse, IntelliJ, or Android Studio is needed; to build for iOS Xcode is needed, and so on.

With Apache Cordova 2.x and 3 both the installation process and setting up your development environment have become much easier. You can now use the build services provided by Adobe, plus Cordova 2.x and 3 ships with a set of command-line tools that make it easier to develop cross-platform applications. Installing PhoneGap is now as easy as downloading the latest distribution from the website and unzip the package—you'll get the following folders:

```
| -doc
| -lib
| ---android
```

```
| ---bada  
| ---badaWac  
| ---blackberry  
| ---ios  
| ---symbian  
| ---webos  
| ---windows-phone
```

All the necessary tools for each platform are bundled in the archive file and for OSX users using PhoneGap 2.2 or previous it's mandatory to mount the DMG file located in the lib/ios folder and run the Cordova package available in the DMG file.



The download folder of PhoneGap varies accordingly to your operating system, since now I will refer to it as PHONEGAP_ROOT in all the command-line commands.



Installing dependencies

In order to be ready to build a PhoneGap app it's mandatory to download and install the latest SDKs for each target platform of the app from the respective official websites:

- ◆ **Android:** <http://developer.android.com/sdk/index.html>.
- ◆ **BlackBerry 10:** <https://developer.blackberry.com/html5/downloads/#blackberry10>.
- ◆ **Firefox OS:** <https://addons.mozilla.org/en-US/developers/docs/sdk/Firefox-22/dev-guide/tutorials/mobile.html> (experimental and available only with PhoneGap 3.x).
- ◆ **iOS:** <https://developer.apple.com/devcenter/ios/index.action>.
- ◆ **Windows Phone:** [http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402523\(v=vs.92\).aspx](http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402523(v=vs.92).aspx).

Once the SDKs are installed it's recommended to install also the client for GitHub. GitHub is a social coding platform where you can find most of the cool open source projects (such as Apache Cordova), including access to the latest patches, builds, and sources.

For OSX users it is important to install also ios-sim. The `ios-sim` tool is a command-line utility that launches an iOS app on the simulator. To install the tool, download it from GitHub, expand it, and launch a command line tool as follows:

```
$ curl -L https://github.com/phonegap/ios-sim/zipball/1.3 -o ios-sim-1.3.zip  
$ unzip ios-sim-1.3.zip  
$ rake install prefix=/usr/local/
```

Depending on your privileges you may have to run the `rake` command as an administrator (i.e., adding `sudo` in front of the `rake` command).

Setting up your development environment

This section provides a detailed guide to set up the Android, iOS, and Windows Phone development environment, in order to set up a development environment based upon a simple text editor it suffices to run few commands using the command-line tool.

Time for action – setting up Android using PhoneGap 2.x

Get ready to set up the Android development environment and to create a PhoneGap app using Android as target platform. Follow these steps:

1. Open the folder in which you unpacked the PhoneGap distribution, launch a command-line tool (Terminal on OS X or DOS Prompt on Windows) and change directories to the bin directory for Android:

```
$ cd PHONEGAP_ROOT/lib/android/bin
```

2. In order to run the Android SKD tools, you must add them to the path where the Android tools reside. Use the command-line tool and mount the path (the paths vary depending on the location of the Android SDK):

```
$ export PATH=$PATH:~/android-sdks/tools/  
$ export PATH=$PATH:~/android-sdks/platform-tools/
```

3. In order to create a PhoneGap project, all you have to do is run the tool `./create` from the `bin` directory of android:

```
$ ./create ~/PhoneGapProjects/PGGettinStarted/ch01/android com.  
gnstudio.samples.cordova.hello HelloPG
```

The tool actually needs three parameters: the path of the project files, the package of the project, and the name of the project. The tool creates an Android project including the folders and the files needed to debug, emulate, and build the PhoneGap app:

```
| -assets  
| ---www  
| -bin  
| ---res  
| -cordova  
| -gen  
| -libs  
| -res  
| -src
```

4. The root folder contains the `AndroidManifest.xml` file; it's important that the package name and the activity name defined in the files match the arguments used when launching the `./create` command:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/  
    android" android:windowSoftInputMode="adjustPan"      package="com.  
    gnstudio.samples.cordova.hello" android:versionName="1.1"  
    android:versionCode="5">  
    <activity android:name="HelloPG" android:label="@string/app_name"  
        android:configChanges="orientation|keyboardHidden">
```



The `AndroidManifest.xml` file `windowSoftInputMode` attribute value is `adjustPan` because the default behavior of Android is to resize the web view when the soft keyboard is displayed.

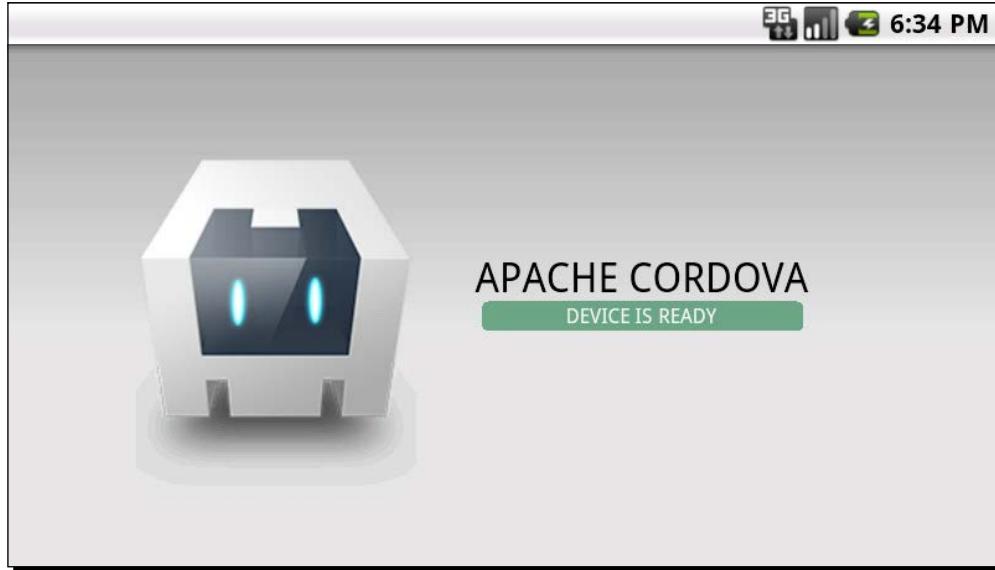
5. In the `www` folder you'll find the HTML/JS/CSS files needed to run the sample PhoneGap app bundled with the binary of the distribution. In order to run the application on the emulator, it's enough to run the app. Open a command-line tool and point to the `cordova` folder, define the path to the Android SDK tools, and launch the `./emulate` command-line tool:

```
$ cd ~/the/path/to/your/source/code/ch01/android/cordova  
$ export PATH=$PATH:~/android-sdks/tools/  
$ export PATH=$PATH:~/android-sdks/platform-tools/  
$ ./emulate
```

The tool will check whether some virtual devices are already defined and prompt the user to define one if not. If there is more than one device already defined the tool will ask which one to use. When the selected device is loaded into the emulator you can debug the application; when launching the `./debug` tool the app is installed and can be launched from the control panel of the emulator.

What just happened?

You created a PhoneGap project and emulated the HelloPG app in one of the testing devices configured within your Android SDK.



In order to correctly emulate the app on Windows you have to install the latest Android JDK, the Apache Ant binary, and the Android SDK. After all of those are installed add the path `C:\Program Files\Java\jdk1.7.0_09\bin;C:\Users\YOUR_USER\Documents\apache-ant-1.8.4\bin;C:\Program Files (x86)\Android\android-sdk\tools` to your path system variable. You can access the path system variable by navigating to **Control Panel | System | Advanced System Settings | Advanced | Environment Variables | System Variables**.

Time for action – setting up iOS using PhoneGap 2.x

The steps to setup the iOS development environment and to create a PhoneGap with iOS as target are very similar to the Android setup. Follow these steps:

1. Navigate to the bin/ios folder of the current PhoneGap installation and launch the `./create` tool:

```
$ cd phonegap-phonegap-2.0.0/lib/android/bin  
$ ./create ~/PhoneGapProjects/PGGettinStarted/ch01/ios com.  
gnstudio.  
samples.cordova.hello HelloPG
```

The syntax is the same as the `./create` command used for Android with the exception of the target path, which is a subfolder of the iOS one. The result is a folder structure familiar to Objective-C developers:

```
| -HelloPG  
| -HelloPG.xcodeproj  
| -build  
| -cordova  
| -www
```

2. Run the `./emulate` command from the `cordova` folder, the result is that the iOS emulator is opened without the need to open XCode with the PhoneGap app running:

```
$ cd ~/PhoneGapProjects/PGGettingStarted/ch01/ios  
$ ./emulate
```

What just happened?

You created a PhoneGap project using iOS as the target platform and emulated the app using the iOS Emulator.



If you are on Windows and you want to be able to use the Linux syntax used throughout this book you can download and install **Cygwin** from <http://www.cygwin.com/>.

Time for action – setting up Windows Phone using PhoneGap 2.x

If you want to start building a Windows Phone app using PhoneGap you have to work on a Windows machine (physical or virtual). Once you have access to the machine follow these steps:

1. Download the SDK available at <http://dev.windowsphone.com/en-us/downloadsdk> or install a 90 days trial of Visual Studio available at <http://www.microsoft.com/visualstudio/eng/downloads>.
2. Go to the `PHONEGAP_ROOT/lib/windows-phone-8/templates/standalone` folder and open the file `CordovaSolution.sln` using Visual Studio.
3. Export the template in order to use it when creating new PhoneGap projects, navigate to **File | Export Template**.

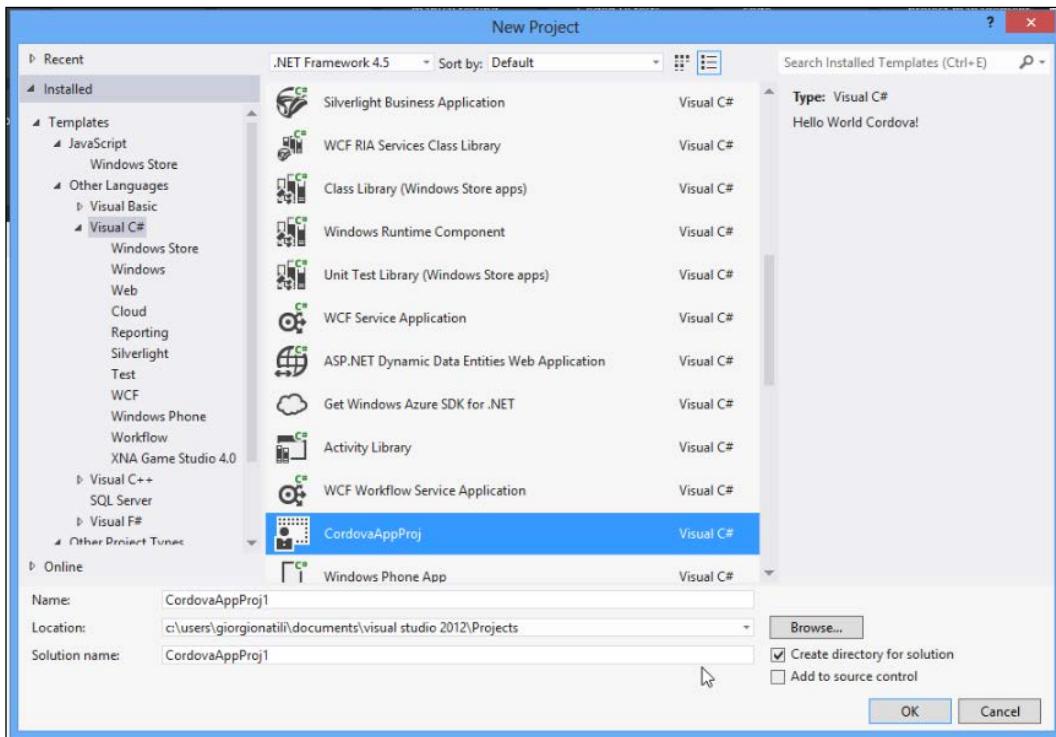


Remember to check the option **Automatically Import the Template into Visual Studio** in order to have the template available in the Visual Studio project wizard.

4. Restart Visual Studio.

What just happened?

You installed a new template in Visual Studio you will find the template available in the C# branch. The template includes all the PhoneGap sources so that you can emulate and build your app easily.



Getting Started with Android and Eclipse

One of the most popular IDEs for Java is Eclipse, you can download the latest version from the Eclipse website <http://www.eclipse.org/downloads/>. In order to run a sample application based on Apache Cordova/PhoneGap you need to install Eclipse, download the Android SDK, and add the **Android Development Tools (ADT)** plugin for Eclipse to your Eclipse install. ADT extends the capabilities of Eclipse to let you quickly set up new Android projects, create an application UI, add packages based on the Android Framework API, debug applications using the Android SDK tools, and even export signed (or unsigned) APK files in order to distribute the application.

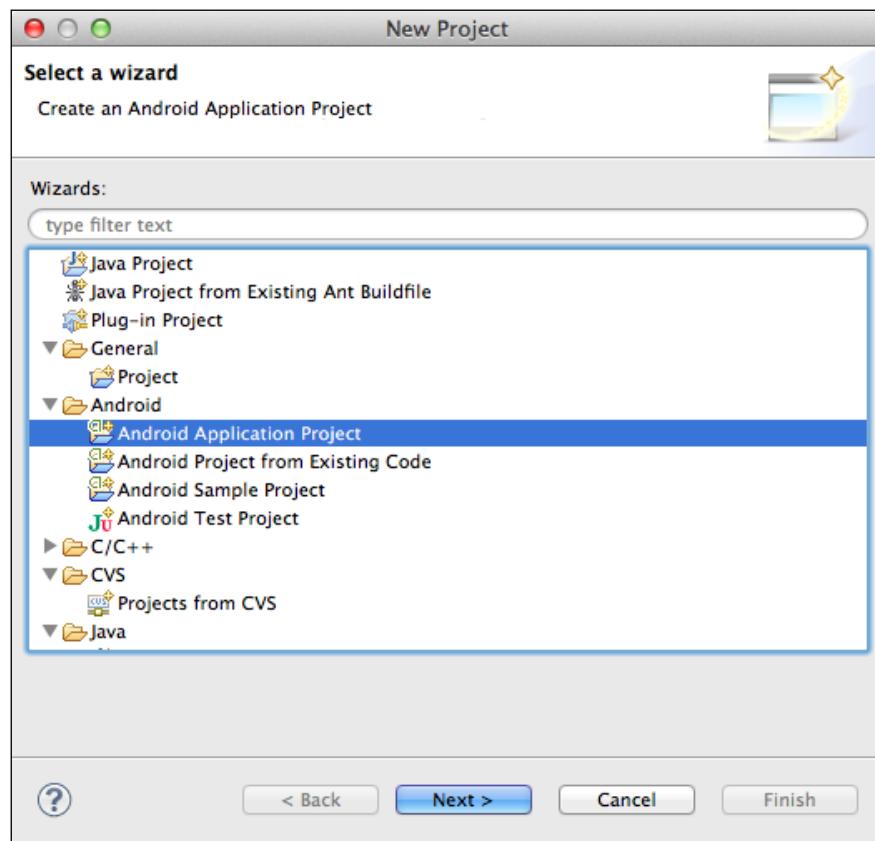
Time for action - installing ADT into Eclipse

In order to install ADT into Eclipse it's enough to perform the following steps:

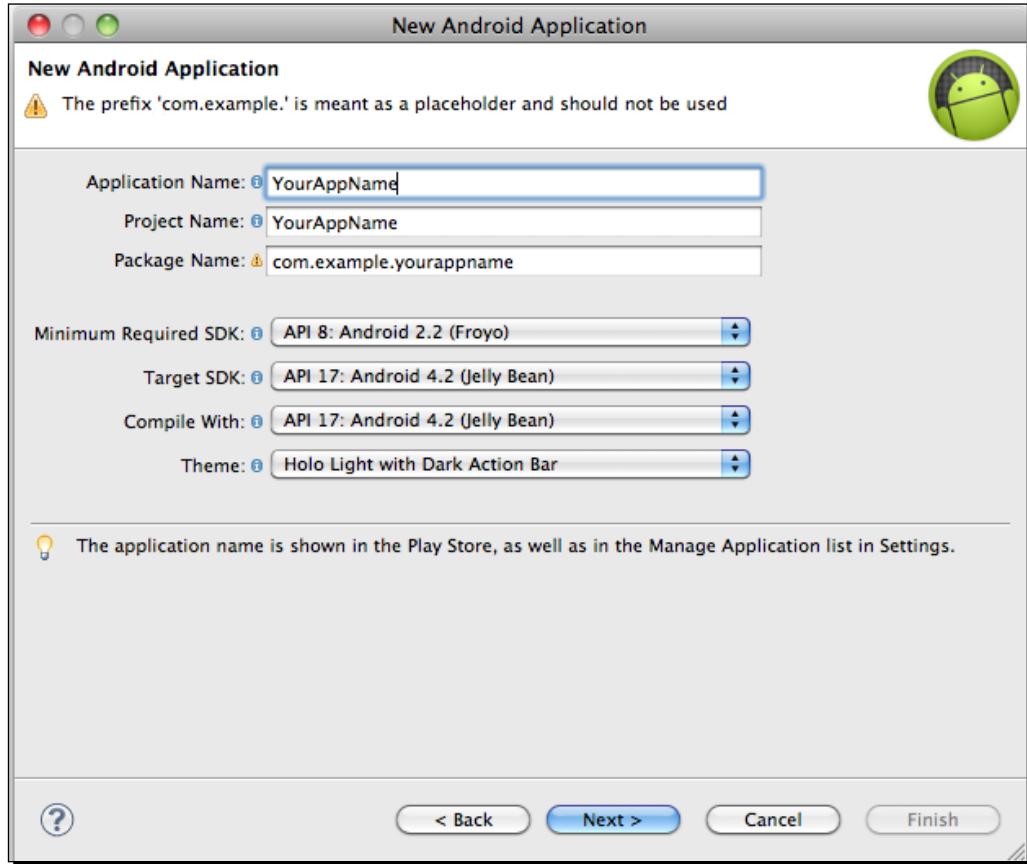
1. Start Eclipse then navigate to **Help | Install New Software**.
2. Click on **Add** in the top-right corner.
3. In the **Add Repository** dialog that appears, enter **ADT Plugin** in the **Name** field and the following URL for the **Location** field: <https://dl-ssl.google.com/android/eclipse/>.
4. When the installation completes, restart Eclipse.

What just happened?

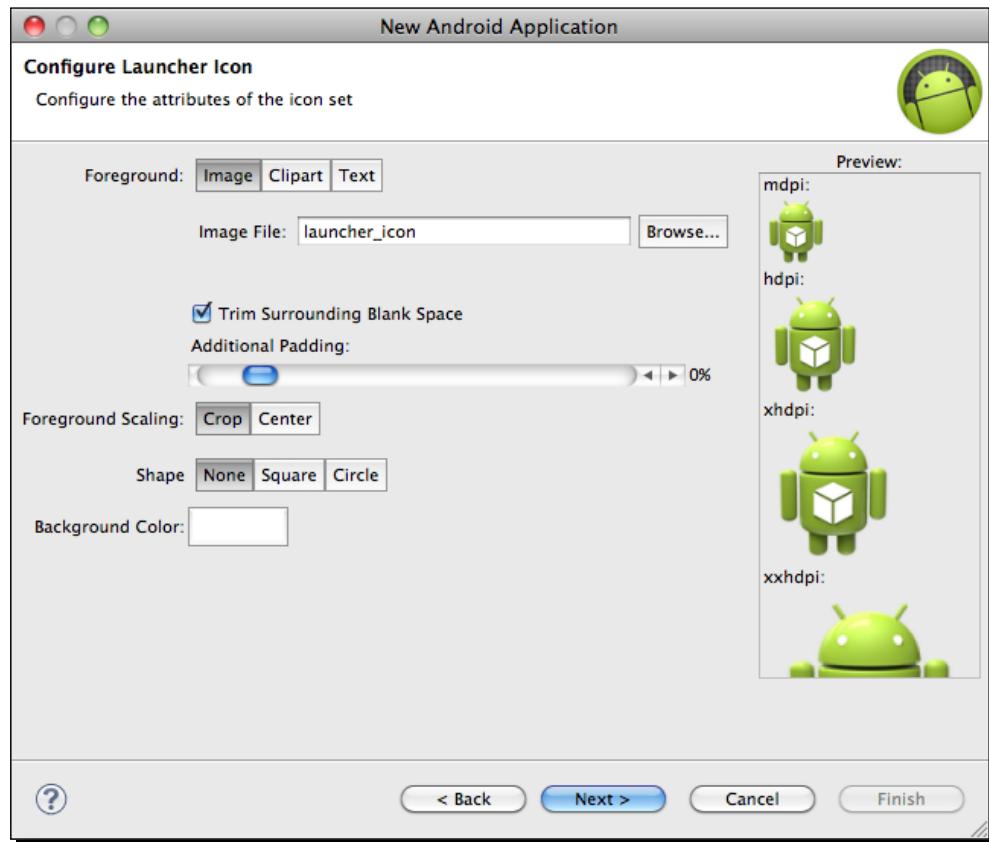
Once the Eclipse installation is properly configured it's possible to create a new project using the appropriate wizard.



Next, define the application name, the packaging, the Android SDK to use in order to build the app and the minimum SDK.



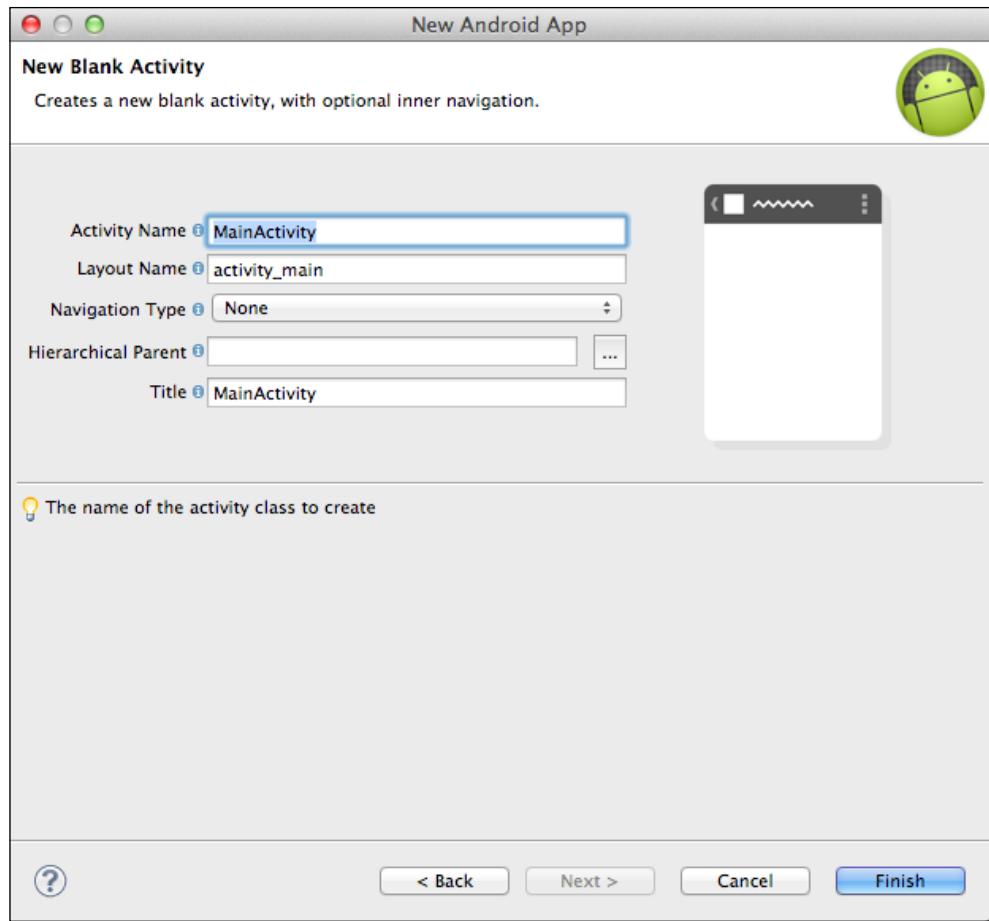
The wizard lets you define the application icons, i.e., the icons end users will see in the mobile device when using the app.



An important step when creating an Android project based upon PhoneGap is to select the option to start with a blank activity and configure it properly. An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the `Activity` class takes care of creating a window in which it's possible to place the UI of the application. Make sure the activity doesn't inherit from anything—i.e., leave the **Hierarchical Parent** field blank.



Always edit the **Title** field in order to show the appropriate name of your app when installing, running, and uninstalling it.



It's very important to have PhoneGap in the Eclipse workspace or be aware of the path to reach the `PHONEGAP_ROOT` folder in order to use the files required to run the app.

In the root directory of the project create two new directories: `/libs` and `assets/www`.

Place a copy of the `cordova-2.x.y.jar` and `cordova-2.x.y.js` into these directories, the files are located in the `PHONEGAP_ROOT/lib/android` folder. Copy the `xml` folder inside the project `res` folder.

Verify that `cordova-2.x.y.jar` is listed in the Build Path of the project. Right-click on the `/libs` folder and go to **Build Paths | Configure Build Path**. Then, in the **Libraries** tab, add `cordova-2.x.y.jar` to the project.

In order to run and debug the application, the Main file of the project should look like the following code snippet:

```
package com.gnstudio.samples.cordova.hello;  
import android.os.Bundle;  
import org.apache.cordova.*;  
public class MainActivity extends DroidGap{  
    @Override  
    public void onCreate(Bundle savedInstanceState){  
        super.onCreate(savedInstanceState);  
        super.loadUrl(Config.getStartUrl());  
    }  
}
```

The Main file by default contains the `override` of the default Android menu function. It can be safely removed because it's not needed at all.

 In order to correctly load a file into the WebView, Internet permission is required. In order to enable it add the following tag to the `AndroidManifest.xml` file `<uses-permission android:name="android.permission.INTERNET" />`.

Getting started with iOS and Xcode

In order to start to develop apps for iOS devices it's mandatory to have a Mac and to download the iOS SDK and Xcode, both available on the **Apple Developer Center** <http://developer.apple.com>. Complete the following steps and if you are using PhoneGap 2.0 run the installer `Cordova-2.0.0.pkg` available after mounting the `Cordova-2.0.0 DMG` file locate into the folder `/phonegap-phonegap-2.0.0/lib/ios`:

- ◆ Install Xcode from the App Store.
- ◆ Install the Xcode command-line tools (**Xcode Preferences | Downloads | Components | Command Line Tools | Install**).

Once everything is installed navigate to the `bin/ios` folder of the current PhoneGap installation and launch the `./create` tool:

```
$ cd PHONEGAP_ROOT/lib/android/bin  
$ ./create ~/the/path/to/your/source/code/ch01/ios com.gnstudio.samples.  
cordova.hello HelloPG
```

In the target folder you'll find the file `HelloPG.xcodeproj`; when you open the file in Xcode, your project is already set up and ready to be debugged and deployed.



If you are using a PhoneGap version greater than the 2.0.0 there is no need to mount the mentioned DMG file in the PHONEGAP_ROOT folder. In order to create a project, is enough to go to the PHONEGAP_ROOT/lib/ios/bin folder and run the ./create tool or use the cordova command line utility available as a npm module.

In order to debug the app change the **Target** in the **Scheme** drop-down menu on the toolbar to **HelloPG** (or the current project name) and change the Active SDK in the **Scheme** drop-down menu on the toolbar to **iOS [version] Simulator**.



To deploy the app on a physical device: open the file `HelloPG-Info.plist`; under the Resources group, change `BundleIdentifier` to the identifier provided by Apple; change the **Target** in the **Scheme** drop-down menu on the toolbar to **HelloPG** (or the current project name); and, with the device connected through USB, click on the **Run** button in the project window's toolbar.



If you are searching a tool for Objective-C with advanced refactoring features, a better code completion, a great support for Unit tests and powerful code inspection tool you have to consider to buy **AppCode** from **JetBrains**, more information is available at <http://www.jetbrains.com/objc/>.

Getting started with Windows Phone and Visual Studio

In order to start to work with Visual Studio, it is enough to use the template we previously installed. If you want to debug the app on a real device anyway you have to create a developer account at <https://dev.windowsphone.com/en-us/account> in order to unlock for debugging a real device.

Creating a common code base for multiplatform apps

A good habit in order to have a common code base when developing a mobile app with multiple target platforms is to create a virtual folder in the source code folder of each mobile project. This ensures that all the common code (i.e., HTML/CSS/JS files) can be shared between several projects. In order to do this move to the source folder of the Android project and of the iOS project. Using your command-line tool, create a new virtual folder:

```
$ ln -s ~/the/path/to/your/source/code/common/src ~/the/path/to/your/source/code//ch01/android/assets/www
```

```
$ ln -s ~/the/path/to/your/source/code/common/src ~/the/path/to/your/source/code/ch01/ios/www
```

Both the virtual folder link to the common base located into `~/PhoneGapProjects/PGGettingStarted/common/src`, the same can be done for image and media files.



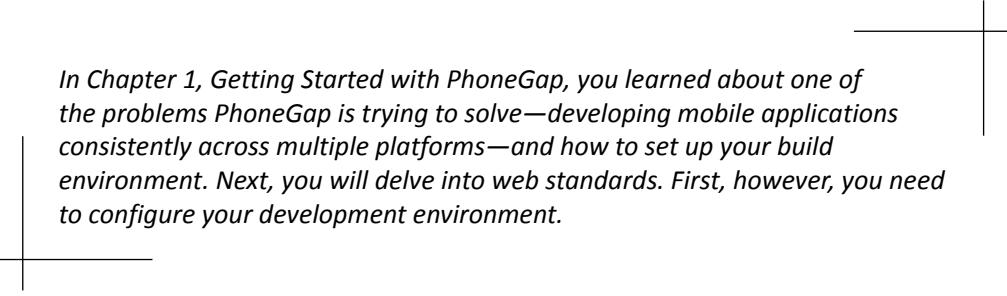
Please remember that this technique could be helpful with the older version of PhoneGap; you will learn shortly how to manage a common code base easily with the `cordova-cli` tool.

Summary

In this chapter you learned how to set up your development environment using the CLI tools included into Apache Cordova. The next chapter will help you choose a development environment and show you how to debug your first app on multiple platforms.

2

Building and Debugging on Multiple Platforms



In Chapter 1, Getting Started with PhoneGap, you learned about one of the problems PhoneGap is trying to solve—developing mobile applications consistently across multiple platforms—and how to set up your build environment. Next, you will delve into web standards. First, however, you need to configure your development environment.

In this chapter you will:

- ◆ Take a look at different development environments and their relative workflow
- ◆ Get an overview of the browser's debugging tools
- ◆ Review the debug workflow with mobile in mind
- ◆ Discover how to debug apps on devices
- ◆ Learn how to use Ripple to emulate a target device

Development tools

As mentioned in *Chapter 1, Getting Started with PhoneGap*, you can work either with Eclipse, Xcode or Visual Studio. Each tool has its pros and cons. However, if you would like to create some custom plugins for your apps, you may want to consider using all of them.

One of the main advantages of Apache Cordova and its PhoneGap distribution is that you can build mobile applications in the browser using web standards and well-known technologies. This is the reason why you can use either a text editor or an **integrated development environment (IDE)** when working on PhoneGap apps; most of the time you will work on HTML, CSS, and JavaScript files. The difference between a text editor and an IDE is that an IDE understands the programming language, whereas a text editor understands text. Depending on the task at hand, some developers prefer to work with an editor rather than an IDE.

Let's take a look at the main features of some different tools: Sublime Text, IntelliJ IDEA, Adobe Brackets, and Eclipse. All of them run on Mac OS X, Windows, and Linux with the exception of Adobe Brackets, which does not run on Linux (actually there is not an official build for Linux, but there is a porting very close to completion and soon to be officially released <https://github.com/adobe/brackets/wiki/Linux-Version>).

Sublime Text

Sublime Text is the tool to use if you want a fast, feature-rich, and highly customizable editing environment with a clean user interface. One of its most compelling features is Go Anywhere: Press *Ctrl + P* (Windows) or *Command + P* (OS X) and start typing part of the name of a file to open it in the editor; the speed and the accuracy of the navigation between files is impressive. You can also use the following shortcuts in conjunction with the Go Anywhere feature:

- ◆ Press the *@* key to scroll through an auto suggest list of HTML IDs, CSS selectors, JavaScript functions, and so on; type the initial letters of the element you are looking for to narrow the list of suggestions

- ◆ Press : to scroll through and navigate to a specific line of code.

```

34  @
35  b CSS: body
36  c CSS: #info
37  m CSS: #info > h4
38  w CSS: #stage.theme
39  p CSS: #stage.theme > dl
40  } CSS: #stage.theme > dl > dt
41  CSS: #stage.theme > dl > dd
42  #in CSS: #stage.theme > h1, #stage.theme > h2, #stage.theme > p
43  f CSS: #stage.theme > h1
44  m CSS: #stage.theme > h2
45  } CSS: #stage.theme a.btn
46  CSS: #stage.theme a.btn.large
47  #stage.theme{
48  padding-top:3px;
49  }

```

Using Sublime Text you can also easily change multiple occurrences of the same word in a file (also known as **refactoring**); just press *Ctrl + D* (Windows) or *Command + D* (OS X) multiple times in order to select multiple occurrences of the same word and then start typing.

Another great feature of Sublime Text is the column layout, which you can activate by navigating to **View | Layout**. Splitting the editor in multiple columns allows you to edit multiple files simultaneously. When used in conjunction with the distraction free mode (**View | Enter Distraction Free Mode**), this feature can be a great time saver when coding.

The beauty of Sublime Text is that everything is highly customizable, both on the editor level as well as the user level (through preference files), which is important because a text editor is a key tool in a developer's workflow. There are a lot of plugins for Sublime Text. In order to access them you can install Package Control (http://wbond.net/sublime_packages/package_control); press *Ctrl + `* (Windows) or *control + `* (OS X) to open the editor console in order to run the installation script of Package Control and restart Sublime Text. After you restart the editor you can access the installation commands by pressing *Ctrl + Shift + P* (Windows) or *command + Shift + P* (OS X) and type *install* to browse and install plugins.

Useful plugins you want to install include:

- ◆ **SideBarEnhancements** for a richer feature set such as copy, paste, find in project, reveal, and open in the browser
- ◆ **SublimeLinter** for visual feedback of potential errors in HTML, CSS, JS, and PHP files (among others)
- ◆ **CSS Prefixer** get automatically the specific browser vendor prefixes in your CSS

- ◆ **CodeIntel** for code hinting and navigating between files using *Ctrl + click*
- ◆ **Emmet** to add helpers such as Lorem Ipsum text generation, CSS code completion, and more to the text editor (note that this plugin is a work in progress; for details refer to the GitHub repository available at <https://github.com/sergeche/emmet-sublime/blob/master/README.md>)

The most common and useful shortcuts to use on OS X with Sublime Text are summarized in the following table; for a complete list of the shortcuts available for each platform, refer to the tutorial available at <http://www.wdtutorials.com/2013/06/23/sublime-text-2-keyboard-shortcuts-cheat-sheet-win-os-x-and-linux>.

Shortcut	Result
<i>Control + Command + P</i>	Switch between different projects
<i>Command + /</i>	Comment/uncomment code
<i>Command +]</i>	Indent code
<i>Command + [</i>	Unindent code
<i>Control + Command + up</i>	Move the selected line up
<i>Control + Command + down</i>	Move the selected line down
<i>Command + number</i>	Switch to an open tab (<i>command + 1</i> goes to the first tab, <i>command + 2</i> to the second tab, and so on)
<i>Shift + Command + F</i>	Search through the whole project
<i>Shift + Control + K</i>	Delete selected code or line

The strengths of Sublime Text are that it helps you save time coding and makes it easy to switch between projects and files. However, you also have to invest some time to install all the plugins and the project configuration can be a little tricky for a novice.

Just to make an example let's open Sublime and create a new project by navigating to **Project | Save Project As**. The editor creates a `YOUR_PROJECT_NAME.sublime-project` file, by opening it you can configure your project.

Imagine working on a PhoneGap project and being able to concentrate only on the files that are more relevant to you. Most of the time you have to work on the files in the `assets/www` folder or in the `www` folder depending on whether you are working on an Android or iOS project.

If you want to see only the folders that are relevant to you, it's enough to open the Sublime Text project file and specify which paths you want to exclude from the side bar view.

```
{  
  "folders":  
  [  
  ]
```

```
{  
    "path": " ~/PhoneGapProjects/PGGettinStarted/ch02",  
    "folder_exclude_patterns":  
        ["android/bin", "android/cordova",  
         "gen",  
         "res",  
         "ios/cordova"]  
}  
}  
]  
}
```

Downloading the example code



You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

As a result, in the side bar you will see only the folders you are interested in.



[ You can download Sublime Text from the product website <http://www.sublimetext.com/> and you can update the Sublime Text blog, <http://www.sublimetext.com/blog/>.]

IntelliJ IDEA

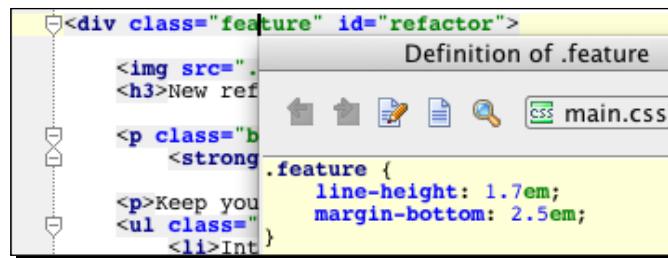
IntelliJ IDEA is the tool to use if you want to be able to switch quickly from web development to native development. It's a very powerful IDE, yet it doesn't require a lot of resources to run smoothly.

IntelliJ IDEA is a commercial development tool that competes with the free Eclipse and NetBeans editors. The key objective of IntelliJ IDEA is developer productivity, and to a large extent it achieves that goal. It's an IDE designed to automate parts of the coding process, to support a large number of different frameworks, tools, and targets, and to work with multiple languages.

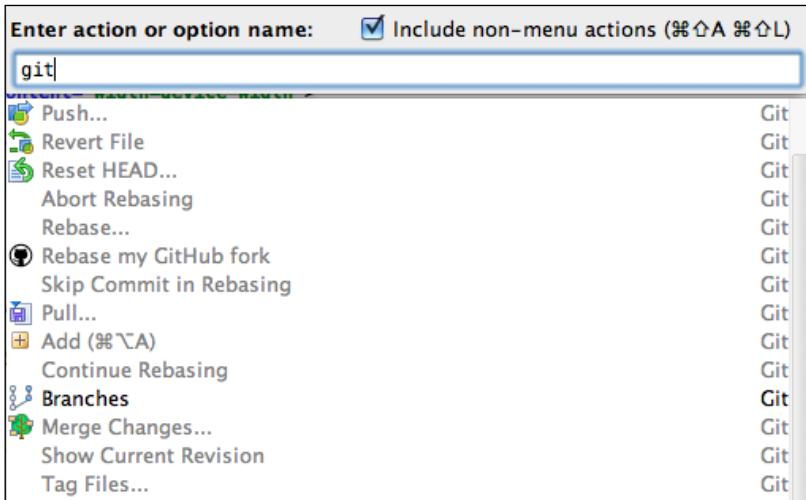
Although IntelliJ IDEA strives to be a clean, uncluttered IDE, the multitude of features has undermined that to some extent. Like any IDE, it has a learning curve, with many keyboard shortcuts and a distinctive logic that may elude beginners.

IntelliJ IDEA code completion helps you recognize method, property, and other names, and provides an easy way to work with several languages. To access code completion press *Ctrl + Space bar*, according to the current programming language, you get different goodies:

- ◆ When using JavaScript you get code completion for the most popular libraries such as jQuery, Ext.js, MooTools, Prototype, and YUI
- ◆ When using HTML you are able to navigate from markup element IDs to the CSS file, define in an external file a CSS class, or just preview a CSS definition by pressing the *Ctrl + Shift + I* or *command + Shift + I* keyboard combination
- ◆ When writing a SQL query you get suggestions based on your defined data source
- ◆ When writing Java you can use features such as method overriding and interface implementation

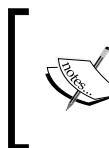


IntelliJ IDEA supports intelligent language-specific code assistance features, which is perfect for PhoneGap projects because you can switch from any language by getting the same kind of support. IntelliJ IDEA is also well integrated with the most common source controls, including the popular Git and GitHub. To gain quick access to a feature, simply press *Ctrl + Shift + A* (Windows) or *Command + Shift + A* (OS X) and you can start typing the name of the feature you want to gain quick access to it.



IntelliJ IDEA also supports powerful code refactoring features; you can change occurrences in a project, extract variables, define missing properties or variables at the click of a button, and more. (**Code refactoring** is a technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. If you want to know more about refactoring there is an excellent book written by *Martin Fowler* dedicated to this topic available at <http://www.amazon.com/Refactoring-Improving-Design-Existing-Code/dp/0201485672>.)

The main advantage of using IntelliJ IDEA is that you can use one tool to write HTML, CSS, JavaScript, Java, and so on. The only missing feature of IntelliJ IDEA is that it is not an editor for Objective C. If you want to extend your PhoneGap app for iOS you have to use Xcode or seriously consider AppCode from JetBrains, which offers the same environment and the same features of IntelliJ IDEA but is oriented to the Apple platform.



To download IntelliJ IDEA refer to the download page <http://www.jetbrains.com/idea/download/>, if you want to be updated about the new upcoming features follow the IntelliJ IDEA blog <http://blogs.jetbrains.com/idea/>.

Recently Google released the Android Studio, a new Android development environment based on IntelliJ IDEA that, similar to Eclipse with the ADT Plugin, provides integrated Android Developer Tools for development and debugging. You can find more information and the download link from <http://developer.android.com/sdk/installing/studio.html>.

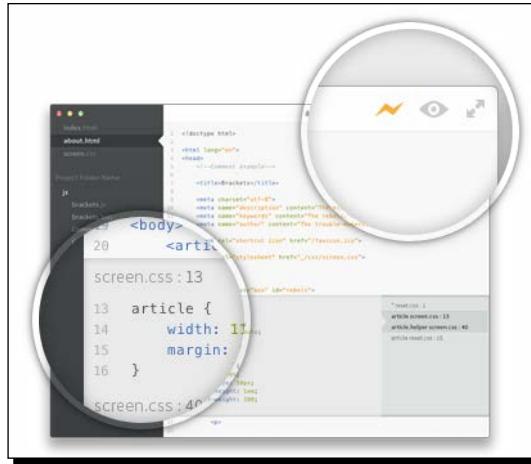
Adobe Brackets

Adobe Brackets provides a good integration with the PhoneGap build services. The PhoneGap build services are cloud-based services that handle for you the build process in order to let you concentrate on HTML/CSS/JavaScript. If you are planning to use the online services to handle the build process of your app, this is the tool for you!

Brackets is an open source editor for web design and development built on top of web technologies such as HTML, CSS, and JavaScript. The project was created and is maintained by Adobe, and is released under an MIT License.

Brackets is hosted on GitHub, within two different repositories: one for the main source code of the web application (the view of Bracket is built using HTML/CSS/JavaScript); the other one containing the source of the native application shell, which makes it possible for the editor to run as a standalone application on Windows and Mac OS (Linux as already said is on the bracket's roadmap). For this native application shell, Brackets uses the Chromium Embedded Framework that embeds a version of WebKit.

At the time of writing Brackets has two killer features: it lets you edit selected CSS rules directly inline by pressing *Ctrl + E* (Windows) or *command + E* (OS X), and then lets you see the changes in the browser in real time without saving the file.



Rendering changes in real time works the same as the Developer Tools of any major browser but with the ease of a text editor; however, at the time of writing, this feature is supported only by Chrome and you will be asked to restart your browser when activating this feature.

Brackets is greatly integrated with PhoneGap; in fact you can handle the build process directly from the editor. As mentioned earlier, you can use the build services provided by Adobe by registering at <http://build.phonegap.com>. Using Brackets you can access the build services and link your files with a build project.

Time for action – configuring the cloud service in Brackets

In order to be able to work with the cloud-based build services, you have to complete the following steps:

1. Create an account to use the PhoneGap build services here
<https://build.phonegap.com/plans>.
2. Open a terminal window and locate the user extensions folder in Brackets; for example, on OS X use the following command:

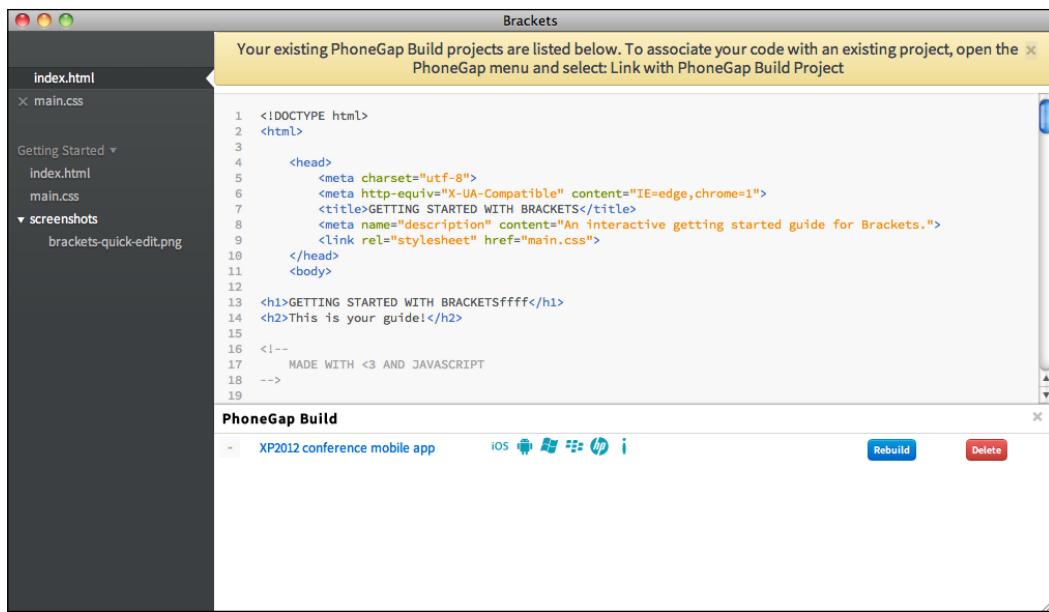
```
$ cd /Applications/Brackets Sprint XX.app/Contents/www/
extensions/user
```

Where xx is the sprint number of Brackets at the time you will test it.

- 3.** Clone the GitHub repository containing the *brackets-phonegap* extension available at <https://github.com/adobe/brackets-phonegap>:

```
$ git clone https://github.com/adobe/brackets-phonegap.git PGBuild
```

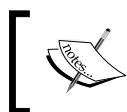
- 4.** Reload Brackets using the *Ctrl + R* or *command + R* shortcut and log in to your account to access the PhoneGap build service.



What just happened?

After you log in to your account, you will see a new menu item, which you can use to handle the build process smoothly.

The main advantage of introducing Brackets in your workflow is the native capability to get a live preview of the changes in Chrome and the capability to integrate with the online build services.



To download Brackets and to stay informed about new features and releases, refer to the GitHub project available at <https://github.com/adobe/brackets/>.

Eclipse

Eclipse is a multi-language integrated development environment (IDE) comprising a base workspace and an extensible plugin system for customizing the environment. The plugin ecosystem makes Eclipse a tool suitable for most programming languages and development environments. The **Android Development Tools (ADT)** is a plugin for the Eclipse IDE that enhances the IDE and lets you create an application UI, add packages based on the Android Framework API, and debug your applications using the Android SDK tools.

There are several distributions of Eclipse, considering that when working with PhoneGap you use HTML/CSS/JavaScript, a good configuration should be downloading Aptana Studio available at <http://www.aptana.com/products/studio3/> and installing the ADT plugin. The main advantage of using Eclipse is that it's an open source environment and that there are several tools and plugins to use to speed up your development process (i.e., there are plugins able to integrate the command-line tool in the IDE such as <https://code.google.com/p/elt/>).

Native, web, and hybrid apps

There is an ongoing debate among developers about the pros and cons of native, web, and hybrid apps. The truth is that each development approach has different points of strength.

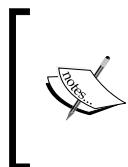
Native apps are marked by better performance, snappier animations and transitions, and faster load times. Furthermore, native apps can store more data offline and have full access to a device's hardware and OS features. Typically native apps are more expensive to build and the build costs increase significantly for each new platform because the code base needs to be reworked for each OS.

Web apps are built using web standards, so it's possible to have a common code base for all the supported devices and it's much easier to find developers with the appropriate skills.

On the con side, a web app is not able to access the entire device's hardware and OS features and is not installed on the user device, which means that users cannot access it when the device is not online.

A hybrid app can be viewed like a web app on steroids; in fact it uses web standards and can access most of the device's hardware and OS features. However, it's still subject to the store's approval process and revenue sharing and it's not immediately updated.

At first glance the hybrid app seems to be the perfect solution for every app, but actually it isn't. In fact I believe that the right solution depends on the software requirements (for example, a 3D game should be built as a native app while a card game may run just fine when built as a hybrid app). Note also that the current trend is to make mobile web apps look like native apps, which is why the hybrid approach seems the most reasonable right now.



It's not easy to state which kind of app can be developed with a hybrid technology and which kind of app has to be native. Most of the time the right choice is a balance between budget and performances; anyway you can find several interesting apps developed using PhoneGap at <http://phonegap.com/app/>.

Unless you are an experienced client-side developer, building a high-quality hybrid app can be a challenge because the available tools are still evolving and documentation is scarce. For this reason mastering in-browser debugging techniques is important for your success.

Working with desktop browsers

Because PhoneGap leverages open web standards (HTML, CSS, and JavaScript), you can start work in a desktop browser and then move on to a native project once the functionality is fleshed out. This way it's possible to speed up our development cycles and spend more time implementing core functionality. You can use the latest versions of any of the major desktop browsers Internet Explorer (IE), Google Chrome, Firefox, Safari or Opera to get started with a PhoneGap app. All of these browsers have Developer Tools for logging and debugging your code. PhoneGap is an intermediary layer that talks with the mobile device and the application; the app resides inside a chromeless browser, and using the PhoneGap API you can connect to phone features such as contact and camera.

The UI layer of a PhoneGap application is a web browser view that takes up 100 percent of the device width and height; think of the UI layer as a chromeless browser. The UI layer is known as **WebView**. The WebView used by PhoneGap is the same one used by the native operating system; on iOS this is the Objective-C `UIWebView` class, on Android it is `android.webkit.WebView`, and on Windows Phone 8 it is the `Object.DependencyObject.UIElement.FrameworkElement.WebView` class.



On every platform supported by PhoneGap the WebView provides a UI element that hosts HTML content within a native app.

Since there are differences in the WebView rendering engines between operating systems it's highly recommended to write cross-browser code.

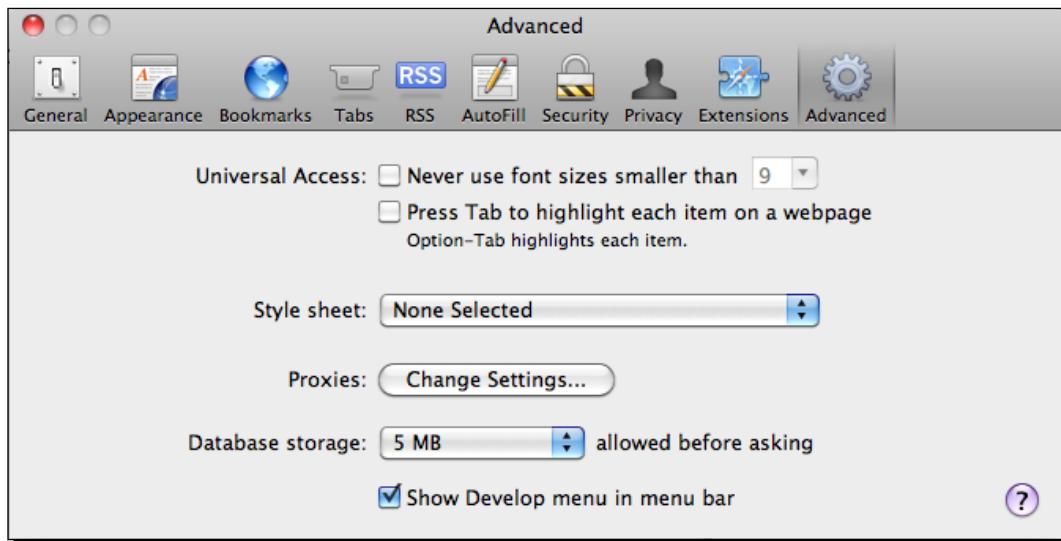
WebView apps have many of the advantages of native apps (for example, they can access phone APIs/features and discoverability via app stores).

Based upon web standards, PhoneGap doesn't require you to test in multiple browsers. However, it's a good habit to have a common code base and a platform-specific code base optimized for the WebView the app will use. In short, optimizing your apps for each browser is key for best performance across platforms. For this reason let's take a look at the tools each of the major browsers support as well as some debugging techniques. If you target primarily Android and iOS devices you only need to use a WebKit-based browser or the nightly build of WebKit.

WebKit debugging (Chrome, Safari, and Opera)

WebKit-based browsers support various debugging tools. For example, when encountering JavaScript issues you can launch the Web Inspector or the Developer Tools and start to explore logs and errors using the JavaScript console.

In Chrome you can access the Developer Tools from the **View** menu (**View | Developer | Developer Tools**); when working with Safari, you first have to enable the Developer Tools by opening Safari's **Preferences** panel and then selecting the **Show Develop menu in menu bar** checkbox.



You can then access the inspector by choosing **Show Web Inspector** from the application's **Develop** menu.

Since the Web Inspector is part of the WebKit codebase, you can use the same shortcuts in Chrome and Safari to access the debugging tools.

On Windows and Linux, press:

- ◆ *Ctrl + Shift + I* to open Developer Tools
- ◆ *Ctrl + Shift + J* to open Developer Tools and bring focus to the Console
- ◆ *Ctrl + Shift + C* to toggle the Inspect Element mode

On OS X, press:

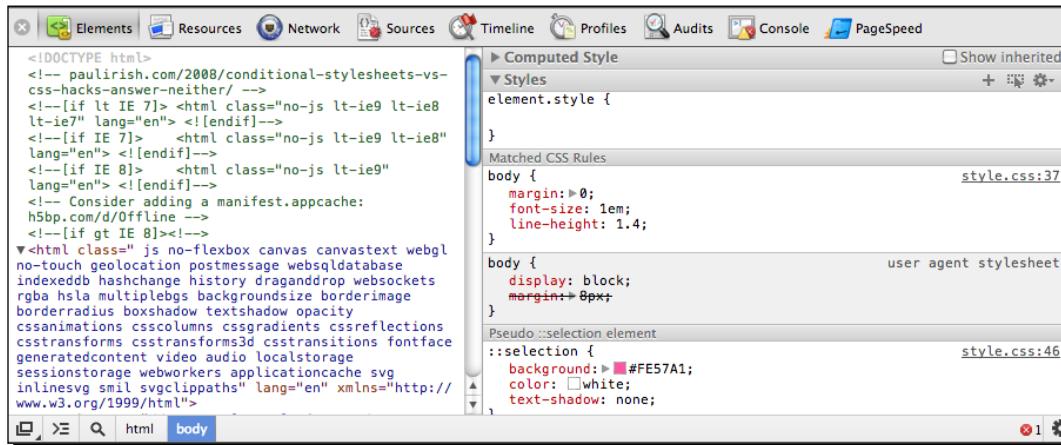
- ◆ *⌥ ⌘ I (option + command + I)* keys to open Developer Tools
- ◆ *⌥ ⌘ J (option + command + J)* to open Developer Tools and bring focus to the Console
- ◆ *⌥ ⌘ C (option + command + C)* to toggle Inspect Element mode

When accessing the Developer Tools you can switch between tools by clicking on the respective icon:



The **Elements** panel allows you to see the web page as the browser renders it. When using the **Elements** panel you can see the raw HTML, CSS, and explore the **Document Object Model (DOM)**. By clicking on the **Elements** panel and moving around the source of the page, you can identify the HTML blocks and change on-the-fly the CSS selectors value in order to experiment and fix possible rendering issues.

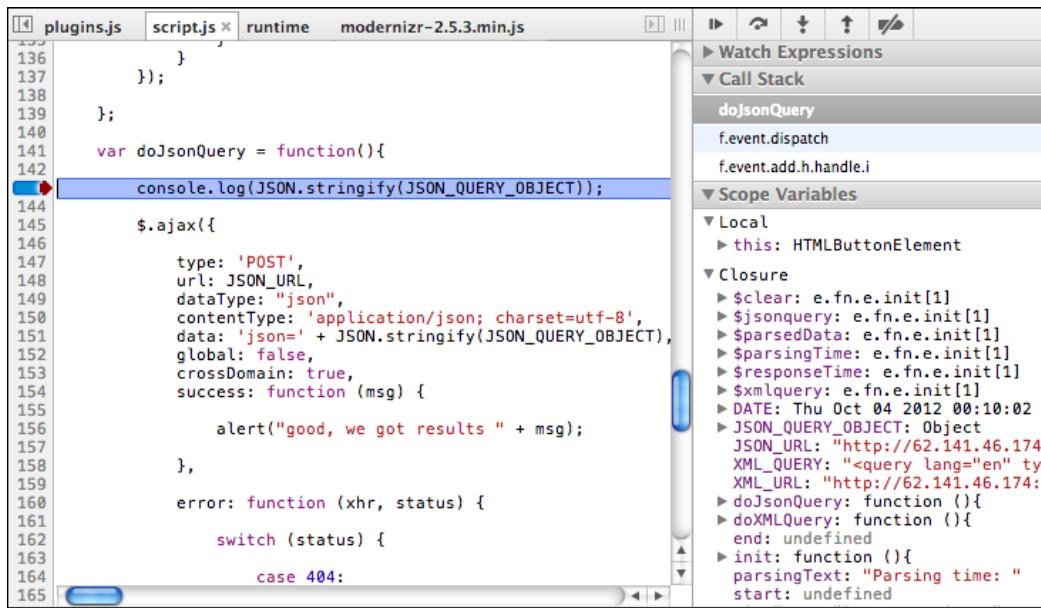
Before diving into the debugging techniques, let's review the WebKit Developer Tools available in Chrome and in Safari.



The **Resources** panel lets you inspect resources that are loaded/available to the inspected page. It lets you interact with Frame trees containing frame resources (HTML, JavaScript, CSS, Images, Fonts, and so on), HTML5 Databases, Local Storage, Cookies, and AppCache.

Using the **Network** panel you can explore the components a web page or application is requesting from web servers, how long these requests take, and how much bandwidth is required.

Using the **Sources** panel you can access all the resources loaded into the page. Use this panel to access the JavaScript, set breakpoints in the code, and explore the stack trace for each error. In order to set a breakpoint, select the script in which you want to set the breakpoint, and then click on the line number you are interested in. When the debug tool reaches the breakpoint, you can see what's happening in your code by exploring the call stack (i.e., the chain of functions and/or methods executed until this breakpoint) and the scope variables, and move in and out of functions.



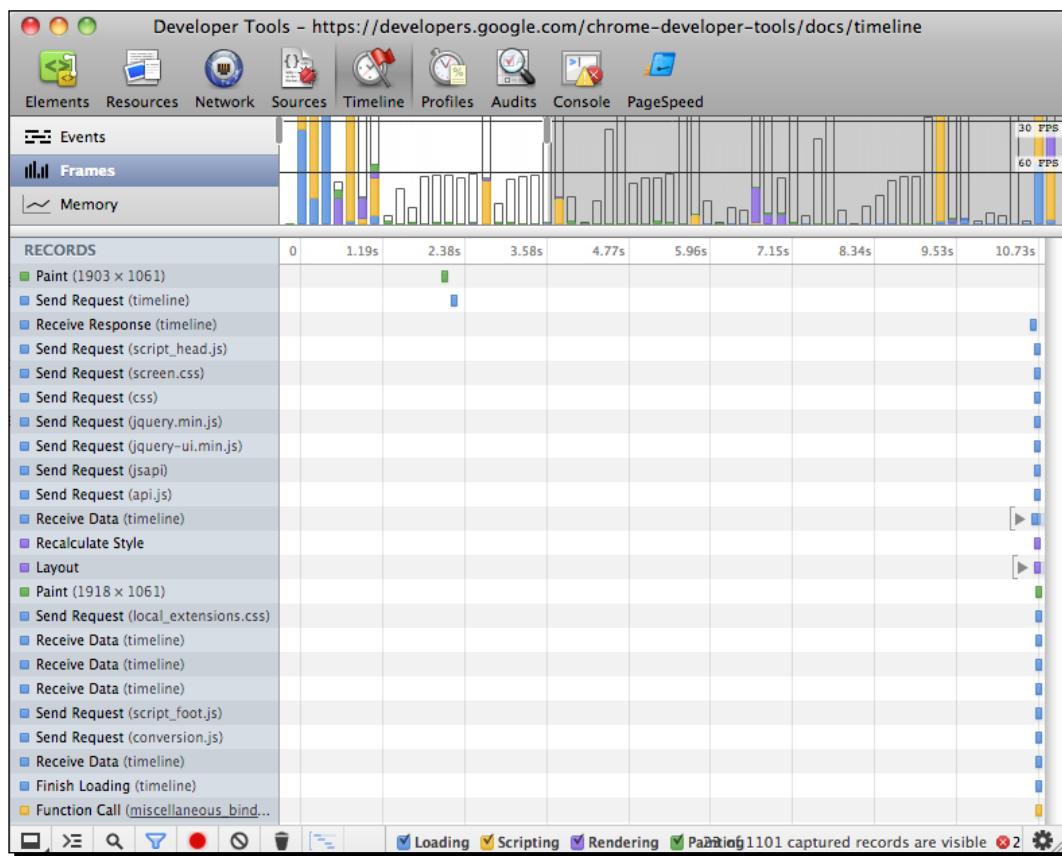
The JavaScript can be edited directly inside the debugger and you can see your changes on-the-fly by going back and forth using the navigation arrows. If you want the debugger to stop the code execution each time an exception is raised, use the Pause all button at the bottom left of the panel.

The **Timeline** panel lets you analyze the various WebKit behind-the-scenes activities such as how long the browser takes to handle DOM events, render page layouts, and handle events.

Once you press the record button, you can start to inspect what's happening in the page you are currently viewing.

The Events and Frames icons (available in Chrome) allow you to access two different timeline data views: the first one is based upon time and the second one is based upon frames; you can zoom into each view using the grey vertical controls at the top.

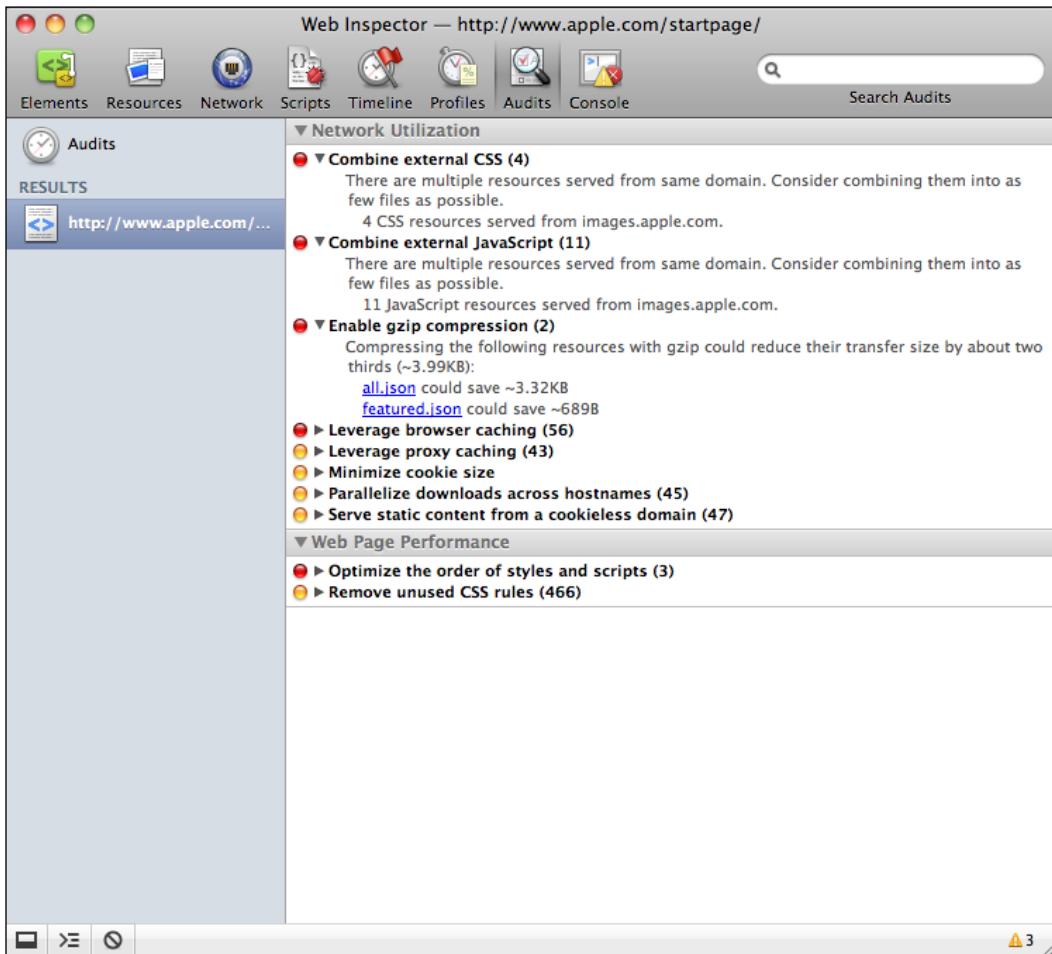
The Memory icon lets you explore the memory usage for a specific web page; in order to be more accurate during the exploration, it's a good habit to force the garbage collector by pressing the Trash icon at the bottom of the panel. Garbage collection is a form of automatic memory management; the collector attempts to reclaim garbage or memory occupied by objects that are no longer in use by the browser's window.



The **Profiles** tool helps you capture and analyze the performance of JavaScript scripts. For example, you can learn which functions take the most time to execute and then zoom in on possible bottlenecks and understand exactly where to optimize.

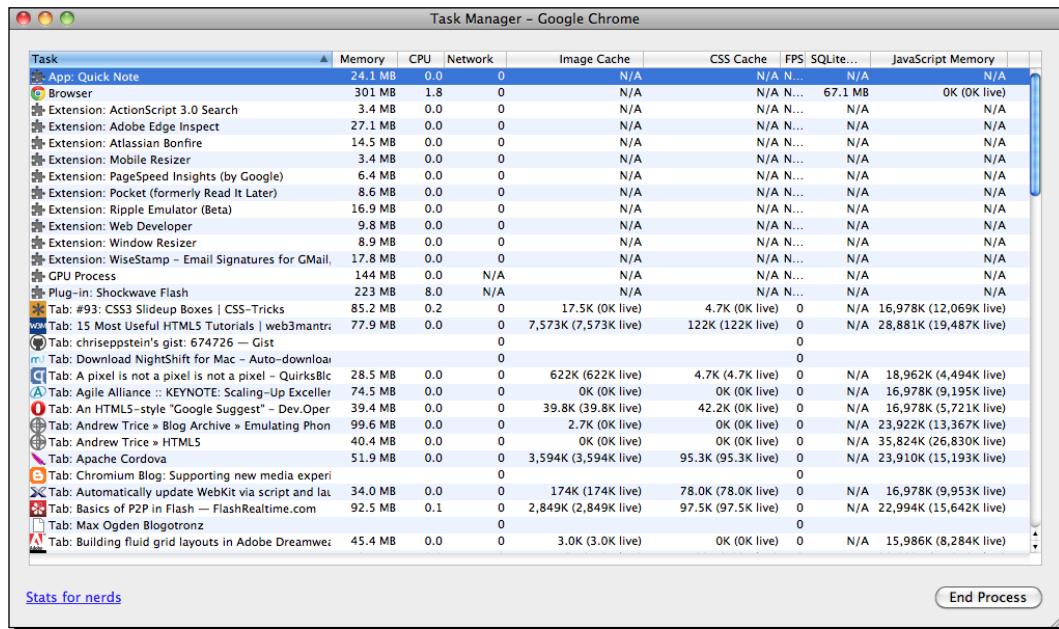
The **Audits** panel is like having your own web optimization consultant sitting next to you. This panel can analyze a page as it loads and then provide suggestions and optimizations for decreasing page load time and increasing perceived responsiveness.

Take a look at the following results when the Apple .com start page is loaded with the **Audits** panel opened. Also, the web page of Apple can be optimized, so use this tool as a means to improve your page load time and not as a metric. Once the result is acceptable you can move on to your next task.



Last but not the least is the **Console** panel. From the **Console** panel, you can enter arbitrary JavaScript code to programmatically interact with your page.

For best results, you may want to optimize the HTML/CSS/JavaScript of your app throughout the Development phase and not just at the end; the same holds true when developing mobile apps. A very useful tool available in Chrome is the Task Manager (**Window | Task Manager**), which helps you explore memory usage, the image cache usage, the SQLite storage, and more of each opened tab.



The screenshot shows the Google Chrome Task Manager window. It lists various tasks (tabs and extensions) along with their memory usage, CPU usage, network activity, and other performance metrics. The columns include:

Task	Memory	CPU	Network	Image Cache	CSS Cache	FPS	SQLite...	JavaScript Memory
App: Quick Note	24.1 MB	0.0	0	N/A	N/A N...	67.1 MB	N/A	N/A
Browser	301 MB	1.8	0	N/A	N/A N...	N/A	OK (OK live)	N/A
Extension: ActionScript 3.0 Search	3.4 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
Extension: Adobe Edge Inspect	27.1 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
Extension: Atlassian Bonfire	14.5 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
Extension: Mobile Resizer	3.4 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
Extension: PageSpeed Insights (by Google)	6.4 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
Extension: Pocket (formerly Read It Later)	8.6 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
Extension: Ripple Emulator (Beta)	16.9 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
Extension: Web Developer	9.8 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
Extension: Window Resizer	8.9 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
Extension: WiseStamp – Email Signatures for GMail,	17.8 MB	0.0	0	N/A	N/A N...	N/A	N/A	N/A
GPU Process	144 MB	0.0	N/A	N/A	N/A N...	N/A	N/A	N/A
Plug-in: Shockwave Flash	223 MB	8.0	N/A	N/A	N/A N...	N/A	N/A	N/A
Tab: #93: CSS3 Slidelup Boxes CSS-Tricks	85.2 MB	0.2	0	17.5K (OK live)	4.7K (OK live)	0	N/A	16,978K (12,069K live)
Tab: 15 Most Useful HTML5 Tutorials web3mantr...	77.9 MB	0.0	0	7,573K (7,573K live)	122K (122K live)	0	N/A	28,881K (19,487K live)
Tab: chrisepstein's gist: 674726 — Gist	0	0	0	0	0	0	0	0
Tab: Download NightShift for Mac – Auto-download	0	0	0	0	0	0	0	0
Tab: A pixel is not a pixel is not a pixel – QuirksBlc	28.5 MB	0.0	0	622K (622K live)	4.7K (4.7K live)	0	N/A	18,962K (4,494K live)
Tab: Agile Alliance :: KEYNOTE: Scaling-Up Exceller	74.5 MB	0.0	0	OK (OK live)	OK (OK live)	0	N/A	16,978K (9,195K live)
Tab: An HTML5-style "Google Suggest" – Dev.Oper	39.4 MB	0.0	0	39.8K (39.8K live)	42.2K (OK live)	0	N/A	16,978K (5,721K live)
Tab: Andrew Trice » Blog Archive » Emulating Phon	99.6 MB	0.0	0	2.7K (OK live)	OK (OK live)	0	N/A	23,922K (13,367K live)
Tab: Andrew Trice » HTML5	40.4 MB	0.0	0	OK (OK live)	OK (OK live)	0	N/A	35,824K (26,830K live)
Tab: Apache Cordova	51.9 MB	0.0	0	3,594K (3,594K live)	95.3K (95.3K live)	0	N/A	23,910K (15,193K live)
Tab: Chromium Blog: Supporting new media experi	0	0	0	0	0	0	0	0
Tab: Automatically update WebKit via script and lar	34.0 MB	0.0	0	174K (174K live)	78.0K (78.0K live)	0	N/A	16,978K (9,953K live)
Tab: Basics of P2P in Flash — FlashRealtime.com	92.5 MB	0.1	0	2,849K (2,849K live)	97.5K (97.5K live)	0	N/A	22,994K (15,642K live)
Tab: Max Ogden Blogotronz	0	0	0	0	0	0	0	0
Tab: Building fluid grid layouts in Adobe Dreamwe	45.4 MB	0.0	0	3.0K (3.0K live)	OK (OK live)	0	N/A	15,986K (8,284K live)

[Stats for nerds](#) End Process

You can customize the information available in the task manager by right-clicking on the **Task** column header.

If you type in the JavaScript console `performance.timing` you get a lot of useful information stored in the `PerformanceTiming` object such as the domain lookup time or the time to complete the DOM rendering. This information can help you identify possible bottlenecks in the view of your app.

The `performance.memory` property lets you know the JavaScript memory usage of your page. When used in conjunction with the `timing` property, `performance.memory` lets you know the numbers about your page. Using these numbers you can easily debug your app especially because there are objects you can use to display debug information on the screen.

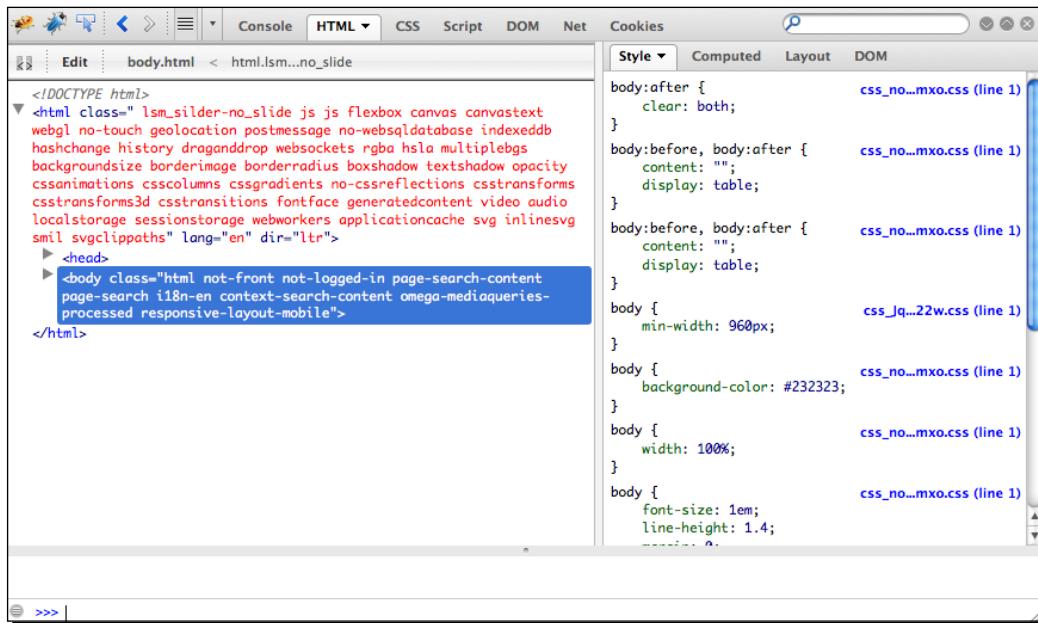
Gecko debugging (Firefox)

Firefox is based upon the Gecko open source layout engine used in many applications developed by the Mozilla Foundation and the Mozilla Corporation. New developers tend to prefer WebKit-based browsers; at the time of writing, Chrome has the largest install base market share followed by Internet Explorer and Firefox. Anyway it offers good debugging tools and it's evolving quickly including innovative projects such as Desktop WebRT, which lets you build a desktop web application at runtime that provides web apps with a native-like look and feel along with platform integration APIs on Windows, OS X, and other desktop platforms.

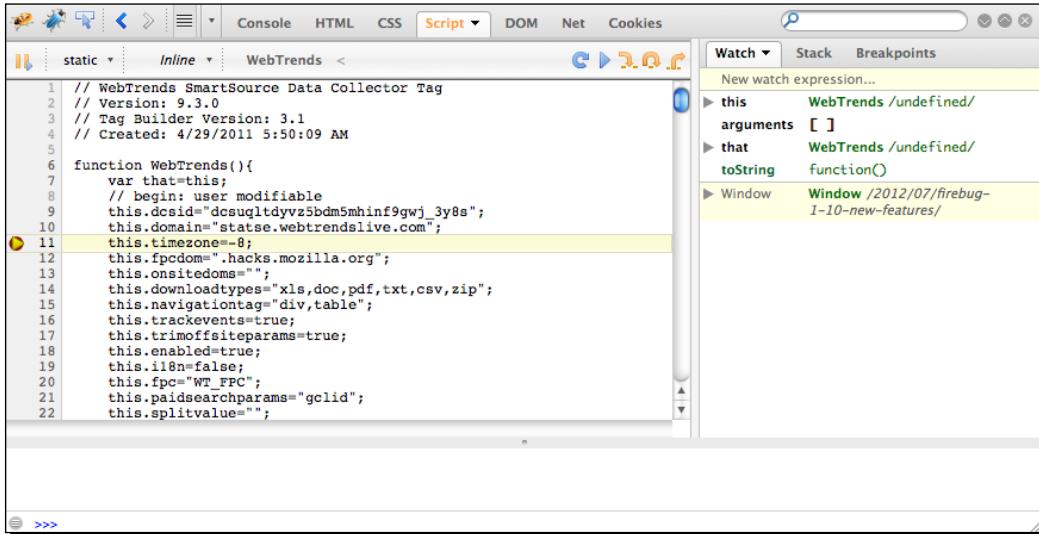
 Chrome also offers a way to build native apps based upon web standards. This technology is known as **Google Packaged Apps**, you can find more information about it at http://developer.chrome.com/apps/about_apps.html.

If you are not developing apps for Android or iOS, you can use the Firefox layout engine, which offers some powerful development and debugging tools. Let's explore quickly how to use Firefox to inspect and debug your app; as you will see, there are several similarities between the debug tools available in WebKit and in Firefox.

Firebug, which integrates with Firefox, puts a great set of Developer Tools at your fingertips that rivals the features of the WebKit Web Inspector. In order to install the Firebug extension you have to go to <https://getfirebug.com/downloads/> and install the latest version. Once installed, you can open the extension by navigating to **View | Firebug**.



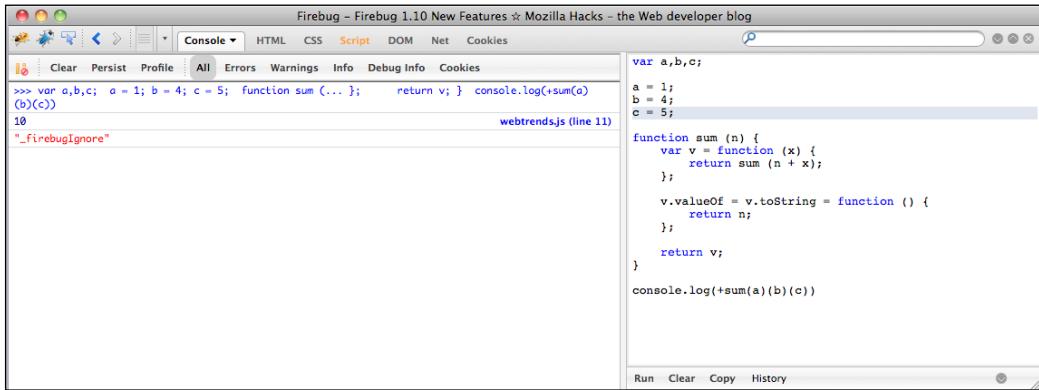
The Firebug toolbar gives you access to HTML source code and CSS rules, lets you explore and debug JavaScript functions, and more.



Once the debugger reaches a breakpoint you can:

- ◆ Explore the variables defined in the block of code in which you set up the breakpoint
- ◆ Explore the stack of function/method calls
- ◆ Create watches in order to understand how the content of a variable changes during the execution of the code

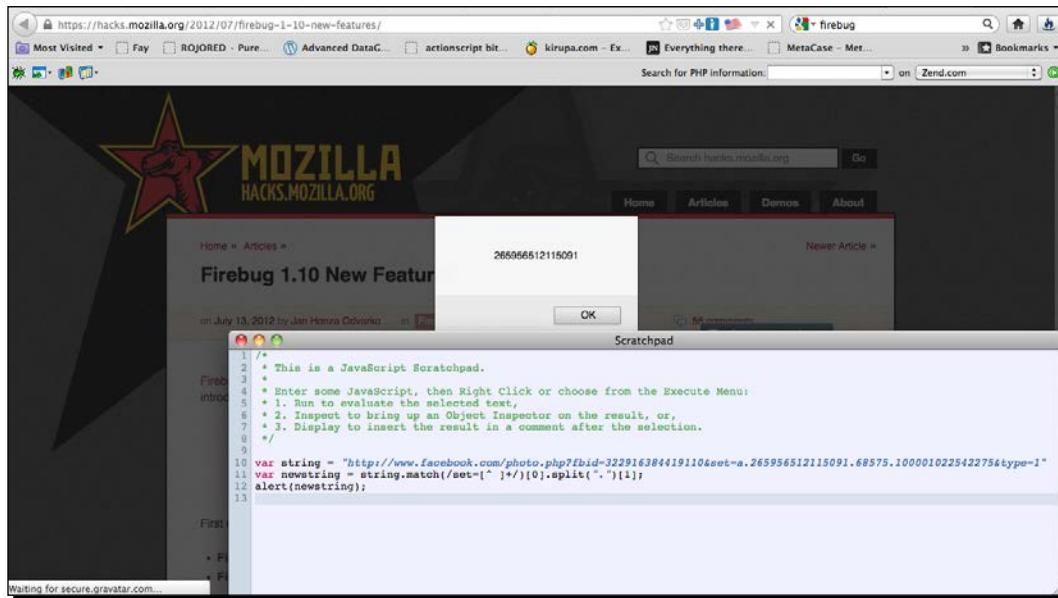
The Script console in Firebug is amazing. You can type your code in the right window and then run it and see the results in the console by clicking on the **Run** button at the bottom right of the panel.



To filter logs use the **All**, **Errors**, **Warnings**, **Info**, **Debug Info**, and **Cookies** selectors at the top of the window.

As mentioned previously, Firefox has three great native development tools: Scratchpad, Inspect, and Responsive Design View. You can access these tools through the menu bar by navigating to **Tools | Web Developer**.

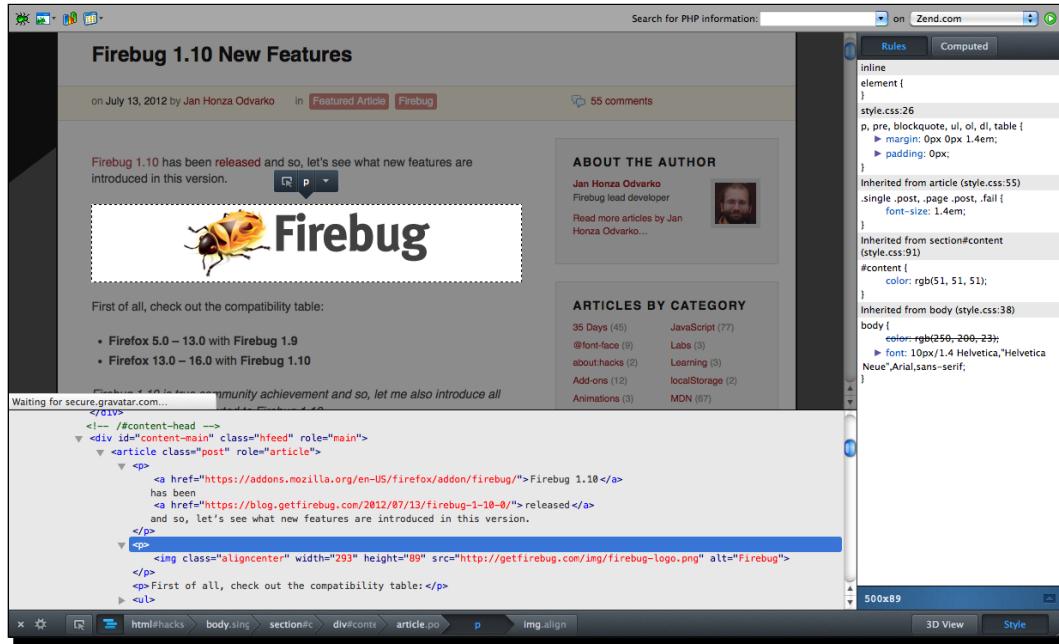
Think of Scratchpad as a text editor; you can use it to type and execute JavaScript. The difference between Scratchpad and the console is that it looks like a text editor and you can write all the code you want before executing it.



Another useful tool is Native Inspector (**Tools | Web Developer**). The Native Inspector is very well designed and very fast when executed; when selecting an HTML node, the UI makes what you selected clear and lets you change the CSS selectors and ID values contextually.

The navigation bar at the bottom and style panel is super easy to use and intuitive, even more you can access it from the bar to the 3D view.

The 3D view shows you how many nested elements are in your DOM.



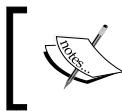
The Responsive Design View tool lets you change the resolution without resizing the browser. You can also use it to simulate device rotation.



Internet Explorer 10

Internet Explorer at the time of writing still has a wide install base; it's also the least favorite browser among developers. Virtually every developer has experienced serious issues when optimizing a web page for IE; this is due to the fact that IE in significant areas diverged from the web standards, but things are changing and the preview of IE 10 is getting good scores in various tests.

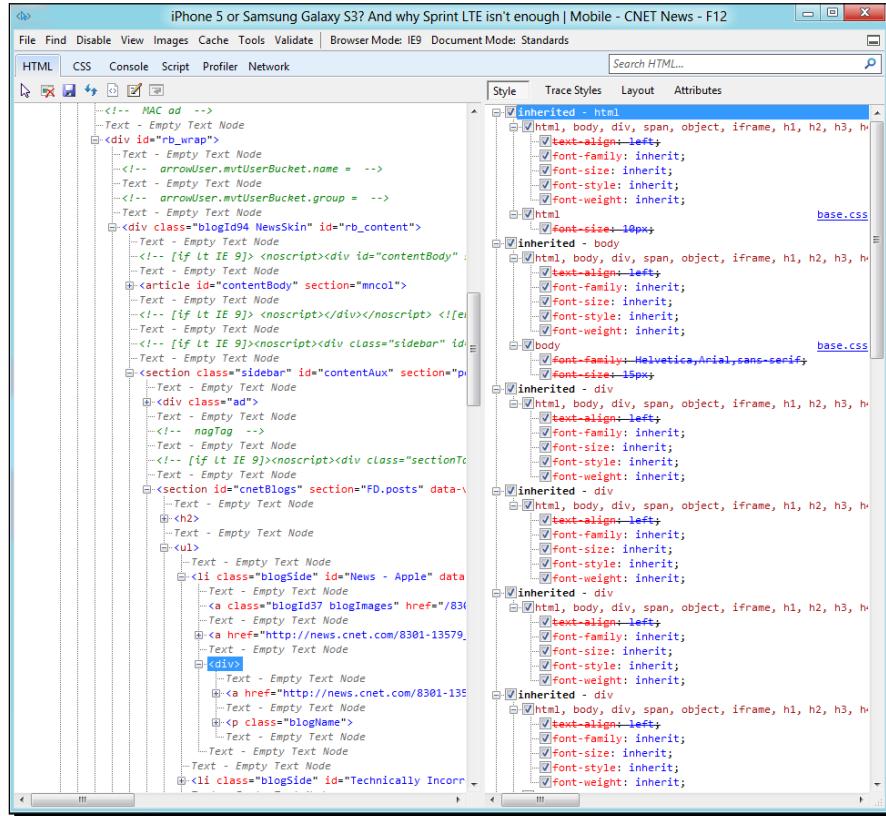
name	score	Ringmark	Security	Rich Text	Selectors API	Network	Acid3	JSKB	# Tests
Chrome 24 →	75/100		16/17	1046/1308	100.0%	12/16	100/100	81	2420
Firefox 18 →	72/100		13/17	942/1308	100.0%	13/16	100/100	81	3065
IE 10 →	75/100	39/77	14/17	492/1308	100.0%	12/16	100/100	81	754
Safari 6.0.2 →	73/100		14/17	1057/1308	100.0%	11/16	99/100	81	418



In order to get a detailed overview of the support of the web standards in each browser, you can refer to the HTML5 Test tool available at <http://html5test.com/compare/browser/chrome27/ff22/ie10.html>.

Developer Tools were introduced in Internet Explorer 8, and updated with new functionality in Internet Explorer 9. Developer Tools in Internet Explorer 10 add Web Worker debugging and support for multiple script sources.

You can access the Developer Tools by pressing **F12** or by navigating to **Tools | Developer Tools** from the menu bar.



The IE 10 Developer Tools provide a similar user interface to the Developer Tools in Safari, Chrome, and Firefox.

Mobile debugging workflow

As you have seen each browser offers different debugging tools and each tool has its pros and cons. Regardless of which tool you use, however, your debugging workflow is the same.

When investigating a specific problem, you will usually follow this process:

- ◆ Find the relevant code in the debugger's code view pane.
- ◆ Set breakpoint(s) where you think interesting things may occur.
- ◆ Run the script again by reloading the page in the browser if it's an inline script, or by clicking on a button if it's an event handler.
- ◆ Wait until the debugger pauses execution and makes it possible to step through the code.
- ◆ Investigate the values of variables. For example, look for variables that are undefined when they should contain a value, or return `false` when you expect them to return `true`.

If necessary, you can use the console to evaluate code or change variables for testing. You can also execute complex JavaScript code and test a solution before implementing it.

Identifying the problem by learning which piece of code or input caused the error conditions and isolating it is a suitable approach. However, with mobile apps things are not always so straightforward. The advantage of PhoneGap is that you can develop and debug in a common environment such as the browser, but keep in mind that a mobile app has to be tested and debugged on the target devices as well.

Remote debugging

Remote debugging is the process of debugging a program running on a system different from than the debugger. To start remote debugging, the debugger connects to a remote system over a network. Once connected, the debugger can control the execution of the program on the remote system and retrieve information about its states.

Remote debugging has come a long way over the past few years; the most common and easy-to-use debuggers include **iWebInspector**, **weinre**, and **Adobe Edge Inspect**.

Using iWebInspector (OS X only)

iWebInspector is a free tool to debug, profile, and inspect web applications running on the iOS Simulator. You can check resources, see and change HTML tags and CSS rules, use breakpoints on JavaScript code, create charts, and more, just as if you were on Safari for Desktop, Chrome, or Firebug. For the latest available build of iWebInspector go to <http://www.iwebinspector.com/>.

Time for action – configuring iWebInspector for iOS debugging

In order to debug an Apache Cordova/PhoneGap app using iWebInspector, you have to perform the following steps:

1. Find the `application:didFinishLaunchingWithOptions` method in the `AppDelegate.m` file.

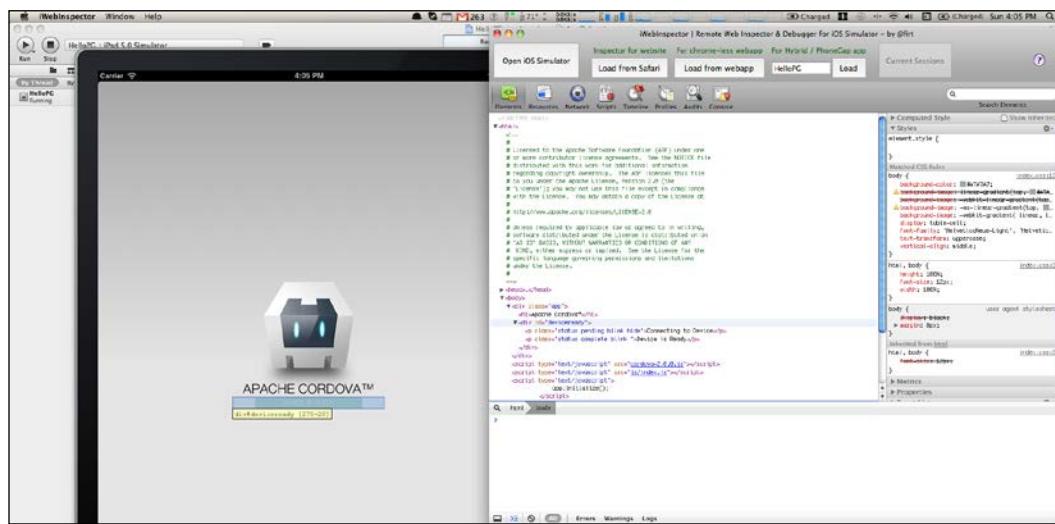
2. Add the following snippet of code inside the method (it works with iOS 5 and greater):

```
Class class = NSClassFromString(@"WebView");
[class performSelector: @selector(_enableRemoteInspector)];
```

3. Once added the snippet creates a build.

4. Run the app in the emulator.

5. Use iWebInspector to load the app and play with the debug tools as you do when developing for the browser.



What just happened?

The app can be inspected as an HTML page; you can use all the browser debug techniques when emulating the app.

Debugging with weinre

With the acronym **weinre** (**WEb INspector REmote**) developers refer to a debugger for web pages, such as Firebug (for Firefox) and Web Inspector (for WebKit-based browsers), except it's designed to work remotely, and in particular, to allow you debug web pages on a mobile device.

Time for action – configuring Node.js and weinre

At first glance, using weinre may look complicated but if you follow these steps you will be able to debug any app from your desktop:

- 1.** Download and install Node.js from <http://nodejs.org/>.
- 2.** Install the package manager npm available at <https://npmjs.org/>.
- 3.** Install weinre using the command-line tool:

```
$ npm install weinre -g
```
- 4.** Start weinre using boundHost as your network IP address (i.e., the one other devices can use in your LAN to reach your computer):

```
$ weinre -boundHost xxx.xxx.xxx.xxx
```
- 5.** Copy in the head of your page the script needed to include the JavaScript that enables remote debugging:

```
<script src="http://xxx.xxx.xxx:8080/target/target-script.js#anonymous"></script>
```
- 6.** Build your app, install it on a real device connected to the same LAN, and start debugging it.

Be sure to use the right IP address and to add the `script` tag at the beginning of the page; if you follow these two steps carefully, you're up and running with weinre in no time. The word `anonymous` you see at the end of the script is used in order to uniquely identify the debug session; in fact, the online services assign a random ID to each session.

What just happened?

You configured a private weinre server to be used locally or remotely and you have full control on it.

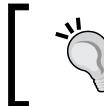
When using the PhoneGap build services, you can enable remote debugging in the app build settings and access the debug services provided by Adobe at <http://debug.phonegap.com/>; the services are powered by weinre.

Wireless debugging with Adobe Edge Inspect

Edge Inspect allows you to easily pair multiple smartphones and tablets with your computer and enables you to work more efficiently by providing synchronous browsing, remote inspection, and the ability to take screenshots of your mobile web content from connected devices.

In order to use Edge Inspect, you need an account on the Adobe Creative Cloud website and Chrome running on your desktop.

This tool is part of a paid subscription but is really helpful because it's pretty fast to set up and allows you to perform remote debugging using the PhoneGap debug services. A detailed list of features is available on the Adobe website at <http://forums.adobe.com/docs/DOC-2496>.

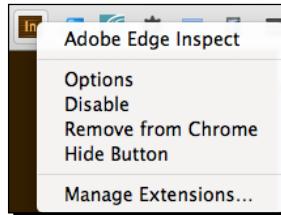


There is a free version of Adobe Edge Inspect. It only allows one connected device at a time and doesn't do screenshots. It's not a trial but completely free software.

Time for action – integrating Edge Inspect and weinre

The latest release of Edge Inspect (previously known as Adobe Shadow) allows you to set up your custom weinre server instead of the default Adobe's hosted one. Follow these steps in order to set up your own server:

1. Right-click on the extension icon in Chrome.
2. Select **Options**.



3. Change the **weinre server** option to **custom weinre server**.
4. In the text input, set up your local or remote address to weinre.

What just happened?

You connected the extension to a private server avoiding any kind of traffic outside your network (it could be very important when working on NDA projects).

iOS 6 remote debugging

iOS 6 introduces the Safari Web Inspector to both the iPhone and the iPad via the Safari Remote Debugging interface.

It works similar to Chrome for Android remote debugging and behaves in a similar fashion, such that when you select an item in the Safari desktop inspector, it is highlighted on the iOS device. You can enable the Web Inspector by navigating to **Settings | Safari | Advanced** and use it to debug:

- ◆ A safari window on your iOS device or simulator
- ◆ A chromeless web app installed on your iOS device or simulator
- ◆ A native app using a WebView such as PhoneGap apps

Mimicking mobile counterparts

Several years ago I had a conversation with a cool guy working at Opera, Charles McCathieNevile. He told me in mixed English/Italian, "Mobile is tutto casino!" (Mobile is total chaos). This was true in 2004 and it's still just as true today due to the high device fragmentation and difference in screen resolutions.

In order to emulate devices you can use several tools including the ones provided by the major SDK vendors. Usually these tools are time-consuming and sometimes not easy to configure; for this reason, I strongly encourage you to use the web as your development and debugging platform.

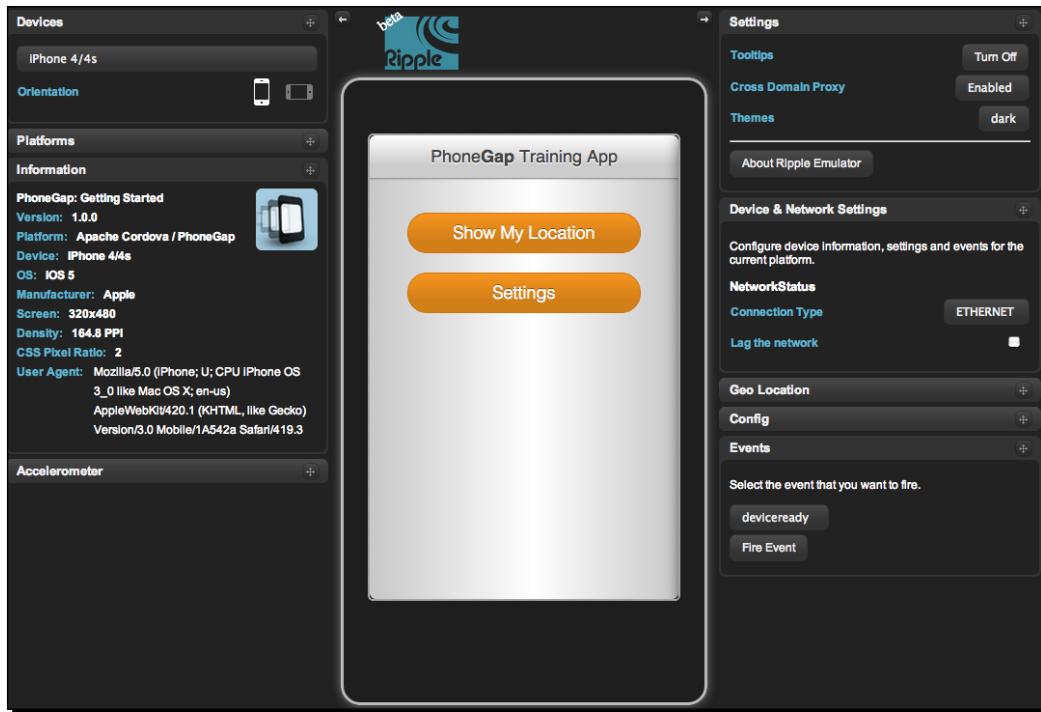
One of the greatest advantages to using PhoneGap is that you can use web standards and thus avoid having to continuously test the app in the browser emulator that comes with the SDK of each mobile platform. It doesn't mean that you don't need to emulate the app at all or that you don't have to test it in a real device, but you can safely assume that it's enough to test the UI and the main user interactions inside your desktop browser.

There is a very interesting open source project called **Ripple** that lets you emulate several mobile platforms in your browser. Unfortunately, at the time of writing, it's available only for Chrome. You can get the Ripple extension for Chrome at <https://chrome.google.com/webstore/detail/ripple-emulator-beta/geelfphabnejjhdkljhgipohgpdnoc>; after you install the extension, you can access it from the top right of your Chrome installation.



By clicking on the Ripple icon, the extension starts to emulate one of the available platforms, including Mobile Web, PhoneGap, and BlackBerry 10 Web Works. The Mobile Web platform offers several devices to emulate with skins and different orientations.

When selecting the PhoneGap platform, you can emulate the device and test in a familiar browser environment the app features and interactions with the Apache Cordova framework.



All the relevant device information and settings are available in the same window you can use to debug your app.

On the left side you can select the device, its orientation, get detailed information about the device, and simulate the accelerometer. On the right side it's possible to set up the network settings, the lag, the geo location information, and play with events. You can select the event you want to emulate and then fire it; in this way you can easily simulate device-specific events such as the pressure of the back button or of the home button. If you have an appropriate XML configuration file in the app folder, you can get a graphical representation of it inside Ripple.

Have a go hero – improve the "Hello World" app

Review the instruction provided to add interactivity to the "Hello World" app and make the necessary changes to open a modal window when the app starts.

Pop quiz – getting started with mobile apps

Q1. Is it possible to debug an iOS app installed on a device using iWebInspector?

1. It's not possible at all.
2. Yes, it's the main purpose of this software.
3. Yes, it's the main purpose of this software but you have to make some changes in the source code.

Q2. Which is the main advantage in using Adobe Edge Inspect and weinre together?

1. There is no particular advantage.
2. The app source code is executed faster.
3. You can simultaneously debug multiple devices running on the same app.

Summary

After providing an overview of several tools and some debugging techniques, this chapter prepared you to move to the next step, that is, designing and deploying a multi-platform app with Apache Cordova and its PhoneGap distribution.

3

Getting Started with Mobile Applications

In Chapter 2, Building and Debugging on Multiple Platforms, you learned how to configure your development environment, emulate a mobile application, and choose the most useful tools to debug it. In this chapter you will learn how to improve the performance of a mobile app. You will see how to define the building blocks of a modern hybrid app built using web standards and PhoneGap.

In this chapter you will:

- ◆ Review some basic knowledge of mobile web development
- ◆ Explore best practices to write quality code for your app
- ◆ Review some best practices for performance optimization
- ◆ Get an overview of the most popular app frameworks
- ◆ Learn how to create a project folder structure optimized for cross-platform development with PhoneGap
- ◆ Create a "Hello World" app using a framework
- ◆ Learn how to add a native look and feel to your app

Mobile-centric HTML/CSS/JavaScript

When using PhoneGap, you create hybrid apps based upon standards. The app is rendered to the user through a `WebView`, which means it is a browser instance wrapped into the app itself.

For this reason, it's important to know how to use mobile-specific HTML tags, CSS properties, and JavaScript methods, properties, and events.

The viewport meta tag

The `viewport` meta tag was introduced by Apple with iOS 1.0 and is largely supported in all the major mobile browsers. When a web page doesn't fit the size of the browser, the default behavior of a mobile browser is to scale it. The `viewport` meta tag is what you need in order to have control over this behavior.

A `viewport` meta tag looks like the following code snippet:

```
<meta name="viewport" content="width=device-width,  
height=device-height, initial-scale=1, minimum-scale=1,  
maximum-scale=1.5, user-scalable=1">
```

What you are actually saying to the browser is that the default width and height of the content are the width and height of the device screen (`width=device-width` and `height=device-height`), that the content is scalable (`user-scalable=1`), and what the minimum and maximum scale is (`minimum-scale=1` and `maximum-scale=1.5`).

An exhaustive reference about the `viewport` meta tag is available on the Apple Developer Library website at <https://developer.apple.com/library/safari/#documentation/appleapplications/reference/SafariHTMLRef/Articles/MetaTags.html>. Some useful information is available on the Opera developers website at <http://dev.opera.com/articles/view/an-introduction-to-meta-viewport-and-viewport/>.

Remember that by default the `WebView` used by PhoneGap ignores the settings defined in the `viewport` meta tag; you will learn during this chapter how to enable the handling of the `viewport` settings in your app.

Unwanted telephone number linking

Mobile browser click-to-call format detection on most phones isn't that accurate; plenty of numbers get selected, including addresses, ISBN numbers, and a variety of different types of numeric data that aren't phone numbers. In order to avoid any possible issues and to have full control on a call from your HTML markup, it is necessary to add the following meta tag to the header of your page:

```
<meta name="format-detection" content="telephone=no">
```

Defining this tag you can then control how to handle numbers using the `tel` or the `sms` scheme in the `href` attribute:

```
<a href="tel:18005555555">Call us at 1-800-555-5555</a>
<a href="sms:18005555555?body=Text%20goes%20here">
```

Autocorrect

Submitting data using mobile devices is a tedious operation for the user, because sometimes the built-in autocorrect features don't help at all. In order to disable the autocorrect features, use the `autocorrect`, `autocomplete`, and `autocapitalize` attributes in conjunction with an `input` field:

```
<input autocorrect="off" autocomplete="off" autocapitalize="off">
```

CSS media queries and mobile properties

One of the interesting features of CSS is media queries. Media queries themselves are actually quite old and are not mobile-specific, but they are really useful when handling different screen sizes on mobiles. Media queries can be used inline:

```
@media all and (orientation: portrait) {
    body { }
    div { }
}
```

Or as the `media` attribute of a link tag:

```
<link rel="stylesheet" media="all and (orientation: portrait)"
      href="portrait.css" />
```

There is no best way to use them because it depends on the type of app. Using media queries inline, the CSS file tends to grow and the parsing can be slow on old devices. On the other hand, having CSS rules organized in separate files helps to keep the code well organized and speeds up the parsing, but it means more HTTP calls that are usually not the best option on mobiles due to the latency of mobile connections.

A good balance should be reached using offline caching strategies, which you will learn more about in the next chapters.

There are several CSS mobile-specific properties; most of them are vendor-specific and are identified with prefixes. The most common properties used in mobile development are:

- ◆ -webkit-tap-highlight-color (iOS): This overrides the semitransparent color overlay when a user clicks on a link or clickable element, this is the only property which is iOS specific
- ◆ -webkit-user-select: This prevents the user from selecting text
- ◆ -webkit-touch-callout: This prevents the callout toolbar from appearing when a user touches and holds an element such as an anchor tag

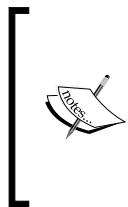
Always remember that the usage of browser prefixes in JavaScript is possible only using mixed case or lower camel case formatting, which means that in order to prevent the user from selecting text through JavaScript you have to use the following syntax:

```
yourElementVariable.style.webkitUserSelect = 'none';
```

The camel case formatting is due to the fact that the dash sign cannot be used in a variable name in JavaScript.

JavaScript for mobile 101

With JavaScript, you can add interactivity to an app and define its behavior. The JavaScript community is really vibrant and there are thousands of frameworks and utilities around the web you can use to speed up development or enhance the user experience.



In this book, I will briefly cover relevant JavaScript syntax where necessary. However, if you need to consult a reference guide for JavaScript, check out *David Flanagan's JavaScript: The Definitive Guide* or visit the Mozilla Developer Network at <http://developer.mozilla.org> or the Apple Safari Developer Library at <http://developer.apple.com/library/safari>.



One of the most common questions among new developers is whether it's a good habit to use external frameworks. One school of developers strongly opposes the use of external frameworks, because they add weight to an app. In my opinion it depends on the nature of the app. This doesn't mean that a complex app needs a framework and a simple one doesn't; instead, it depends on the architecture and on the features of your app.

I use jQuery and I love it, but I will not advise the use of jQuery when building a hybrid multi-page app. Also, if the jQuery library is downloaded once, the file is parsed each time it's included in an HTML page. Performance on mobiles is crucial. If you don't seriously consider optimizing each aspect of your app, you risk losing users. Bad performance can also lead to high battery consumption.

There aren't too many tools to analyze performance on the market right now, so you have to rely on the ones you have in your browser. You can test your pages using several resources such as <http://gtmetrix.com/>, <http://blazemeter.com/>, and <http://mobitest.akamai.com/>.

Let's review the basics of adding interactivity without using jQuery and some mobile-specific JavaScript APIs.

querySelector and querySelectorAll

The `getElementById()` method accesses the first element with the specified ID. It's one of the oldest and most well-known methods of the document object; these days, however, it is often replaced by jQuery selectors that are usually more accurate and provide better cross-browser support.

There are also other methods to inspect the DOM, including `querySelector` and `querySelectorAll`. `querySelectorAll` is a new DOM API that accepts CSS classes, IDs, or HTML tags and returns the element(s) it matches.

```
// Recover the first <p> tags in the current document
var first = document.querySelector ('p');

// Recover all the <p> tags in the current document
var all = document.querySelectorAll('p');
```

addEventListener

One of the means we have to ensure unobtrusive JavaScript is the `addEventListener` function, the DOM level 2 mechanism to register event listeners.

Using `addEventListener` you can get a better separation of concerns, define multiple listeners, and handle custom events efficiently as you will do in the "Hello World" sample app that will listen for the device to be ready before enabling its controls. It's good practice to separate application logic from the event handlers in order to keep the code cleaner and to make it easier writing tests.

```
addEventListener('event', eventHandler);  
  
function eventHandler(evt) {  
  
    // Call another function and eventually pass the  
    // values stored in the event object  
  
}
```

Don't pass the event object around but just the values needed by the function you are calling.

Screen orientation

The screen orientation is important when dealing with an app because the size of the screen dramatically changes when the orientation is changed. The `orientationchange` event is triggered at every 90 degrees of rotation (portrait and landscape modes), and it's possible to listen to it using `addEventListener`; the current orientation is available through `window.orientation`.

Device orientation

If you want to get more detailed information about the orientation of the device, you can define a listener for the `deviceorientation` event. The `deviceorientation` event will fire very frequently and gives information about the device's orientation in three dimensions.

The `deviceorientation` event is strictly related to the existence on the device of a gyroscope; the gyroscope measures the 3D angle orientation, even when the device is at rest.

Shake gestures

Gesture handling is the key for successful apps. The `devicemotion` event fires when the user shakes or moves his/her device. The `devicemotion` event is strictly related to the accelerometer, which fires events off when the device accelerates.

Media capture API

While old versions of iOS are still lacking basic file input, Android, iOS Version 6 and later, Windows Phone 8 and BlackBerry 10 are giving developers fine-grained control over content users can upload and allow you to access the device camera and microphone.

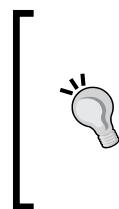
```
<!-- opens directly to the camera -->
<input type="file" accept="image/*;capture=camera"></input>
<!-- opens directly to the camera in video mode -->
<input type="file" accept="video/*;capture=camcorder"></input>
<!-- opens directly to the audio recorder -->
<input type="file" accept="audio/*;capture=microphone"></input>
```

Data URI

You can represent an image as a Base64 string, which ensures higher performance because there is no TCP negotiation in order to open a new HTTP connection. Practically speaking it means that there is a lower latency when compared to the usual way to load an image on the Web. When a `base64` string is assigned as the `src` attribute to an `img` tag the code looks like the following snippet:

```
<img src='data:image/png;base64,
R0lGODlheAAOALMAOazToeHh0tLS/7LZv/0jvb29t/f3//Ub//ge8WSLf/rhf/3kdbW1mxsbP//mf///
yH5BAAAAAAALAAAAAQAA4AAARe8L1Ekyky67QZ1hLnjM5UUde0
ECwLJoExKcppV0aCcGCmTIHEIUEqjgaORCMxIC6e0CcgwWw6aFj
sVMkkIr7g77ZKPJjPZqIyd7sJAgVGoEGv2xsBxqNgYPj/gAwXEQA7' width='16'
height='14' >
```

When converting an image to Base64 there is a 30-40 percent weight increase; for this reason, you have to optimize the image carefully before converting it and when possible activate `gzip` compression on the server.



Also, if JPEG and PNG formats are already compressed and `gzip` cannot compress them any further, you can use the `Base64OutputStream` Apache codec available at <http://commons.apache.org/proper/commons-codec/apidocs/org/apache/commons/codec/binary/Base64OutputStream.html> in order to apply additional compression to the Base64 string.

Rendering a Base64 image can be very CPU-expensive for a mobile device. Consider carefully if you really need a Base64 image for your app.

Performance best-practices

The success of any mobile web application relies on design and performance. An entire book could be written about performance best practices especially when dealing with mobile web and native apps. However, there are some fundamentals that are important to review, because with PhoneGap you are actually working with a browser instance embedded in a native app.

When possible use small images to render the UI elements of your app. A very good approach is the one you can see in action when opening Gmail that uses small GIF files combined to render the user interface, ensuring a very short processing time. The images are not rendered directly in the UI; instead they are combined using different CSS rules. Furthermore, the images are usually grouped together and then rendered using a technique known as **CSS Sprites**.

Generally speaking, inline CSS and JavaScript are less expensive for the device, but they are not good practice at all. You can use the server to inject in the page inline styles and JavaScript and avoid the common issues that you may face on combining views and application logic. The server-side inclusion allows you to optimize the app working on the views. For instance, you can implement a data cache mechanism so that the app loads just once (or when a significant update occurs) the required resources. The only pitfall I can see is when there is no connection, or if the quality of the connection is not fair.

The most recent devices are very powerful but the global mobile market includes billions of devices with several performance limitations. A common performance issue occurs when you try to load and parse big files such as external libraries. A reasonable approach is to avoid embedding big files and identify the needs for each view of your app avoiding unnecessary scripts or libraries. The same approach can be used for CSS; in fact, you can reduce the size of external CSS files by identifying the properties needed to properly render each view of your app.



Be careful when using the Google Analytics JavaScript library, because the JavaScript used by GA forces a dynamic network request that cannot be cached. Thus, even though the site could have been rendered from cache, the phone still has to pay the high cost of setting up a 3G session. To get analytics about your app consider downloading the Mobile App Analytics SDK from Google available at <http://analytics.blogspot.com.es/2012/10/mobile-app-analytics-updates-and-public.html>.

Generally speaking, always try to reduce the number of external files loaded by your app, first loading only what you really need and then compressing and combining multiple files.

You have probably heard that you should always use GPU hardware acceleration to improve the performance of your app. This is a very easy task to perform. In fact, it's enough to add a CSS `translate3d` transform style:

```
-webkit-transform: translate3d(0, 0, 0);
transform: translate3d(0, 0, 0);
```



GPU is short for **graphics processing unit**, a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display.

As often happens, easy-to-use techniques can lead you to bad results. When enabling hardware acceleration on an HTML element you have to consider that the element will be uploaded to the GPU (actually the upload happens each time the HTML element is modified). For this reason you have to follow some rules:

- ◆ Don't enable the GPU hardware acceleration on all the HTML elements
- ◆ Keep the nesting structure of the HTML elements as simple as possible
- ◆ Enable the GPU hardware acceleration only for HTML elements that are not frequently updated



If you are experiencing issues with the GPU, you can try to identify the issue using your browser. On Safari, for instance, you can launch the browser enabling the Safari Debug Menu.

```
$ export CA_COLOR_OPAQUE = 1
$ export CA_LOG_MEMORY_USAGE = 1
$ /Applications/Safari.app/Contents/MacOS/Safari
```

When using Chrome you can type in `about:flags` in the URL bar, scroll down a few items, and click on **Enable for the FPS counter**.

In Firefox type `about:config` in the URL bar and in the **Firefox Web Console** section set `webgl.verbose = true`.

If you want to learn more about GPU debugging, take a look at the interesting presentation available at <http://goo.gl/xh4Qr>.

Another important point to consider is how to update HTML elements minimizing the browser reflow.



Reflow is the name of the web browser process for recalculating the positions and geometries of elements in the document.

Reflow processes are invoked any time the DOM content changes, DOM elements are resized, CSS positioning/padding/margins are changed, and so on. In order to avoid performance issues you have to follow some simple rules:

- ◆ Use fixed-width and -height layouts for DOM elements
- ◆ Avoid deeply nested HTML DOM structures
- ◆ Preload images or assets used in CSS styles
- ◆ Append new DOM elements once (i.e., don't add the content to a `table` tag row-by-row)

These are but few recommendations; a complete overview of web performance is beyond the scope of this book. For more information, start reading the articles available at <https://developers.google.com/speed/>.

Understanding screen size and pixel density

When working with the mobile web, screen size and pixel density play a very important role and sometimes may overcomplicate things, because a pixel is not a pixel anymore (see the article by *Peter-Paul Kock* available at http://www.quirksmode.org/blog/archives/2010/04/a_pixel_is_not.html).

With the advent of high-pixel density devices, the pixel itself is a relative unit and fixed layouts have similar challenges to liquid layouts. When dealing with a native app (hybrid or not) on density-independent pixel displays, the virtual pixel-scaling algorithm adapts your app to the screen. For instance, a 100 px width image scales as follows according to the pixel density:

- ◆ 1x, 100px
- ◆ 1.5x, 150px
- ◆ 2x, 200px

The worst result you can get is that the images used in the UI are pixelated. In order to avoid this issue, you can simply embed in your app twice the resolution images, and then scale them accordingly to the device pixel ratio.

Time for action – scaling UI images according to pixel density

Follow these steps to scale the UI images of your app with a few lines of JavaScript:

1. Define a CSS class to mark all the images you want to handle according to the device pixel ratio.

```
<img class='highRes' />
```

2. Create a script able to get all the images marked with the previously defined class and change their width according to the pixel ratio.

```
function processImages() {  
  
    var pixelRatio = window.devicePixelRatio;  
    if(window.devicePixelRatio > 1) {  
        var matches = document.querySelectorAll("img.highRes");  
        for(var i = 0; i < matches.length; i++) {  
  
            matches[i].width = (matches[i].width / pixelRatio);  
  
        }  
    }  
}
```

3. Run the script when the page is loaded or when the `deviceready` event is fired. When using PhoneGap, it is strongly recommended that the code that needs to run at startup is executed immediately after the `deviceready` event.

```
addEventListener('deviceready', onDeviceReady);  
  
function onDeviceReady(evt) {  
  
    processImages();  
  
}
```

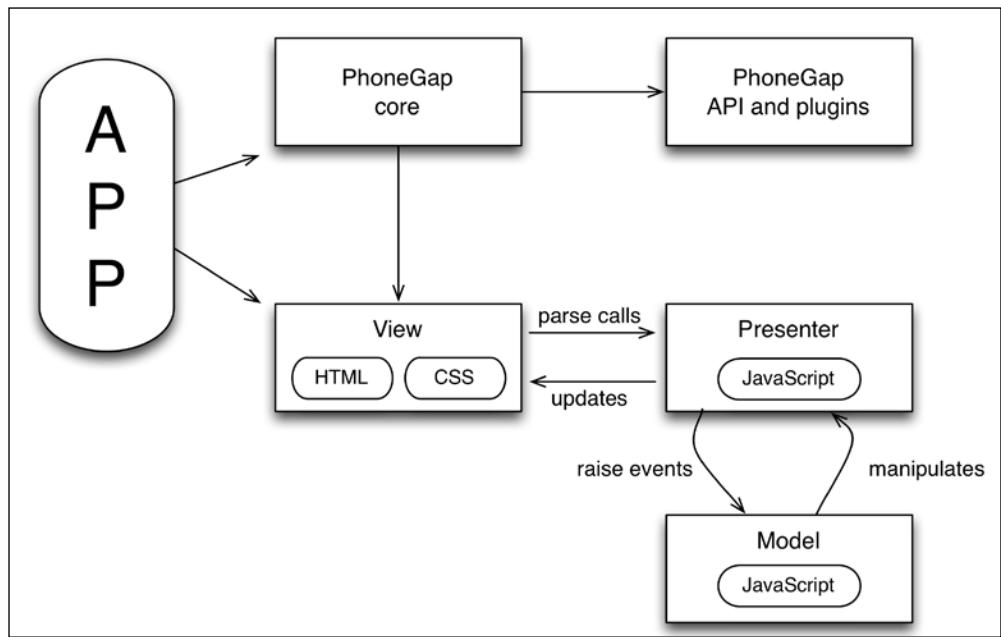
What just happened?

The images that compose the UI of the app are adapting themselves to the device pixel ratio with a really simple and fast script. In this way, the look and feel of the app will be consistent with the design provided to you.

Writing effective JavaScript

There are several resources online, and books that deeply discuss JavaScript best practices and effectiveness. Let's quickly review some practices to keep JavaScript maintainable and to keep the layers of your app strongly separated.

The following figure shows a rough representation of the components that compose a PhoneGap app; the view represents the UI of the app itself.



Loose coupling

It's important to keep HTML, CSS, and JavaScript separated in the app view, because dependencies between presentation and behavior can make the code of the app unmaintainable. This separation is also known as loose coupling of components, and it's achieved when you're able to make changes to a single component without making changes to other components.

Always separate your JavaScript from your HTML markup and be sure to separate JavaScript modules from each other. All JavaScript should be contained within the `<script>` tags, which ideally point to a linked JavaScript file. If the app needs additional HTML to render the UI, consider one of the following solutions:

- ◆ Load it from the server using an `XMLHttpRequest` object to retrieve additional markup templates
- ◆ Create a client-side template engine based upon regular expressions able to render markup pieces within slots that must be filled by JavaScript
- ◆ Use a complete client-side template engine such as HandleBarJS
<http://handlebarsjs.com/> or Mustache <http://mustache.github.io/> in order to handle expressions

Try to avoid assigning styles through the `style` property in JavaScript as it will help keep all the style information in your CSS. Instead, use classes to style an element as in the following example:

```
var myElement = document.querySelector('#id');
myElement.classList.add("cssClassYouNeed");
```

The practices described previously help you to ensure that there are fewer dependencies between CSS and JavaScript, and that each piece only manages its specific responsibilities.

Event handling best practice

As you know, JavaScript is only a means to add a behavior to a web page or to the view of your app. Most of the time the behavior of the page is defined through several event handlers that cooperate together to the definition of the behavior itself. When you handle an event it's a good habit to avoid defining logic in the event handler with complex conditional clauses. It means that the event handler should only call other methods or functions in order to make easier future changes. Remember always to remove an event listener when it's not needed anymore.

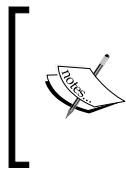
In order to clarify what I mean, let's explore a quick example that assumes the HTML page contains a DIV with the class `squareDiv` applied.

```
var APP = (function() {
    var init = function(event) {
        event.target.removeEventListener(event.type,
            arguments.callee);
        var target = document.querySelector('.squareDiv');
        target.addEventListener('click', openPopup);
    };
    var openPopup = function(event) {
        createWindow(event.x, event.y);
    };
    var createWindow = function(x, y) {
        console.log('Opening a window in ' + x + ' : ' + y);
    };
    document.addEventListener("deviceready", init, false);
}());
```

As you can see the `click` handler doesn't contain any logic; it simply calls another function passing around only the `event` object information needed by the other function.

Choosing web app templates

Developers typically have their own template libraries, built from scratch or commercial, to jumpstart their projects. In this section, I provide a short overview of some useful HTML templates you can evaluate as your blueprint, libraries, and frameworks you can easily integrate within your projects.



A library is essentially a set of functions that you can call, these days usually organized into classes or files. A framework embodies some abstract design, with more behavior built-in. *Martin Fowler* discusses further the difference between a library and a framework in his article available at <http://martinfowler.com/bliki/InversionOfControl.html>.



HTML5 Mobile Boilerplate

This is a very clean, mobile-friendly HTML template that includes an optimized Google Analytics snippet, placeholders for touch-based device icons, the library Zepto (a minimalist JavaScript library for modern browsers with a largely jQuery-compatible API), and the Modernizr feature detection library (a library that uses object detection techniques to discover if a feature is available before you use it, allowing for graceful degradation or progressive enhancement of web pages).

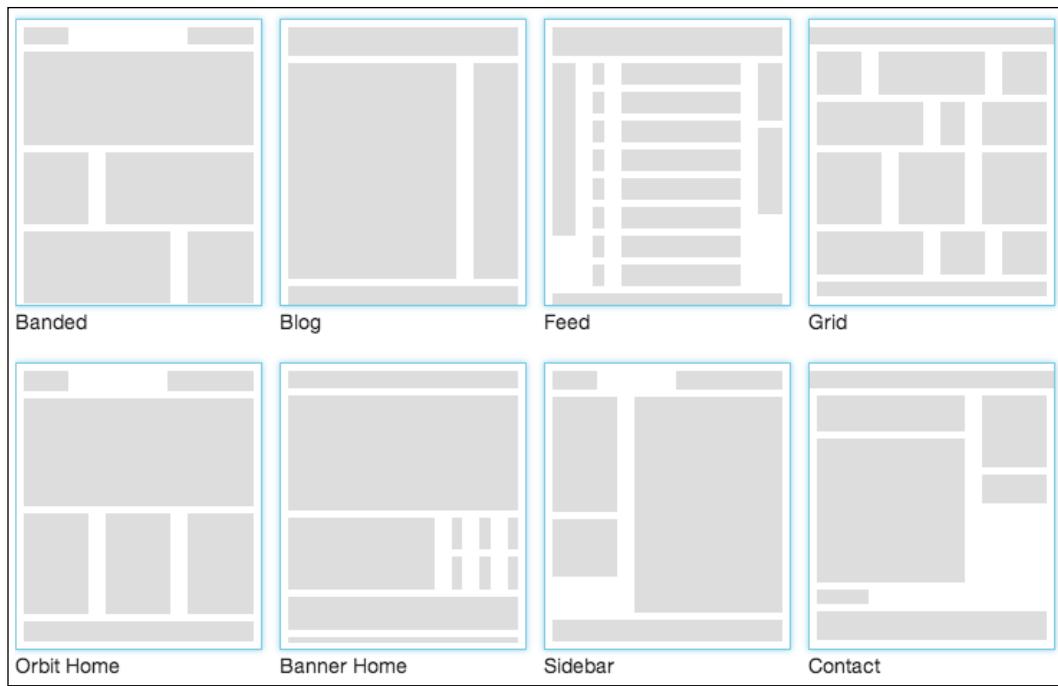
You can download the HTML5 Mobile Boilerplate template from the official website available at <http://html5boilerplate.com/mobile/>; for updates or to get involved follow the project on GitHub available at <https://github.com/h5bp/mobile-boilerplate>.

If you want to download a customized version of HTML5 Mobile Boilerplate, which enables you to select which templates to use, the JavaScript libraries to include, and so on, go to <http://www.initializr.com/>.

Foundation

You've probably already heard about responsive design, which is website design that responds to the device constraints of the person viewing it. It's a hot topic right now and the Foundation framework's most important feature is the responsiveness of its layout mechanics.

Furthermore, Foundation provides a good selection of templates to use for the most common sections of your app; you can choose the templates you want when downloading the framework from <http://foundation.zurb.com/download.php>.



The strengths of Foundation are:

- ◆ A 12-column, percentage-based grid with an arbitrary maximum width
- ◆ Image styles that disregard pixels—Foundation images are scaled by the grid to different widths
- ◆ UI and layout elements including common pieces such as typography and forms, as well as tabs, pagination, N-up grids, and more
- ◆ Mobile visibility classes—Foundation lets you very quickly hide and show elements on desktops, tablets, and phones

To keep up-to-date with Foundation and get the latest builds, follow the GitHub project available at <https://github.com/zurb/foundation>.

Bootstrap

Twitter Bootstrap is a free collection of tools for creating websites and web applications. It contains HTML- and CSS-based design templates for typography, forms, buttons, charts, navigation, and other interface components, as well as optional JavaScript extensions.

This project is one of the most popular on GitHub; it's very well organized and seems born to build apps. In fact, it includes basic CSS and HTML for creating Grids, Layouts, Typography, Tables, Forms, Navigation, Alerts, Popovers, and so on.

It's pretty easy to start working with Bootstrap because it uses jQuery. To download Bootstrap you can refer to the project download and customize page available at <http://twitter.github.com/bootstrap/customize.html>, if you want to download a template for Bootstrap you can refer to the already mentioned www.initializr.com website.

jQuery Mobile

The jQuery Mobile framework is a unified user-interface system across all popular mobile device platforms, built on the rock-solid jQuery and jQuery UI foundation. Its lightweight code is built with progressive enhancement and has a flexible, easily themeable design.

jQuery Mobile has broad support for the vast majority of all modern desktop, smartphone, tablet, and e-reader platforms. In addition, feature phones and older browsers are supported because of our progressive enhancement approach (for details go to <http://jquerymobile.com/gbs/>).

The main features of jQuery Mobile can be summarized as follows:

- ◆ Cross-platform, cross-device, and cross-browser
- ◆ UI optimized for touch devices
- ◆ Themeable and customizable design
- ◆ Usage of nonintrusive semantic HTML5 code only
- ◆ AJAX calls automatically load dynamic content
- ◆ Lightweight (12 KB compressed)
- ◆ Progressive enhancement
- ◆ Accessible

To download the last stable version and to keep up-to-date with the project, refer to the official website available at <http://jquerymobile.com> where you can find useful examples and a tool to create your own themes at <http://jquerymobile.com/themeroller/>.

Which is the right one?

It's pretty hard to say which is the best library to use. Most of the time, I have to say that it depends on the features you have to implement and even more so on the nature of your app. For instance if the app is just for mobiles, then you can decide to go lighter and use HTML5 Mobile Boilerplate; on the other hand, if the app is intended for the Web and mobiles, then a more sophisticated library can be the right choice.

Always keep in mind that your goal is to find a balance between built-in features and performance because mobile devices are far less powerful than a desktop.

Setting up your project using cordova-cli

When you create a new project it's good practice to have a folder structure that allows you to have a common code base for the view (i.e., HTML/CSS/JS) separated from specific platform folders and plugins.

The Apache Cordova community is vibrant and there is a project on GitHub named `cordova-cli` that you can use to create a folder structure that makes it easy to build, debug, and emulate cross-platform mobile apps. For more information and to get the source code, go to <https://github.com/apache/cordova-cli>.

Time for action – installing cordova-cli using npm

To install `cordova-cli` on your development machine and create a project enabling only the desired mobile platforms, follow these steps.



Node.js and npm are required in order to run `cordova-cli`. You can download and install the distribution for your platform from the official website available at <http://nodejs.org/download/> and the package manager (i.e., npm) available at <https://npmjs.org/>. Since Node.js Version 0.6.3 npm is bundled with the installer.

1. Install `cordova-cli` using the command-line tool:
`$ npm install -g cordova`
2. Change the permissions on the `cordova` folder in `node_modules` in order to run commands as root:
`$ sudo chown -R <username> /usr/local/lib/node_modules/cordova`



Once typed, this command can be easily recovered by pressing *Ctrl + R* and then typing your username when using the command-line tool.

3. Create a new project specifying the path and optionally a name and an ID using the `cordova` utility:

```
$ cordova create ~/the/path/to/your/source/code/ch03 HelloWorld
```

4. Switch to the project directory:

```
$ cd ~/the/path/to/your/source/code/ch03
```

5. Add the platform you want to target your mobile app to:

```
$ cordova platform add android  
$ cordova platform add ios
```

What just happened?

You just created a PhoneGap project that shares a common `www` folder in which you can add all the files needed to render the UI of your app. The folder structure looks like the following schema:

```
|-ch03  
|-.cordova  
|-.platforms  
|---android  
|---ios  
|----...  
|-.merges  
|-.plugins  
|-.www
```

Using the command-line tools from inside these folders you can easily build your app for all the added mobile platforms using the `cordova build` command, or debug and emulate it using the other commands discovered in the previous chapters.

Your first application – "Hello World"

As mentioned previously, a PhoneGap application is a "native-wrapped" web application through the web browser native widget that most of the mobile development SDKs provide as a part of their UI framework (this widget is known as **WebView**; for more information go to http://developer.apple.com/library/ios/#documentation/uikit/reference/UIWebView_Class/Reference/Reference.html and <http://developer.android.com/reference/android/webkit/WebView.html>).

In purely native applications, WebView controls are used to display HTML content either from a remote server or a local HTML package along with the native application in some way. The native "wrapper" application generated by PhoneGap loads the end developer's HTML pages into one of these WebView controls and displays the resulting HTML as the UI when the application is launched. One of the benefits of this approach is that since the app functions offline, there is lower latency for retrieving assets, and available bandwidth is less of a concern. When developing a PhoneGap-based app, it's enough to open the `index.html` file directly from your filesystem to get the optimal PhoneGap development workflow; even better, open the `index.html` file with Ripple and check the behavior of the app on several platforms.

Time for action – creating your first cross-platform app

To create your first PhoneGap app for Android and iOS, follow these steps:

1. Locate the `ch03` folder you just created or follow the steps described in the previous paragraph.

```
$ cd ~/the/path/to/your/source/code/ch03
```

2. Export the paths of the Android SDK required tools.

```
$ export PATH=$PATH:~/android-sdks/tools/  
$ export PATH=$PATH:~/android-sdks/platform-tools/
```

3. Run the `build` command to build your app for all the platforms you added after defining the project.

```
$ cordova build
```

4. Locate the CLI tools for iOS and debug the app; go to the `platforms/ios/cordova` folder inside your `project` directory and use the CLI tools discussed in the previous chapters.

```
$ cd ~/the/path/to/your/source/ch03/platforms/ios/cordova  
$ ./debug
```

5. Locate the CLI tools for Android and debug the app; go to the `platforms/ios/cordova` folder inside your `project` directory and run the `debug` command.

```
$ cd ~/the/path/to/your/source/ch03/platforms/android/cordova  
$ ./debug
```

What just happened?

As you can see, the result is that you can already deploy your app for multiple platforms using the same code base for the view (actually you used the one delivered with PhoneGap). You build and then debug the app on Android and iOS emulators or real devices.

Add interactivity to your app

Now that you are able to handle a single-code base for multiple platforms let's add interactivity to your app using one of the frameworks discussed at the beginning of this chapter.

Download a customized HTML5 boilerplate template, including Twitter Bootstrap from <http://www.initializr.com/>.

The template folder looks like the following tree in which you can find the CSS files, the index.html file, the JavaScript main file, all the required JavaScript libraries, and the images:

```
| -css  
| -img  
| -js  
| --vendor
```

Copy the files and folders you downloaded into the www folder inside your project and run the build and debug commands again. Your app now looks pretty different. Next, add some interactivity to it.

Time for action – programmatically opening a modal window using Bootstrap

In order open a popup in your PhoneGap app based on Bootstrap, you have to perform the following steps:

1. A framework like Bootstrap makes it easy to define a modal window; all you have to do is add the CSS classes `modal`, `hide`, and `fade` into a `div` tag and assign to it an ID (for example, `myModal`) that can be used as an anchor in the `href` attribute of the link you want to use to open the window.

```
<div class="modal hide fade in" id="myModal" tabindex="-1"  
role="dialog" aria-labelledby="myModalLabel"  
aria-hidden="true">  
    <div class="modal-header">  
        <!-- All the tags needed to render the modal window  
            here -->  
    </div>  
</div>
```

2. In order to use the Apache Cordova APIs, you have to include the `cordova-3.x.y.js` file in the `index.html` file; it is also good practice to include the main JavaScript file and initialize the app.

```
<script type="text/javascript" src="cordova.js"></script>
<script type="text/javascript" src="js/index.js"></script>
<script type="text/javascript">
    app.initialize();
</script>
```

3. The `index.js` file is the one created by the CLI distributed with Apache Cordova. Its contents are pretty simple because it listens for the `deviceready` event and then manipulates the DOM; the function that manipulates the DOM needs to be updated in order to add the `href` attribute to the `<a>` tag you want to use as the trigger for the modal window.

```
onDeviceReady: function() {

    var item = document.getElementById('modalLauncher');
    item.setAttribute('href', '#myModal');

}
```

`modalLauncher` is the ID assigned to the `<a>` tag contained in the original template.

4. In order to let the user zoom in and out of your app, you have to update the `viewport` meta tag defined in the template (it works only in Android). Particularly you have to change the values of the `maximum-scale` and `user-scalable` attributes.

```
<meta name="viewport" content="width=device-width,
    height=device-height, initial-scale=1, minimum-scale=1,
    maximum-scale=1.5, user-scalable=1">
```

In order to get the results due to these changes in Android, you have to make some changes to the main Java file of your app. First of all you have to import the `WebSettings` and the `ZoomDensity` classes. Then you have to enable the zoom by changing the settings immediately after the `super.loadUrl('')` method call.

```
import android.webkit.WebSettings;
import android.webkit.WebSettings.ZoomDensity;
.....
@Override
public void onCreate(Bundle savedInstanceState) {

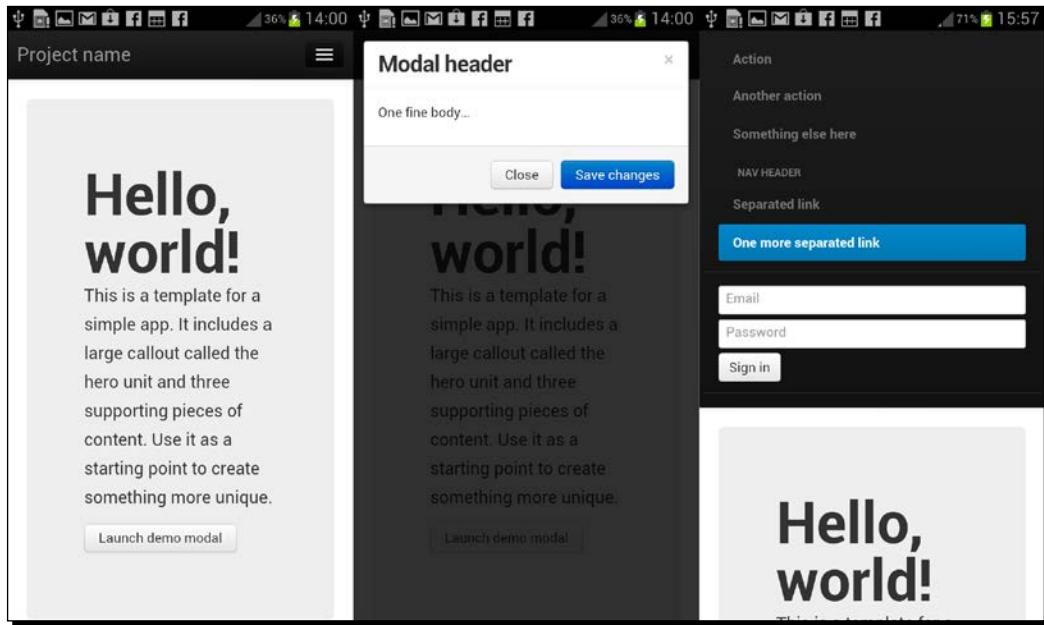
    super.onCreate(savedInstanceState);
```

```
super.loadUrl(Config.getStartUrl());  
  
WebSettings settings = appView.getSettings();  
settings.setBuiltInZoomControls(true);  
settings.setSupportZoom(true);  
settings.setDefaultZoom(ZoomDensity.MEDIUM);  
  
}
```

5. Build the app for all the platforms you added to the project and then, using the command-line tool, move to the platform-specific folder and run the `./emulate` or `./debug` commands in order to check the app inside the emulator or in a device.

What just happened?

The following screenshot is taken from a Samsung Galaxy S3:



With only a very small amount of code, your first PhoneGap app looks pretty good; it can handle multiple screen sizes and a complex menu.

You took an application that was developed and debugged in a desktop browser and rapidly created a native mobile project from that application.

The power of this approach is that you can avoid continuously debugging in the emulator or on a device, thus significantly reducing the development time. If you've worked in the past with Symbian, this is just a dream come true because the time waiting in front of the emulator is dramatically reduced.

Achieving a native look and feel on iOS

One of the biggest problems with the iOS platform is the publication of your app in the **Mac App Store (MAS)**. In fact, Apple is pretty scrupulous when checking whether an app can be added to the store.

One of the most important criteria to be admitted to the MAS is that the app provides an iOS user experience. For more details on the requirements an app must meet to make it into the MAS, go to the Apple Developer website available at <http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/UEBestPractices/UEBestPractices.html>. Also consider the following interesting approach discovered by *Lim Chee Aun* with regard to rendering the UI when working with HTML and CSS available at <http://cheeaun.com/blog/2012/03/how-i-built-hacker-news-mobile-web-app>.

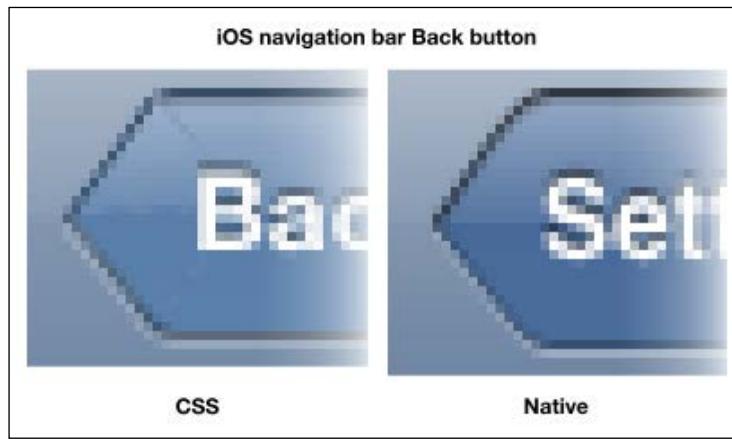
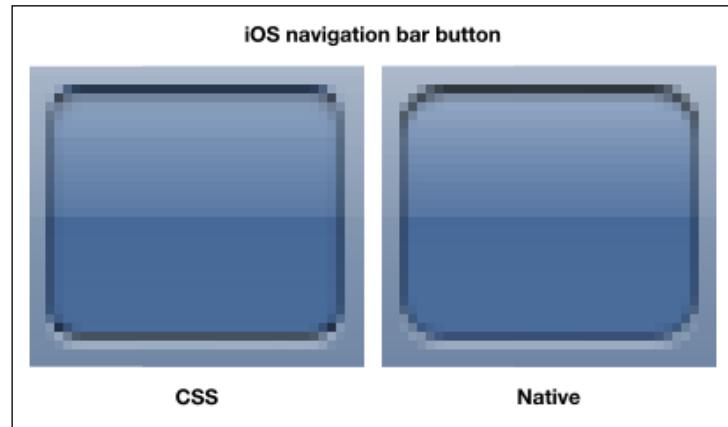
Time for action – setting up a native-like CSS for your app

In order to quickly create a native iOS-like UI of a PhoneGap app, perform the following steps:

- 1. Open the first ever developed iOS web app:** Change the user agent of your browser and navigate to the iPhone user guide available on the Apple Developer site <http://help.apple.com/iphone/>; you will load in the browser HTML and CSS that really look similar to a native app.
- 2. Grab the CSS and the images:** Open the Web Inspector and download all the images from the **Resource** panel; locate and grab the CSS file from the Network one (or pull the project from GitHub <https://github.com/cheeaun/hnmobile>).
- 3. Use the CSS and the images in your project:** Change the CSS file according to your needs and use it across the view of your app.

What just happened?

You created the building blocks of the styles of your app for iOS. As you can see from the following screenshots, the results look great and approximate a native app look and feel:



Have a go hero – improve the "Hello World" app

Review the instruction provided to add interactivity to the "Hello World" app and make the necessary changes to open a modal window when the app starts. Change the content of the popup to notify the user that the app will be ready soon and test the app on whatever device you have at hand (Android, BlackBerry, iOS, Windows Phone, and so on), verifying that the basic features implemented so far still work properly.

Pop quiz – getting started with mobile apps

Q1. How do you enable GPU-accelerated rendering in a PhoneGap-based app?

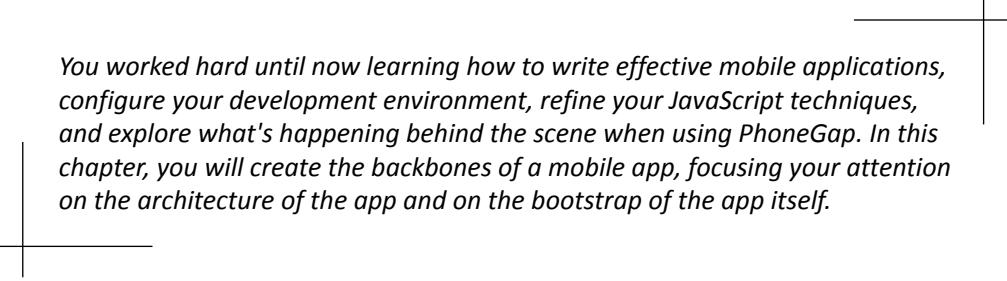
1. It's not possible at all.
2. Using the CSS rule `-webkit-transform: translate3d(0, 0, 0);` but it works only on iOS.
3. Using the CSS rule `-webkit-transform: translate3d(0, 0, 0).`
4. `transform: translate3d(0, 0, 0);` to make it working on all platforms.

Summary

In this chapter you learned how to build an interactive PhoneGap app and completed an overview of techniques and best practices that prepare you to build a more articulate app. In the next chapter you will learn how to develop the most interesting pieces of a cross-platform app.

4

Architecting Your Mobile App



You worked hard until now learning how to write effective mobile applications, configure your development environment, refine your JavaScript techniques, and explore what's happening behind the scene when using PhoneGap. In this chapter, you will create the backbones of a mobile app, focusing your attention on the architecture of the app and on the bootstrap of the app itself.

In this chapter you will:

- ◆ Learn how to configure your dev environment for developing hybrid apps rapidly
- ◆ Discover the basics to make your shell appealing and well suited for your specific needs
- ◆ Review JavaScript practices to write modular code
- ◆ Create a single-page HTML app to be deployed with PhoneGap
- ◆ Learn how to add a splash screen to your app

Fine-tuning your development environment

You probably will never stop fine-tuning your development environment, given the number of tools and scripts at your disposal, courtesy of the Web. Perhaps you're familiar with the GitHub community that collects and discusses dotfiles (<http://dotfiles.github.com/>), which include resources to accomplish common developer tasks using the command line. The command-line tool and your preferred text editor are the building blocks of your development workflow, so it's only natural that you invest some time to find solutions to your most common tasks that suit your needs.

Speeding up folder access with jump (OS X)

One of most common tasks you perform when working with the command-line tool is to change directories in order to move files, execute scripts, and so on. Usually you perform this task using the `$ cd` command, followed by entering a path, and accessing help by pressing the *Tab* key.

You already discovered how powerful Node.js can be and how it's easy to install additional modules using npm. Next, we will see how to install a useful Node.js module and how to make it available in the command-line tool.

Open your command-line tool and execute the following line to download the `jump` module:

```
$ sudo npm install jump -g
```

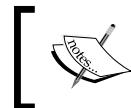
After npm has downloaded and installed the `jump` node module, add it as a source to your `.bash_profile` file.

```
jump >> ~/.bash_profile && source ~/.bash_profile
```

Now you are able to jump from one folder to the other of your operating system by typing the command and a query. Open your shell, type `j` and press *Enter*; you are now able to type some words (for example, `java`) and navigate to all the matching folders on your system using the arrow keys and pressing *Enter*.

```
j
> java
~/Documents/javascript
~/phonegap-phonegap-2.2/lib/blackberry/sample/lib/cordova.2.2.0/javascript
~/android-sdks/docs/guide/google/gcm/client-javadoc
~/android-sdks/docs/guide/google/gcm/server-javadoc
~/android-sdks/docs/reference/javax
```

Unfortunately `jump` works only on OSX, if you are a Windows user you can install Clink <https://code.google.com/p/clink/> or PYCmd <http://sourceforge.net/projects/pycmd/> in order to get a better shell and some features available in the Bash shell.



With the last stable version of Node.js (v.0.10.10 at the time of writing), `jump` is not working anymore but actually there is a fork to fix it on GitHub <https://github.com/GiorgioNatili/jump>.

Creating a server alias with serve

When working on a hybrid app, it is common to dynamically load scripts, templates, and so on. Most modern browsers don't allow you to load files locally because of security concerns. For this reason developers tend to keep their HTML/CSS/JS files on a local web server.

However, you have another option that allows you to run temporary web servers in any folder of your operating system.

Open the command-line tool and use npm to install the `serve` module:

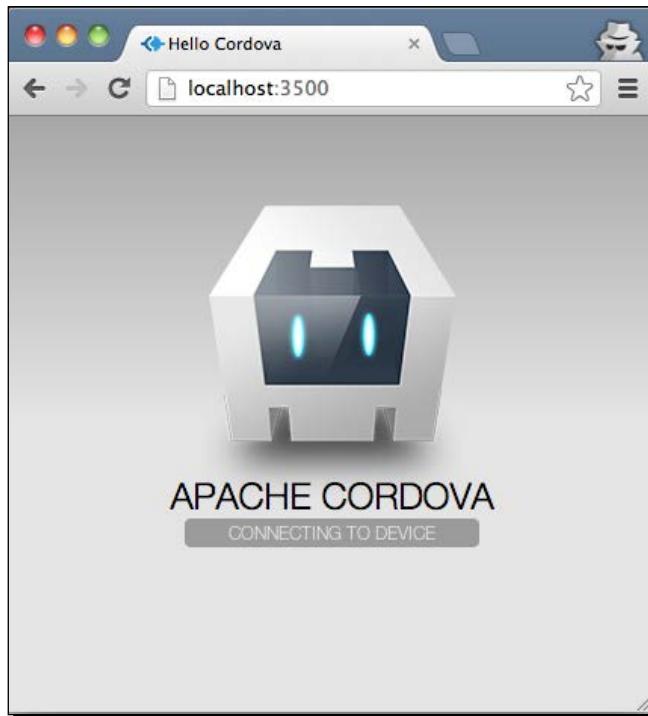
```
$ sudo npm install serve -g
```

Once the installation is completed you can run a server whenever you are specifying the port and other options (for a complete list of options, refer to the online guide available at <https://npmjs.org/package/serve>).

Using the command-line tool go to the `www` folder you created in *Chapter 3, Getting Started with Mobile Applications*, and run the following command:

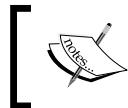
```
$ serve -p 3500
```

You can now open a browser tab and run your web page just like in a local web server.



Since Version 2.3, cordova-cli 2.3 includes a utility to launch a local web server for a platform's www directory on the given port. Use the following command to run the utility:

```
$ cordova serve android|blackberry|ios|windows -p xxxx
```



With Version 3.0 the `serve` command is deprecated; you will learn later how to use the `ripple` command that is a more robust and flexible testing/emulating-in-browser approach.



Customizing your shell with iTerm2 (OS X)

The more mobile apps you build, the more time you will spend with your command-line tool. Alas, the default configuration of the command-line tool is not exactly something that would make you want to work with this tool day in and day out. *Paul Irish* has been an advocate of using a more user-friendly shell with font colors and more.

Time for action – customizing the shell

Let's see how you can get a more comfortable shell by setting up font colors, getting additional info each time the shell is loaded, and so on.

1. Download and install iTerm2 from the official website <http://www.iterm2.com/#/section/home> (it's a replacement for the default Terminal OS X app with a rich set of features).
2. Explore the preferences by pressing *Command + ,* and specify your preferences.
3. Locate your `.bash_profile` file (usually it's in the `home` folder of your `user` directory) and open it with your preferred text editor.
4. Add the paths you typically use from the command-line tool (i.e., the Android SDKs).

```
PATH=$PATH:~/android-sdks/tools:/~/android-sdks/platform-tools/
```

5. Enable the Git bash completion commands.

```
source /scripts/.git-prompt.sh
```

6. Define the command-line tool colors you want to use.

```
BLACK=$(tput setaf 0)
RED=$(tput setaf 1)
GREEN=$(tput setaf 2)
YELLOW=$(tput setaf 3)
LIME_YELLOW=$(tput setaf 190)
POWDER_BLUE=$(tput setaf 153)
```

```
BLUE=$(tput setaf 4)
MAGENTA=$(tput setaf 5)
CYAN=$(tput setaf 6)
WHITE=$(tput setaf 7)
BRIGHT=$(tput bold)
NORMAL=$(tput sgr0)
BLINK=$(tput blink)
REVERSE=$(tput smso)
UNDERLINE=$(tput smul)
```

7. Set up the output of the default message you will get each time you run a command.

```
PS1='${MAGENTA}\u$WHITE} in ${GREEN}\w${WHITE}${MAGENTA}`__git_
ps1 " on %s"`${WHITE}\r\n`set_prefix`${NORMAL}${CYAN}\033[s\033[6
0C (`date "+%a, %b %d")\033[u${WHITE} \r\n'
```

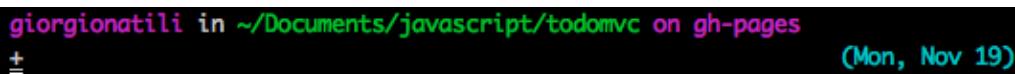
What just happened?

You took your first steps toward creating a better development environment in which you get colored strings instead of monochromatic messages in your shell. Furthermore, iTerm2 is rich with features such as full-screen view mode (*Command + enter*), search (*Command + F*), and autocomplete (*Command + ;*), among others.

If you get the -bash: __git_ps1: command not found error on OS X, open your command-line tool and manually install the `git-prompt.sh` file from GitHub.

 `curl -o ~/.git-prompt.sh \ https://raw.github.com/git/git/master/contrib/completion/git-prompt.sh`

Be sure to match the installation folder with the path you defined into your `.bash_profile` file.



giorgionatili in ~/Documents/javascript/todomvc on gh-pages
 (Mon, Nov 19)

Let LiveReload refresh pages for you (OS X)

It's pretty tedious having to continuously refresh pages when developing in the browser. Lucky for you, there is a cool OS X utility called **LiveReload** that refreshes the page for you each time it detects a change to a file.

To get your copy of LiveReload go to <http://livereload.com/>, and don't forget to install the Chrome extension while you're there.



LiveReload is also available for Windows. To enable it on a Windows computer, follow the same steps given in the following section.



Time for action – enabling Live Reload

In order to enable LiveReload in your project you have to perform the following steps:

1. Open LiveReload and add the project folder to the ones LiveReload is listening to.
2. Add the following script to the HTML document you want to refresh each time a change is detected (you can get the script from the LiveReload user interface):

```
<script>document.write('<script src="http://' +  
    (location.host || 'localhost').split(':')[0] +  
    ':35729/livereload.js?snipver=1"></' + 'script>')</script>
```
3. Open the page in a web browser and connect it to LiveReload using the Chrome extension.

What just happened?

You just enabled the automatic refresh of the page you are working on each time a file in the specified folder is modified. Now you can save time when working with CSS and JavaScript!



You can get similar results with Bracket; when you save your changes in Bracket, you can see the changes live in the browser. Depending on your preferences and operating system, you can use it to have a feature very similar to LiveReload.



Reviewing the JavaScript guidelines

In the rest of the book you will work on the source code of a real application (you can get the complete source code and development assets on GitHub <https://github.com/GiorgioNatali/itinero>). The following are some rules that I followed during the development of itinero:

All the constants will be named using only capital letters; if the constant name is made up of more than one word, an underscore will be used as a separator.

All the errors will be handled using custom exceptions, which makes it easier for you to debug the app because errors will be clearly identified by the exception itself.

Arrays and objects will always be instantiated using the literal form, in order to avoid any possible issues and to keep the code more readable.

Due to the fact that `null`, `undefined`, the empty string `' '`, and the number `0` are all `false` in Boolean expressions and the string `'0'`, the empty array `[]`, and the empty object `{}` are all `true` in Boolean expressions, the value checking will not only be performed against `null` but also using a data type check.

DOM manipulation will be performed only through logic-free templates.

Styles and formatting will not be performed through JavaScript. Across the app each specific technology (i.e., HTML, CSS, JavaScript) will be used for the appropriate task.

Each function will perform only one task and will be exposed carefully to the other components of the app.

Exploring the sample app

The itinero sample app is a social trip planner that lets you plan an affordable trip all around the world. Using itinero you can plan a trip defining the destination, how you want to travel there, where you want to sleep, and which kind of accommodation you are searching for. The app uses external services to get all the information you need, relying on the API of the most popular travelling social networks.

Once your trip is planned you can start to document it adding one image and one word per day. At the end of the trip you can share your experience on Facebook, Twitter, or Pinterest.

The following list summarizes the main features of the app and the PhoneGap API that will be used to implement these features:

- ◆ The app shows a splash screen to handle the startup time of the app (`navigator.splashScreen` API)
- ◆ The app's user-interface components are enabled according to the device connection status (`navigator.connection` API)
- ◆ The app lets the user add a contact to the planned trip so that each user can share trip details with friends (`navigator.contacts.*` API)

- ◆ The app allows users to get a picture or to select one from the gallery (`navigator.camera.* API`) and to create a relationship between a day of the trip and the image
- ◆ The app can apply some image effects using external libraries such as Aviary (<https://github.com/AviaryInc>) or the HTML5 canvas
- ◆ The app optionally adds geolocation information to each image associated with a trip (`navigator.geolocation.* API`)
- ◆ The app stores the information of each trip locally so that the user can browse the planned and completed trip info (W3C Web Storage API specification or Cordova's implementation for those devices that don't support the W3C specifications)
- ◆ The app lets users share a trip diary on the most popular social networks through text and image uploads (`window.requestFileSystem` and `FileSystem API`)
- ◆ The app user interface is available in four different languages and the user can set up the favorite language from the preference pane (`navigator.globalization.* API`)

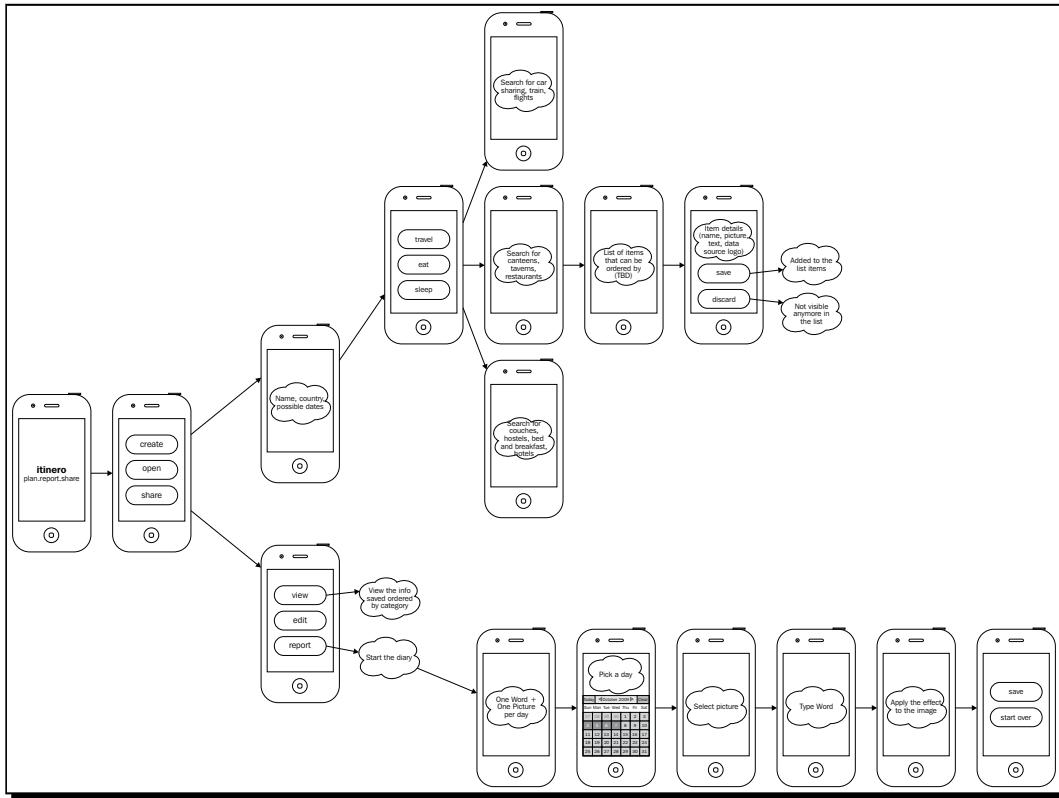
In short, the app covers most of the API offered by PhoneGap. You will also learn how to integrate social network plugins and how to write a simple plugin for your app.

The navigation flow

The navigation flow of the app is pretty simple: users can plan a new trip, open an existing one, or share a trip with friends on some of the most popular social networks.

When creating a new trip, users have to set up some mandatory information such as name, destination, origin, and duration. After entering this information, users can start to collect useful information such as low-cost travel options, cheap taverns, and so on, so that the plan becomes a way to understand how much the trip will cost. When the trip's plan is completed, users can share it with any of the contacts stored in the device's address book or they can use it as a diary during the trip, adding one picture and one word per day. Users can share on social networks only their trip diaries as a multimedia gallery.

The following diagram is a recap of the flow I just described:



The app architecture

The architecture of the itinero app has been defined keeping in mind that performance is of utmost importance on mobile devices. For this reason the only libraries used in the app are:

- ◆ Require.js (<http://requirejs.org/docs/download.html>)
- ◆ Mustache.js (<https://github.com/janl/mustache.js/>)
- ◆ Alice.js (<http://blackberry.github.com/Alice/>)

Require.js is pretty useful to handle dependencies between modules and to load the app module dynamically. Mustache.js is the JavaScript implementation of a standard used for creating templates in several programming languages. Alice.js is a very powerful tool to create animations.

There are tons of tutorials and blog posts on how to use Require.js and the documentation that comes with it is complete and well structured. Require.js lets the app load modules dynamically and helps speed up startup time and the app's overall performances.

Mustache.js is a "logic-less" template syntax; this means that it doesn't allow procedural statements (`if`, `else`, `for`, and so on). Also, Mustache templates are built inside HTML tags. This approach allows you to avoid as far as possible DOM manipulation in the app JavaScript modules.

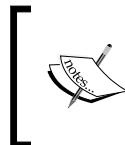
Alice.js is a very lightweight library that has no dependencies on other libraries. The purpose of this library is to create animations using the hardware accelerated capabilities of CSS3. Using Alice you can keep the "effect logic" outside the page and avoid third-party dependencies.

In order to keep modules clearly decoupled, the app core uses a `Mediator` to handle events that come from a module and to react according to the app business logic.

The Mediator pattern defines an object that encapsulates how a set of objects interact. This pattern is considered a behavioral pattern because it can alter the program's behavior. Your app is a program and it's made up by several files that contain the logic and computation of your app. When dealing with a large number of files, one of the issues to address is the communication between these files.

With the Mediator pattern, communication between objects is encapsulated with a mediator object. Objects communicate through the mediator instead of directly communicating with each other. This reduces the dependencies between communicating objects, thereby lowering the coupling.

The core is responsible to run, stop, and destroy a module, and even more to load the modules. The initialization of the app loads only the modules needed to run the app; all the other modules are loaded on demand. Another responsibility of the app core is to handle the errors. It's a good practice that each module and/or utilities involved in the app throws custom errors. In this way the app logic is very well enforced.



The final version of the core of the app also uses a Façade to enhance the module's privacy. In the following chapters you will spend more time reviewing the code needed to implement the Façade patterns in JavaScript.

The source code of the app is available on GitHub at <https://github.com/GiorgioNatali/itinero>. Please use the repository to open bugs or suggest improvements.

Communication between modules

Modules should know as little as possible about other modules. Instead of calling other modules directly, they should communicate between mediators. A mediator acts like a central event-handling system able to add or remove components that react to specific events.

The mediator exposes five methods to the core. Using these methods makes it possible to handle most of the app logic.

- ◆ `broadcast (eventName args, source)`: Invokes a function on a module that subscribes to the mediator for a specific event; the listener should register to an event formatted as `on + eventName` (for example, `onComplete`) that result then in the name of the function to be called
- ◆ `add (name, component, replaceDuplicate)`: Adds a component to the internal literal object used to store the references used by the `broadcast` function; optionally the function can replace a duplicate voiding any error
- ◆ `remove (name)`: Removes a component from the internal literal object used to store the references used by other functions of the mediator
- ◆ `get (name)`: Returns a component registered to the mediator by using its name
- ◆ `has (name)`: Returns a Boolean value accordingly to the existence of a specific named component in the mediator literal object

For a complete overview of the mediator and its features go to the GitHub repository <https://github.com/GiorgioNatali/PhoneGapGettingStarted/tree/master/ch04>.

In the following chapters you will refine this technique, working on a more refined events engine build with mobile in mind. Be sure to refer to the app GitHub repository to get the latest updated files.

The anatomy of a module

Generally speaking a software module is a piece of code that performs a well-defined task. Several modules can make up a larger module; the more these modules are loosely coupled, the more they are reusable across projects or across the same app. A good example of a module can be the emoticons rendering in Gmail. The emoticons, in fact, are used inside a mail message or inside a chat message; the rendering is then delegated to an independent piece of code reusable across the app.

In JavaScript there is a pretty famous pattern known as **Module Pattern** that uses closures to hide complex implementation and exposes a well-known API (.i.e. information hiding). The following code is an example of a module pattern with a self-executing function that returns an object exposing other functions:

```
var module = (function () {
    // private variables and functions
    var foo = 'bar';
    var performSomeOperation = function() {
        // Some code to execute here
    };
    var performOtherOperations = function() {
        // Some other code to execute here
    };
    return {
        doStuff: function(){
            performSomeOperation();
            performOtherOperations();
        }
    }
})();
```

The "complexity" of the functions `performSomeOperation` and `performOtherOperations` is transparent to end users of this module (i.e., other developers) that can simply be aware of the `doStuff` API.

A module defined using Require.js is similar to the one defined in the module pattern implementation; it looks like the following snippet:

```
define('moduleName', [
    // These are paths or paths alias that
    // have been configured in the app bootstrap

    'dependencyOne',      // path/to/the/dependency
    'dependencyTwo'       // path/to/the/dependency
], function(d1, d2){   // A reference to the injected dependencies

    // What is returned in the object it's used by other modules
    return {};
});
```

The use of modules in conjunction with Require.js helps you separate each component of the application, making it a "testable, reusable unit of cognition" defined through code.



Require.js also lets you dynamically load HTML templates. In order to do this you have to include in your project the `text!` Require.js plugin.



Building the app's core

The complete source code of the app you will work on in this book is available on GitHub <https://github.com/GiorgioNatali/itinero>. Each commit has a prefix that indicates the chapter to which the code is referred; the stable code of each chapter is available as a branch.

In this section, you will learn how to set up the application splash screen and how to handle the bootstrap time. In the course of reading this book you will gradually learn about all the pieces that make up the UI of the app. The first screen to implement is the following one:

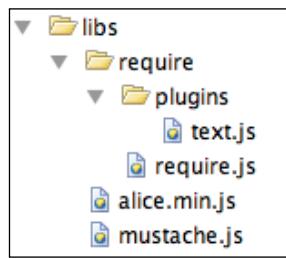


Create a new project using the cordova-cli utility introduced in *Chapter 3, Getting Started with Mobile Applications* (i.e., `$ cordova create ~/PhoneGapProjects/PGGettinStarted/itinero Itinero`), and add the folders needed to mirror the following organization to the www folder:

```
| -css  
| -img  
| -js  
|   ---libs  
|   ----require  
|   -----plugins  
|   ---model  
|   ---modules  
|   ---utils  
| -res  
|   ---icon  
|   ---screen  
| -spec  
| -tpl
```

The folders highlighted in bold are the ones I added manually to improve the organization of the source code. Folders in the `js` folder include: `libs` (stores all the libraries used in the app), `modules` (stores the app modules), and `utils` (stores the utilities needed to run the app). The `tpl` folder stores the Mustache templates that help you keep the layout separated from the business logic of the app.

The `libs` folder contains `require.js`, the `text.js` plugin of `require.js`, and a minified version of `alice.js` and `mustache.js`. You can download the files from the libraries websites; the `text!` Require.js plugin is available at <http://requirejs.org/docs/download.html#text>.



The parent folder `js` contains the files `main.js` and `index.js`. The first one is the one responsible for the configuration of Require.js, and the second one is the actual entry point of the app.

Bootstrap loader

It's a good habit to perform a deep analysis of the libraries and modules that are required to launch the app. This helps you reduce the app startup time and results in a better user experience.

Time for action – Require.js Bootstrap

In order to run and configure Require.js perform the following steps:

1. Add a script tag in the index.html file located in the www folder, specifying the path for the library and the main file to load when the library is ready as the value of the attribute data-main.

```
<script data-main="js/main"
       src="js/libs/require/require.js"></script>
```

Note that when using Require.js you have to refer to the files without the .js extension.

2. Create a file named main.js in the js folder and specify which are the paths of the main dependencies and other optional configurations (for a complete reference to the configuration options of Require.js refer to the online documentation available at <http://requirejs.org/docs/api.html#config>).

```
require.config({
    paths: {
        mustache: 'libs/mustache',
        alice: 'libs/alice.min',
        text: 'libs/require/plugins/text',
        templates: 'tpl'
    },
    waitSeconds: 10
});
```

With the waitSeconds option you can control how much time Require.js has to wait when loading dependencies. Due to the unpredictable speed of mobile devices I increased this option a little bit.

- 3.** Always require the entry module of the app in the `main.js` file and call a function on it (in our entry module the function is named `initialize`; you can find it in the `index.js` file).

```
require([
    'index'
], function(app){

    var appData = {

        appName:      'itinero',
        appSlogan:   'Plan.Report.Share',
        create:       'create your trip',
        open:         'open an existing trip',
        share:        'share your trip',
        year:         '2013',
        rights:       'All rights reserved',
        developer:   'Giorgio Natili',
        developerSite: 'webplatform.io'

    };

    app.initialize(appData);

});
```

What just happened?

The `main.js` file configured Require.js in order to easily access the libraries on which the app depends on and then loaded the `index.js` file. The script refers to the `index.js` file with the argument name `app` defined in a callback function and passes several data to the app core.

Mustache templates

I already discussed the importance of templates. HTML templates are a powerful means to achieve the separation of concerns discussed in the previous chapters.

The `index.html` contains only the tags needed to load the needed JavaScript libraries, the link to a CSS style, and a `<div>` tag that contains the parsed mustache template.

Time for action – creating a Mustache template

In order to create a Mustache HTML template, follow these steps:

1. Open your favorite editor, create an HTML file, and save it in the `tpl` folder.
2. When editing the HTML tags, add the markers you need to be replaced with data.

```
<section id='splashScreen'>

    <header>

        <small>
            {{rights}} | Copyright © {{year}}
            &nbsp;&nbsp;-&nbsp;&nbsp;
            <a href="{{developerSite}}>Credits</a>
        </small>

    </header>

    <hgroup>
        <h1 class='appName'>{{appName}}</h1>
        <h2 class='appSlogan blink'>{{appSlogan}}</h2>
    </hgroup>

    <nav>
        <ul class='mainNav'>
            <li id='createBtn'>
                <a href='create.html'>{{create}}</a>
            <li id='openBtn'>
                <a href='open.html'>{{open}}</a>
            <li id='shareBtn'>
                <a href='share.html'>{{share}}</a>
        </ul>
    </nav>

</section>
```

3. Write markers in order to match the names of the variables used in your data (for example, the object defined in the `main.js` file).

What just happened?

You just created a template to render the first screen of the app. The template markers match the `appData` object defined during the initialization.

Template initialization

The `index.js` file is the entry point of the app; it defines a function named `initialize`.

This function is responsible for the storage of the information needed by the Mustache template described in the previous paragraph.

Time for action – loading and parsing a template

In order to load the template, parse the data, and inject HTML in the view, make the following changes to the `index.js` file:

1. Require the template using the `Require.js` `text!` plugin and define a handler function called when the template is completely loaded.

```
require([
    'text!../tpl/splash-tpl.html'
], templatesReady);
```

2. Parse the data using `Mustache.js` and assign the result as the `innerHTML` value of `div` defined in the `index.html` file.

```
var html = mustache.to_html(tpl, appData);
document.querySelector('div.app').innerHTML = html;
```

What just happened?

The markup of the HTML has been created dynamically with some data that, in a future release of the app, will come from a web service.

The splash screen

If you search for a splash screen definition on Wikipedia you get the following definition:

"A splash screen is an image that appears while a game or program is loading. The term may also be used to describe an introduction page on a website. Splash screens cover the entire screen or simply a rectangle near the center of the screen. The splash screens of operating systems and some applications that expect to be run full-screen usually cover the entire screen."

PhoneGap includes a very powerful API that lets you control the splash screen of your app. More specifically, it allows you to set up a splash screen and then hide it after some time-expensive routines of the app have been completed.

Time for action – adding a splash screen to your app

In order to add a splash screen to an Android app built with PhoneGap you have to perform the following steps:

1. In order to prepare the splash screen images for Android, create several versions, one for each screen size, of the splash screen and save them in the `res/drawable-screensize` folder (please refer to the official documentation available at <http://developer.android.com/tools/help/draw9patch.html> in order to create 9-patch images).



The `res` folder can contain images, pictures, and so on. Each time a new resource is added to this folder it's automatically defined in the `R.java` class as `static int` values so that you can access it easily in your app. The drawable folders are the ones where you can put one image per folder, sized to match the indicated screen density.

2. In order to prepare the splash screen images for iOS, copy the images into the iOS project's `Resources/splash` directory. Only add the images for the devices you want to support, such as iPad or iPhone.
3. Open the Android project and add an XML configuration file named `splash_screen.xml` in the `xml` folder containing all the relevant information.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:background="@drawable/splash"/>
```

4. In the main Java file of the Android project, enable the splash screen and delay the loading of the `index.html` file.

```
super.setIntegerProperty("splashscreen", R.drawable.splash);
super.loadUrl(Config.getStartUrl());
```

5. In the `index.js` file, after the template loading and parsing, you can optionally add the JavaScript needed to hide the splash screen.

```
navigator.splashscreen.hide();
```



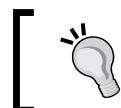
While setting a timeout in Java, if it isn't long enough, the splash screen will disappear before the app is done loading. Anyway, adding it also in the JavaScript files prevents you from having issues with other platforms or undocumented bugs.

- 6.** Open your command-line tool, navigate to the project root folder, and add the Splashscreen plugin.

```
$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-splashscreen.git
```

What just happened?

You defined a splash screen that will be visible only until your app is completely ready to run, and displays the UI to the user employing platform-specific settings and the Splashscreen API.



In iOS copy your splash screen images into the Resources/splash directory of your iOS project and then edit the Cordova.plist file so that the value of AutoHideSplashScreen is false.



Have a go hero – preparing additional app assets

Prepare the assets needed to define the splash screen of your app for all your target platforms. You can get a complete overview of the assets needed for each platform in the online references of each one.

Pop quiz – getting started with mobile apps

Q1. Which is the recommended cordova-cli command to run to emulate your app on a specific target platform?

1. The command serve.
2. The command emulate.
3. The command ripple.

Q2. What is the main advantage of using the module pattern in JavaScript?

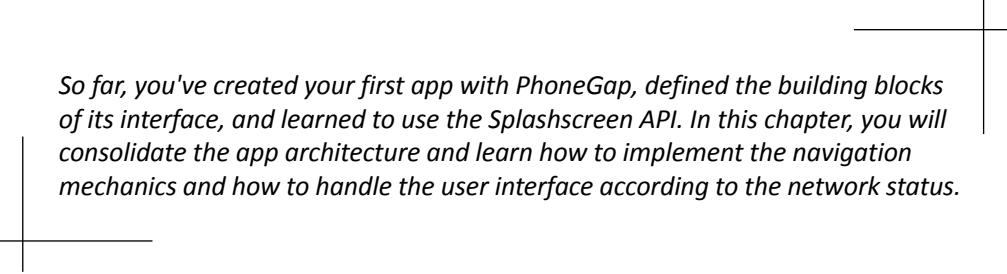
1. Encapsulation.
2. Code executes faster.
3. Hide unneeded complexities.

Summary

In this chapter you learned how to set up the first screen of your app handling the loading time with a graceful splash screen. In the next chapter, you will learn how to handle the navigation between the modules of the app and how to handle connectivity failures.

5

Improving the User Interface and Device Interaction



So far, you've created your first app with PhoneGap, defined the building blocks of its interface, and learned to use the Splashscreen API. In this chapter, you will consolidate the app architecture and learn how to implement the navigation mechanics and how to handle the user interface according to the network status.

In this chapter you will:

- ◆ Learn to compress your JavaScript and why this is especially important for a mobile app
- ◆ Learn more about template engines and how to compress template files
- ◆ Learn how to handle the user interface elements of your app when dealing with a retina display device
- ◆ Understand the PhoneGap app lifecycle and how to interact with the device API
- ◆ Create fluid, multiple views of your app using PhoneGap
- ◆ Learn how to create hardware-accelerated transitions between application screens
- ◆ Discover the PhoneGap connection APIs

Exploring JavaScript compression

In computer programming, you refer to the process of removing unnecessary characters from the source code files, and eventually concatenating them, as **file compression**. When dealing with web standards, you can compress any file type, including HTML, CSS, and JavaScript. The main goal of this process is to reduce the file size in order to speed up download time.

One of the benefits of compressing your source code when working with PhoneGap is performance improvement. When dealing with a mobile app, the files are compiled as a single file that eventually loads external data. However, when dealing with an app built using PhoneGap, the files, even if they are stored locally, have to be loaded in the browser (i.e. the **WebView**). Smaller files will be executed faster, so the end user will get a better experience with a more responsive user interface.

You may think that what really matters on a mobile device is the memory consumption, and that compression will not cause a great reduction of memory usage because the original file and the minified one are interpreted into the same code. However, there are compression tools that can affect the runtime performance as well. The following sections discuss three of the most popular compression tools that may help improve the performance of your app.

Google Closure Compiler

Google Closure is a set of open source tools built in order to help developers speed up the development process of modern web applications. The project consists of a JavaScript optimizer, a comprehensive JavaScript library, a server-side and client-side template engine, and a JavaScript style checker and style fixer. As a complete overview of Google Closure is beyond the scope of this book, I will discuss only the compiler.

One of the sentences that best describes the compiler comes from the online documentation:

"Instead of compiling from a source language to machine code, it compiles from JavaScript to better JavaScript."

You can use the compiler in one of the three ways:

- ◆ You can use it online at <http://closure-compiler.appspot.com/home>
- ◆ You can download a Java application from <http://closure-compiler.googlecode.com/files/compiler-latest.zip> and execute it through the command-line tool
- ◆ You can use the API provided by Google (see https://developers.google.com/closure/compiler/docs/gettingstarted_api)

When you open the online application, you can specify in the left pane the URLs of the scripts you want to compile, what kind of optimization you want to be applied to the output file, and if you want the output to be formatted for readability. On the left pane of the web application, you will get a report dealing with the original size and the optimized size of the file, the compiled code, a list of warnings, eventually some errors, and the POST data sent to the Closure Compiler APIs. The warnings provided refer to possible mistakes in the source code and to optimization that can be performed. For a reference to possible warning messages, go to <https://developers.google.com/closure/compiler/docs/error-ref>.

The screenshot shows the Google Closure Compiler interface. In the left pane, the source code for 'hello.js' is pasted:

```
// ==ClosureCompiler==
// @compilation_level ADVANCED_OPTIMIZATIONS
// @output_file_name default.js
// ==/ClosureCompiler==

// ADD YOUR CODE HERE
function hello(name) {
  alert('Hello, ' + name);
}
hello('New user');
```

In the right pane, the compilation results are shown:

Compilation was a success!

Original Size: 90 bytes (100 bytes gzipped)
Compiled Size: 25 bytes (45 bytes gzipped)
 Saved 72.22% off the original size
 (55.00% off the gzipped size)

The code may also be accessed at [default.js](#).

Compiled Code	Warnings
Errors	POST data

```
alert("Hello, New user");
```

At the bottom, there is a footer with links: ©2009 Google - [Terms of Service](#) - [Privacy Policy](#) - [Google Home](#)

If you prefer working with the command-line tool, you can download the compiler application and execute it, specifying the compiling options and the input and output files:

```
$ java -jar compiler.jar --compilation_level ADVANCED_OPTIMIZATIONS
--js hello.js
```

You get the same result when using the online tool; however, using the command line saves you an extra step: you don't have to upload the source code first.

When you use the advanced optimization be aware that the renaming process will be more aggressive, that the unused code will be removed, and that the body of the function calls will be replaced with the body of the function itself (this process is known as **function inlining**).

Time for action – compressing files using the Closure Compiler

Follow the given steps to get a compressed and optimized file using the Google Closure Compiler:

1. Download and unzip the Closure Compiler application available at <http://closure-compiler.googlecode.com/files/compiler-latest.zip>.
2. Open the command-line tool, move to the unzipped folder, and create a folder named `sample`.
3. In the new folder create three files: `index.html`, `text.js`, and `index.js`. You can use the following commands:

```
$ echo '<!DOCTYPE html><html><head></head><body></body></html>' > index.html
$ echo > index.js
$ echo > test.js
```
4. Open the `test.js` file and define a self-executing function. Within the body of the function declare two other functions and return one of them in order to be able to run this code from another JavaScript file (the purpose of the two functions is to mimic a real use case when some code is kept internal to a closure and some other is exposed through a returning object).

```
var test = (function() {

    var main = function() {
        alert('executing main');
        internal();
    };

    var internal = function() {
        alert('executing internal');
    };

    return {
        init: main
    }
}());
```

5. Open the `index.js` file and declare a variable in order to store the result of the self-executing function and make a call to the `init` function returned by the function itself.

```
var myTest = test.init();
```

6. Return to the command-line tool and run the compiler against the JavaScript files you just created.

```
$ java -jar closure-compiler/compiler.jar --compilation_level  
ADVANCED_OPTIMIZATIONS --js samples/test.js samples/index.js  
--js_output_file samples/app.js
```

7. Open the generated file and take a look at the source code; you will get the following JavaScript:

```
alert("executing main");alert("executing internal");
```

8. Insert the `script` tag in the HTML page and open it in a browser.

What just happened?

You discovered the power of the `ADVANCED_OPTIMIZATIONS` compilation level of the Closure Compiler. As you can see it's pretty aggressive. In fact if you run the same command using the files created in *Chapter 4, Architecting Your Mobile App*, you will not be able to run the application as intended. In short, be sure to check whether the `ADVANCED_OPTIMIZATIONS` option breaks your code; if so, you should consider using a different level of compression.

Next you will discover how to optimize and compress JavaScript modules using `require.js`.



In order to get an exhaustive guide to the Closure Compiler, refer to the online reference at <https://developers.google.com/closure/compiler/docs/api-tutorial3> or just type `$ java -jar compiler.jar --help` in your command-line tool.

UglifyJS2

The **UglifyJS** project became very popular when jQuery started to use it. To be used by one of the *de facto* standard JavaScript libraries resulted in a lot of feedback to the author, which in turn helped him fix a number of bugs.

The new version of the project, named **UglifyJS2**, is slower than the previous one but the overall compression results are much better and there are more advanced features such as multilevel source maps (basically it's a way to map a combined/minified file back to an unbuilt state) like in the Google Closure Compiler.

UglifyJS2 is distributed as a Node.js module. In order to install it you can proceed as with any other node module. Open the command-line tool and use **npm** to install the `uglify-js` module.

```
$ sudo npm install uglify-js -g
```

At this point, compressing your JavaScript files just got a lot easier.

Time for action – using UglifyJS with the Closure Compiler

Let's see how you can get a compressed version of the same files you worked on with the Google Closure Compiler.

1. Open your command-line tool and go to the `sample` folder created to test the Closure Compiler.
2. Type the following command in order to concatenate the JavaScript files and to run the UglifyJS2 compressor:

```
$ cat test.js index.js | uglifyjs --inline-script -o mytest.min.js
```

3. Open the generated file and take a look to the source code; you will get the following JavaScript:

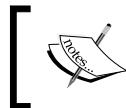
```
var test=function(){var main=function(){alert("executing main");internal();};var internal=function(){alert("executing internal")};return{init:main}}();var test=test.init();
```

4. Insert the `script` tag in the HTML page and open it in a browser.

What just happened?

You created a compressed version of two simple JavaScript files. As you can see the output is rather different from the one created with the Closure Compiler. One of the main features of UglifyJS2 is that the generated output doesn't break the source code.

For a complete reference, you can check the project page on GitHub at <https://github.com/mishoo/UglifyJS2>.



If you run UglifyJS2 in order to compress the files created in *Chapter 4, Architecting Your Mobile App*, you will be able to run the application as intended.

Optimization with Require.js

RequireJS includes an optimization tool named **r.js** that combines related scripts together into build layers and minifies them via UglifyJS or the Closure Compiler. The tool can be used through Node.js or through Java. When using the Closure Compiler it's mandatory to run the tool using Java.

The optimizer is better than using a plain concatenation script because it runs `require.js` as part of the optimization, so it knows how to load the plugins and all the dependencies of the JavaScript modules needed in your application.

For an exhaustive guide to `r.js`, refer to the readme file available on GitHub at <https://github.com/jrburke/r.js>.

Time for action – optimizing JavaScript with Require.js

Follow these steps to optimize the source code of your app using Node.js and Require.js:

1. Install the `require.js` module using **npm** from the command-line tool.

```
$ sudo npm install requirejs -g
```

2. Go to the root folder of the app you worked on in *Chapter 4, Architecting Your Mobile App*, create a file named `build.js`, and add to it the build process configuration info (i.e., the JavaScript folder, the paths to the library used in the project, the name of the main file of the app, and the output folder and filename).

```
({
  baseUrl: 'js/',
  paths: {
    mustache: 'libs/mustache',
    alice: 'libs/alice.min',
    text: 'libs/require/plugins/text'
  },
  name: 'main',
  out: 'js/main-built.js'
})
```

3. Open the command-line tool again and execute the following command in order to build the app:

```
$ r.js -o build.js
```

4. Open the `index.html` file and change the entry point of your app in the `script` tag in the header.

```
<script data-main="js/main-built"
       src="js/libs/require/require.js"></script>
```

5. Open the `index.html` file in a browser.

What just happened?

You created a compressed version of the app JavaScript files minified in a single file specifying the command-line options using a build file. The result is that the code of the app is now optimized using UglifyJS2 (the engine that works behind the scenes) and it still works perfectly. In order to get a complete overview of the build options, refer to the sample build file available on GitHub at <https://github.com/jrburke/r.js/blob/master/build/example.build.js>.



If you prefer to use the Closure Compiler to compress and optimize the app JavaScript files, you have to download the binaries of Rhino (an open-source implementation of JavaScript written entirely in Java) available at https://developer.mozilla.org/en-US/docs/Rhino/Download_Rhino, download `r.js` from the Require.js website at <http://requirejs.org/docs/download.html#rjs>, add the `optimize: 'closure'` option to the build file, and execute the following command:

```
$ java -classpath ~/rhinol_7R4/js.jar:~/compilers/
       closure-
       compiler/compiler.jar org.mozilla.javascript.tools.
       shell.Main r.js
       build.js
```

Where `classpath` refers to the full path to Rhino and the Closure Compiler.

Comparing compression tools

I covered three of the most popular compression tools. Each tool has its pros and cons. As always, the right tool for you is the one that best fits your needs. The following table summarizes the results in bytes you can get compressing `require.js 2.1.8` with the tools I just discussed:

File	Original size	Compressor	Size
<code>require.js</code>	82944	UglifyJS2	24576
		Google Closure	13312
		R.js	15360

As you can see, in this example, UglifyJS2 yields the best result but that is not always the case. If you run the same tests on the popular `raphael.js` library, you get instead the best result with Google Closure Compiler. The results vary depending on the source code writing style; for this reason there is no single best tool to use. I prefer `r.js` because it can run the compressor engine as well as handle the plugins and module dependencies very well.



Other compression tools you may consider include KJScompress, Bananascript, JSMin, ShrinkSafe, and YUI Compressor.

Using template engine compression

There are so many template engines available that discussing the pros and cons of all of them is beyond the scope of this book. Instead I will provide a quick overview of the most common template engines and how to compress a template file.

I strongly believe that there is no such thing as *the* best JavaScript template engine. Each time you work on a project you have to decide which is the right engine for the job at hand. For instance, **underscore.js** templates are fast and lightweight and if you want them already loaded in your app then it's a good option. When using jQuery, the natural choice seems to be **ICanHaz.js** because it returns each template as a jQuery object. When you need a more robust template engine, then Google Closure Templates could be a valid option.

In most cases **Mustache** completely fits the needs of an application because there is no logic in the templates and because the templates are language-agnostic, allowing you to reuse them between frontend and backend. There are several template engines based on mustache including **Handlebars.js**, **Hogan.js**, or **pistachio**.

Handlebars.js is a superset of mustache.js that adds some useful features such as block expressions, helpers, and more (refer to the online documentation for a complete overview: <http://handlebarsjs.com/>).

Hogan.js is a very powerful compiler for mustache templates from Twitter. Hogan.js is also delivered with a command-line utility that compiles all the `*.mustache` templates stored in a folder; the utility is located in the folder `hogan.js-template/bin`. More information about Hogan.js is available on GitHub at <https://github.com/twitter/hogan.js>.

Pistachio is not just another JavaScript template engine based on mustache.js. Its package contains a pure JavaScript compiler that compiles templates into self-contained JavaScript functions that can be used in every JavaScript environment.

In order to start using pistachio's compiler, you can install it as a Node.js module.

```
$ sudo npm install pistachio -g
```

Once installed, you can compile a template typing `pistachio` followed by the path to the file you want compile.

The interesting features of pistachio's compiler are the capability to compile a template as an AMD module or as a Common.js compatible module and the possibility to create the output as a jQuery object. A template compiled with pistachio is still dynamic and can be compressed even more using the Google Closure Compiler. For a complete reference on pistachio, go to <https://npmjs.org/package/pistachio>.

A compressed template speeds up your application rendering because you can cache it as a JavaScript function and avoid continuously loading and unloading it with an **AJAX** request (with some performance penalties involved) when the app is in use.



If you want to include multiple templates in a file, you can simply store them in `script` tags, assign to each tag an ID, and then use the `getElementById()` document object method and the `innerHTML` `HTMLElement` object property to render it.

```
<script type="text/x-mustache" id="tid...">
    /* mustache template */
</script>
```

Time for action – compiling a template using pistachio

Compile the template created in *Chapter 4, Architecting Your Mobile App*, and eventually compress it using pistachio. Follow the given steps:

1. Open your command-line tool and move to the `tpl` folders created in *Chapter 4, Architecting Your Mobile App*.
2. Type the pistachio command and specify the name of the output file and the file to compile.
`$ pistachio --out=splash-tpl.js splash-tpl.html`
3. Create a build file named, for instance, `template-build.js`, for the existing template to use when compressing the file with UglifyJS2, specifying the template name and the desired output filename.

```
({
  name: 'splash-tpl',
  out: 'splash-built.js'
})
```

- 4.** Run the r.js node module from the command-line tool.

```
$ r.js -o template-build.js
```

- 5.** Open the file and check its syntax and size.

What just happened?

You created a compressed version of the template file that is stored in a variable. You can now request it in the modules of the app and avoid any unnecessary XMLHttpRequest.

This technique is most beneficial when working with pretty big and complex templates. Throughout this book you will discover some advanced template caching techniques.

Handling a retina display user interface

Retina display is a brand name used by Apple to identify displays that have a pixel density equal to the "resolution" of the human eye retina. Although there is a debate about the actual resolution of the human eye retina, from a developer's point of view what matters is that the pixels per inch (**PPI**) or pixel density of the screen is high enough to prevent pixilation that is noticeable to the human eye (the amount of horizontal and vertical pixels are actually doubled).

Practically speaking, it's very important to handle the user interface elements in order to avoid a graphic element rendering itself in a pixelated way. If you use standard images on a retina display, they can appear small or blurry; in both cases the user interface will be seriously compromised.

When working with PhoneGap, you are creating a hybrid app based upon standards. For this reason you can handle different pixel densities using CSS media queries and CSS sprites. When using CSS sprites, you can combine an unlimited number of images into one and then use only the area of the image you need for a specific UI element. This technique improves significantly the performances of a website, reducing the number of HTTP requests to be done to render the page, but doesn't have any impact on performance when working on a PhoneGap app.

The real advantage of using CSS sprites is that you can handle more easily a high pixel density display (i.e., a retina display) loading a double size sprite. Using a combination of CSS media queries and the background size property should be sufficient for handling high-resolution displays.

Time for action – user interface elements and retina display

Here's how to prepare the app user interface elements to correctly render on a retina display device:

1. Open the Adobe Illustrator file you find in the `assets` folder on GitHub at <https://github.com/GiorgioNatali/PhoneGapGettingStarted> and create two different PNG files (the first one using the real size of the buttons and the second one doubling the size of the buttons) containing the buttons available in the first screen of the app.
2. Save the images using a name that clearly identifies which is the one to use with retina displays (i.e., `sprite.png` and `sprite@2x.png`).
3. Open the `index.css` CSS file and update the selectors previously created to define the buttons background in order to use a single sprite image.

```
.sectionNav #createBtn a{  
    background:url(..../img/sprite.png) no-repeat 0 0;  
}  
  
.sectionNav #openBtn a{  
    background:url(..../img/sprite.png) no-repeat -100px 0;  
    margin-left: 10px;  
}  
  
.sectionNav #shareBtn a{  
    background:url(..../img/sprite.png) no-repeat -200px 0;  
}
```

4. Using media queries detect whether the device has a retina display and eventually change the background sprite and translate the dimensions of the background to the ones of the normal sprite.

```
@media only screen and (-webkit-min-device-pixel-ratio: 1.5),  
only screen and (min-device-pixel-ratio: 1.5) {  
    .sectionNav #createBtn a,  
    .sectionNav #openBtn a,  
    .sectionNav #shareBtn a{  
        /* Reference the @2x Sprite */  
        background-image: url(..../img/sprite@2x.png);  
        /* Translate the @2x sprite's dimensions back to 1x */  
        background-size: 270px 95px;  
    }  
}
```

What just happened?

You implemented a user interface that is able to gracefully adapt itself to the most modern retina displays without changing the reference of the UI images element in the CSS file.

In order to understand how to create your sprite images you can use **spriteme**, a bookmarklet available at <http://spriteme.org/>.

Android implements a very interesting class named NinePatch to allow the creation of custom graphics that will scale the way that you define. The NinePatch class permits drawing a bitmap in nine sections with the four corners unscaled; the four edges are scaled in one axis, and the middle is scaled in both axes. For a guide to a tool to prepare NinePatch images, go to <https://developer.android.com/tools/help/draw9patch.html>; the tool is part of the Android SDK. Android has an automatic framework that chooses for you the best image for the actual device screen density (and a lot of other factors). For more info, refer to the online documentation at https://developer.android.com/guide/practices/screens_support.html.



In order to work with NinePatch images on iOS you can use the open source **Tortuga 22 NinePatch** library. For the complete Tortuga 22 NinePatch documentation, go to <http://blog.tortuga22.com/2010/05/31/announcing-tortuga-22-ninepatch/>.

Creating fluid, multiple app views

One of the strengths of PhoneGap is that the app UI and logic are built upon web standards. A mobile app is made up of several views that allow the user to interact with its core features. As for a web app, when working with PhoneGap, you can think of a view as a web page or as a fragment of a web page.

You can create multiple views in your app using different HTML pages or dynamically changing the markup of a single HTML page. The first approach is usually known as **multipage pattern**, the second one is known as **single page** pattern.

Generally speaking, the multipage pattern is best suited to applications that mostly comprise static content or with applications that rely mostly on the server for the business logic. When most of the content is static, you can package it using PhoneGap and deliver it as an app. When the business logic is defined on the server, you can think of the client as the presentation layer of your app and rely on a good mobile connection to make it available to users. In both cases your client-side code should be pretty simple and easy to maintain.

The multipage approach has some disadvantages. For instance, when the user navigates from one page to the next, the browser has to reload and parse all the JavaScript associated with the new page. Also, because the JavaScript code is reloaded, all application state is lost if your app does not use other techniques such as Local Storage or the HTML5 history state object to maintain it.

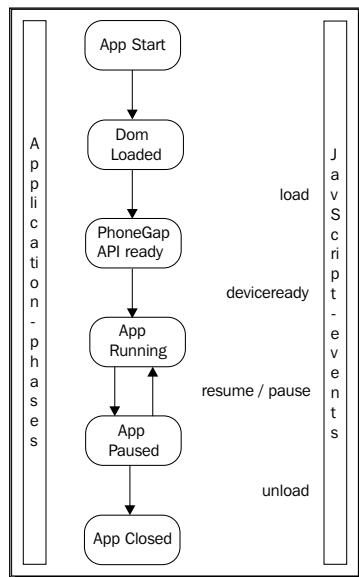
The single page pattern overcomes the disadvantages associated with the multipage approach. The PhoneGap and app JavaScript code is loaded just once, removing the need to pass application state from one page to the next. The disadvantage of this approach is an increased complexity of the JavaScript that contains most of the business logic and that is required to update the UI when navigation occurs.

The most important difference between the two patterns is that with the single page pattern the PhoneGap JavaScript bridge is loaded once. There's a noticeable pause when it's loaded due to the fact that the link between the JavaScript APIs and the native counterparts is created. When the app loads the PhoneGap JavaScript API once, the UI appears more responsive and the user experience is improved.

PhoneGap lifecycle events

When an app built with PhoneGap starts up, several things happen behind the scenes. The most relevant from a developer's point of view are the events `load`, `deviceready`, `resume`, `pause`, and `unload`.

The following figure resumes the events fired during the app lifecycle; each time an event is fired you can handle several aspects of your app.



When the app starts, the `load` event is fired when all of the content has finished loading. By registering an event listener for the `load` event you can handle the timeframe needed in order to have the PhoneGap JavaScript API completely loaded and available. However, this is not the most important event from a PhoneGap perspective. In fact the APIs are not ready until the app receives the `deviceready` event.

Once the `deviceready` event is fired, your app is really ready to start. This is the moment during which the user can start to use it and when you, as developer, can start to access all the device information.

When the app is up-and-running you can register two listeners in order to handle the `pause` and the `resume` events. The first one is fired when the app leaves the foreground and is suspended by the operating system; the second one is fired when the use of the app is resumed.

When the app exits, the `unload` event is fired. This is a very useful event since it allows you to save data in order to let the user start the app again in the same state he/she left it or to delete temporary files the app may use when accessing the `capture` API.

Time for action – accessing the device API

You already used the `deviceready` event in *Chapter 4, Architecting Your Mobile App*, to handle the bootstrap of our app. Use the device API to get information about the type of device you are running once the event is fired.

1. Open the command-line tool and create a new PhoneGap project using the Cordova-client utility you installed in *Chapter 3, Getting Started with Mobile Applications*.

```
$ cordova create ~/the/path/to/your/source DeviceApi
```

2. Move to the directory you just created, add the platforms you want to test on the device API and install the device API plugin.

```
$ cordova platform add android
$ cordova platform add ios
$ cordova plugin add https://git-wip-us.apache.org/repos/asf/
cordova-plugin-device.git
```

3. In the `www` folder inside the project folder you just created, open the `index.html` file and add `div` with `id` to render the device information gathered using the PhoneGap API.

```
<p id='deviceInfo'>Loading device properties...</p>
```

- 4.** Define a listener for the deviceready event in order to access all the supported API.

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

- 5.** Show the device information to the HTML page.

```
function onDeviceReady() {  
    var element = document.getElementById('deviceInfo');  
    element.innerHTML = 'Device Name: ' + device.name + '  
<br />' +  
                        'Device Cordova: ' + device.cordova +  
'<br />' +  
                        'Device Platform: ' + device.platform +  
'<br />' +  
                        'Device UUID: ' + device.uuid  
+ '<br />' +  
                        'Device Version: ' + device.version +  
'<br />';  
  
}
```

- 6.** In order to allow the app to access the device information, open the `app/res/xml/config.xml` file and verify that the device plugin is enabled (you should check if there is the `<plugin name="Device" value="org.apache.cordova.Device" />` XML node in the file).

- 7.** Update the permissions file `app/AndroidManifest.xml`, allowing the app to read the phone state adding the `<uses-permission android:name="android.permission.READ_PHONE_STATE" />` XML node to the file (no permissions changes are required for iOS).

- 8.** Build the project using the Cordova Client utility.

```
$ cordova build
```

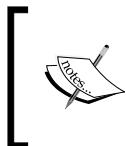
- 9.** Using the command-line tool navigate to the folder `platforms/PLATFORMNAME/cordova` and run the app on an emulator or real device (or run the command `$ cordova run PLATFORMNAME`).

```
$ ./run
```

What just happened?

You handled the `deviceready` event, accessing the relevant device information using the PhoneGap API. The `device` object you just used describes the device's hardware and software:

- ◆ `device.platform`: This gets the operating system name
- ◆ `device.uuid`: This gets the Universally Unique Identifier
- ◆ `device.version`: This gets the operating system version
- ◆ `device.cordova`: This gets the version of Cordova running on the device
- ◆ `device.model`: This gets the model name



Due to the rolling release model of PhoneGap, it's strongly suggested to always refer to the GitHub repository at <https://github.com/apache/cordova-docs/tree/master/docs/en/edge/cordova/device> in order to check whether a specific device API is going to be deprecated.

App views

The app you have been working on starting in *Chapter 4, Architecting Your Mobile App*, is made up of several screens. In order to keep this discussion focused, I will cover only how to handle the loading of the main view of each screen. Due to the nature of the app, you will follow the single page pattern to implement it; most of the features are designed to work offline.

Each screen is made up of a landing page and several subscreens; the landing page of each screen contains the name of the screen and an icon. The main sections of the app and their relevant features are:

- ◆ **Create**: Allows users to define a trip, select a trip mate, and start to plan the trip selecting a cheap travel service and places where to eat, sleep, and so on
- ◆ **Open**: Allows users to open a planned trip, edit the details, and write about one's trip experiences using text and images
- ◆ **Share**: Lets users share the trips they reported on the social networks

In order to render each section of the app, you will create specific templates. The template for the landing screen is pretty simple due to the layout, as you can see in the following screenshot:



In order to keep the code well organized, you will save the mustache template for each landing page in the `tpl` folder you already created. The structure of each template looks like the following code snippet:

```
<section id='create' class='gray-background'>
  <div class="centered-vertical">
    <h1 class='welcomeText'>{{sectionName}}</h1>
    <p class="section-mainIcon create-start shadow">
      {{sectionName}}
    </p>
  </div>
</section>
```

What's actually changed so far between each template is the `id` assigned to the section and the class to render the right icon (i.e., `create-start`). In the following chapters you will expand on each template to meet the layout of each section.

Time for action – creating the templates

Create all the templates needed to start implementing the navigation between the views of your app.

1. In your favorite editor create three new files named `create-tpl.html`, `open-tpl.html`, and `share-tpl.html` and save them in the `tpl` folder of your project.
2. Use the previous code snippet in order to populate the `create-tpl.html` template and save it.
3. From the same code, create the other two templates changing the `id` and the CSS class needed to render the right icon; the `id` needs to match the section name i.e., `open` and `share`) and the CSS class for the icon should be composed by the section name and the string `-start` (i.e., `open-start` and `share-start`).

What just happened?

You created the templates needed to render the welcome screen of each section of the app; the CSS needed to render the template are very simple and are available in the `www/css` folder of this chapter's source code, which you can find at <https://github.com/GiorgioNatali/itinero/tree/features/ch05>, you can grab the files using the following command:

```
$ git clone -b features/ch05 git://github.com/GiorgioNatali/itinero.git
```

Navigation between views

In order to speed up the learning process and to focus mostly on the PhoneGap features, I will discuss only the most important part of the source code.

The core of the app navigation happens in a file named `approuter.js` stored in the folder `js/routers`. This file is a `Require.js` module able to handle several events in order to load the selected route. A route is represented using another `require.js` module named `State.js` and stored in the folder `js/models`. The `State.js` module defines which controller, view, data, and templates are associated with each app screen.

```
;define('model/State', function() {  
  
    function State(data, controller, viewPath, template) {  
  
        this.data = data;  
        this.controller = controller;  
        this.viewPath = viewPath;  
        this.template = template || controller.toLowerCase() +  
            '-tpl.html';  
  
    }  
  
    return State;  
  
});
```

The module is pretty simple and assumes a default value for the `template` variable made up of the name of the controller and the `-tpl.html` string you previously used when saving the app screen templates.

The file `approuter.js` defines an object that stores in each property a route.

```
var routes = {  
    create: new State({sectionName: 'Create'}, 'Create', '/create'),  
    open: new State({sectionName: 'Open'}, 'Open', '/open'),  
    share: new State({sectionName: 'Share'}, 'Share', '/share')  
}
```

Each time you want to load a section of the app it's enough to send to the `approuter.js` module the name of the view; the module will handle the request through the `updateContent` function.

```
function updateContent(route) {  
  
    require(['controllers/' + route.controller],  
        function(controller) {  
  
            controller.init(route);  
            controller.start();  
  
        });  
  
}
```

The `updateContent` function requires a specific controller to call its `init` and `start` functions. In fact, it assumes each controller implements these two functions.

As already stated, when using the single page pattern, the code complexity increases. For this reason you are following here a very specific pattern known as model–view–controller.

"Model–view–controller (MVC) is a software architecture pattern which separates the representation of information from the user's interaction with it. The model consists of application data, business rules, logic, and functions. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible, such as a bar chart for management and a tabular view for accountants. The controller mediates input, converting it to commands for the model or view."

—Wikipedia

If you take a look at the source code that accompanies this chapter you can see that all the controllers are stored in the folder `js/controllers`, and each view is stored in a specific folder created inside the `views` folder (i.e., `views/create/CreateView.js`, `views/open/OpenView.js`, and so on). Each controller knows its **view** and interacts with it using the view-defined APIs. The view by contrast doesn't know the **controller** and eventually loads other views.

In order to have a very decoupled architecture, you can communicate with the router by a custom event. To do this you can create a custom event and then dispatch the event through the DOM in order to let the `approuter.js` handle it. In order to create a custom event, you can use the following syntax where `pushState` is the name of the event:

```
var event = document.createEvent("Event");
event.initEvent('pushState', true, true);

// Send custom data into the detail property
event.detail = {view: 'nameOftheViewToLoad'};
```



Unfortunately, at the time of writing the JavaScript `CustomEvent` constructor is not supported on any of the major mobile browsers. For more information refer to the online documentation at <https://developer.mozilla.org/en-US/docs/DOM/Event/CustomEvent>.

The `approuter.js` registers two event listeners in order to handle the navigation.

```
window.addEventListener('pushstate', onPushState);
document.addEventListener('backbutton', onBackButton);
```

The first one is the listener to the custom event mentioned previously and the second one is the listener for the back button.

The logic implemented into the two handlers is straightforward, because both use the `updateContent` function to render a previously rendered view or notify the app to reset the original view just in case there are no previous views to render.



The back button works only in Android; I am not handling here a specific iOS use case because these screens are only transition screens used to enhance the user experience.

The `SplashScreen.js` module defines the building blocks of the main navigation; in fact it registers a listener for the `click` event to each link available in the `splash-tpl.html` template and then dispatches the custom event discussed in this section. The event is handled in the `approuter.js` module that loads the required controller that on its side starts to render the view. Each view implements a transition in order to gracefully take the user to the next screen.

Using hardware-accelerated transitions

Much has been said about the use of **GPU (graphics processing unit)** hardware acceleration in smartphone and tablet web browsers. The general scheme is to offload tasks that would otherwise be calculated by the main CPU to the GPU in your computer's graphics adapter. (For a very detailed article to better understand hardware-accelerated transitions, go to <http://www.sencha.com/blog/understanding-hardware-acceleration-on-mobile-browsers>.)

GPU can accelerate:

- ◆ The general layout compositing
- ◆ All the CSS transitions
- ◆ The CSS 3D transformations
- ◆ All the canvas drawing operations

You can create smooth animations with the new CSS transitions pretty easily defining them in your stylesheets or you can rely on external libraries.

CSS transitions are supported in the latest versions of Firefox, Safari, and Chrome. They're supported in IE 10 and onwards. If CSS animations aren't supported in a given browser, then the properties will be applied instantly, gracefully degrading. There are several techniques to handle a CSS transition. I will use **Alice.js**, an interesting JavaScript library that allows you to execute hardware-accelerated transitions in your app.

Alice.js

Alice.js (A Lightweight Independent CSS Engine) is a JavaScript library that leverages hardware-accelerated capabilities of browsers in order to generate visual effects. One of the strengths of the library is that it doesn't rely on other libraries and that it's self-contained in a single JavaScript file. (For a complete reference and some interesting examples refer to the official website at <http://blackberry.github.com/Alice/demos/index.html>.)

Each time you want to create a transition with Alice.js, you have to set up a configuration object. This object varies depending on the effect or plugin you are using. However, some configuration properties are shared between all the effects and plugins, including:

- ◆ `elems`, the target element(s) or node
- ◆ `rotate`, the rotation angle in degrees
- ◆ `perspectiveOrigin`, the anchor point, which can be `top-left`, `top-center`, `top-right`, `center`, and so on, or the explicit coordinates in percent of the DIV's entire size, for example, `{x: 200, y: 200}`

- ◆ `duration`, the duration of the effect
- ◆ `timing`, the easing function as per standard CSS specs
- ◆ `delay`, how long before the animation starts
- ◆ `iteration`, the number of iterations
- ◆ `direction`, specifies whether the animation should be played in reverse mode
- ◆ `playstate`, either `running` or `paused`

In this way, it's possible to easily configure a CSS-based animation without any additional required know-how.

In *Chapter 3, Getting Started with Mobile Applications*, you already defined Alice.js in the bootstrap file, so in the view it's enough to define a configuration object and start a transition. The transition you have to implement is a simple slide from the right to the left in order to reveal the next screen.

```
// For brevity the configuration object is not complete
alice.plugins.cheshire({elems: ['splashScreen', 'create'], ...});
```

As you can see the code is pretty simple; for a complete overview please refer to the chapter's branch on GitHub.

Getting started with the PhoneGap APIs

Since the release of PhoneGap 3.0 all the APIs are available as external plugins; you will learn more about plugins later in this book. At this point, all you need to know is that treating each single API as a plugin allows you to compose a version of PhoneGap suited to your project needs.

Plugins are installed and removed using a tool called **Plugman**; from a developer's point of view it's just a command available in the cordova-cli utility.

```
$ cordova plugin add URL_TO_THE_GITHUB_REPO
```

where `URL_TO_THE_GITHUB_REPO` is the path to the plugin (i.e., API) repository.

The following list summarizes the currently available APIs and the command to run to add them to a PhoneGap project. If you want to remove a plugin, use the `$ plugins remove` command instead.

- ◆ Basic device information:
 - Device API: `$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-device.git`

- ◆ Network and battery status:
 - ❑ Network API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-network-information.git
 - ❑ Battery API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-battery-status.git
- ◆ Accelerometer, compass, and geolocation:
 - ❑ Device Motion API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-device-motion.git
 - ❑ Device Orientation API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-device-orientation.git
 - ❑ Geolocation API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-geolocation.git
- ◆ Camera, media capture, and media playback:
 - ❑ Camera API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-camera.git
 - ❑ Capture API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-media-capture.git
 - ❑ Media API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-media.git
- ◆ Access files on device or network:
 - ❑ File API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-file.git
 - ❑ File Transfer API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-file-transfer.git
- ◆ Notifications via dialog box or vibration:
 - ❑ Dialogs API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-dialogs.git
 - ❑ Vibration API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-vibration.git
- ◆ Contacts:
 - ❑ Contacts API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-contacts.git

- ◆ Globalization:
 - Globalization API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-globalization.git
- ◆ Splash screen:
 - Splashscreen API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-splashscreen.git
- ◆ In-app browser:
 - InAppBorwser API \$ cordova plugin add https://git-wip-us.apache.org/repos/asf/cordova-plugin-inappbrowser.git

Exploring the Connection API

When working on a mobile app it's important to know the status of the Internet connection. Using the PhoneGap APIs, you can handle this aspect easily; in fact there are two specific events and an object that allows you to get all the possible information. Once the `deviceready` event has been fired you can register a listener for the `online` event and another one for the `offline` event.

```
document.addEventListener("online", onDeviceOnline);
document.addEventListener("offline", onDeviceOffline);
```

In the `onDeviceOnline` handler, you can easily access the connection object stored in the `navigator.connection` property and access the device's cellular and Wi-Fi connection information using the `type` property. The possible values stored in the `type` property are:

- ◆ `Connection.UNKNOWN`
- ◆ `Connection.ETHERNET`
- ◆ `Connection.WIFI`
- ◆ `Connection.CELL_2G`
- ◆ `Connection.CELL_3G`
- ◆ `Connection.CELL_4G`
- ◆ `Connection.NONE`

In order to allow the app to access the `connection.type` information, you have to add an XML node to the `app/res/xml/config.xml` Android configuration file.

```
<plugin name="NetworkStatus"
        value="org.apache.cordova.NetworkManager" />
```

You also have to explicitly set up the permissions in the `app/AndroidManifest.xml` file.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE"
    />
```

In order to allow the app to access the connection information on iOS, it is enough to add the following XML node to the `config.xml` file:

```
<plugin name="NetworkStatus" value="CDVConnection" />
```

Pop quiz – getting started with mobile apps

Q.1. How do you handle retina displays in PhoneGap?

1. It's not possible at all.
2. With the cordova-cli utility.
3. Using CSS sprites and creating a 2x image file.

Q.2. In a PhoneGap app the `deviceready` event is fired when?

1. After the `load` event.
2. When the battery is fully charged.
3. It's never fired.

Have a go hero – create a network detection utility

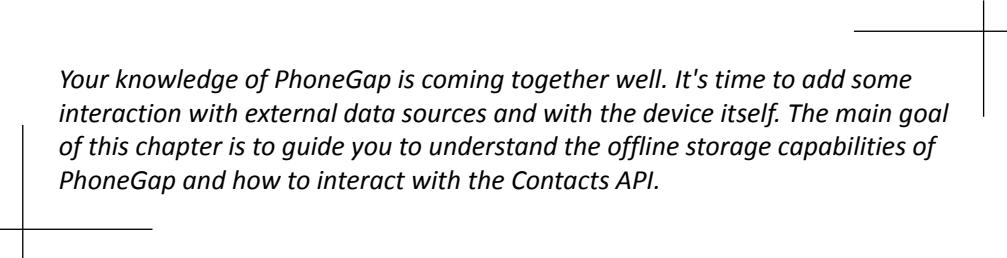
Review the information provided in order to add the API and what you learned about AMD JavaScript; create a `Require.js` module able to detect the network status and to dispatch the information to other modules.

Summary

In this chapter you learned how to optimize the source code of your app and how to handle the app lifecycle events; you also started to dig into the PhoneGap API. In the next chapter, you will learn how to load external data and how to manage the data in order to implement an offline strategy for the `itinero` app.

6

Using Device Storage and the Contacts API



Your knowledge of PhoneGap is coming together well. It's time to add some interaction with external data sources and with the device itself. The main goal of this chapter is to guide you to understand the offline storage capabilities of PhoneGap and how to interact with the Contacts API.

In this chapter you will:

- ◆ Learn how to read and write data on the device using the `localStorage` object
- ◆ Learn how to handle the storage on a local database considering the specific platform implementation
- ◆ Understand database storage limitations and learn how to handle them
- ◆ Get an overview of the PhoneGap Contacts API and its objects
- ◆ Learn how to use the PhoneGap Contacts API in order to read and filter the contacts stored on the device

Application data storage

Every application (desktop, web, or mobile) needs to store (and access) some data in order to work properly. How to store the data depends on the kind of information the application will work with and on the environment in which the application will run. A web application, for instance, can rely mostly on server storage because it runs on the Internet. Most advanced web applications implement an offline strategy and store some data locally on the user machine.

When working on an app, it's very important to consider how to handle the unstable data connection that typically is available on mobile devices. You have to design your app thinking of it as occasionally connected software. The term **occasionally connected computing** usually identifies a software architecture based on the idea that an end user should be able to continue working with an app even when temporarily disconnected or when a wireless connection is unavailable (for a deeper overview, refer to the online documentation about the mobile patent filed in 2007 at <http://www.google.com/patents/US20080014929>).

Modern web development offers several tools in order to let users interact with an application even when they're not connected:

- ◆ The **LocalStorage** API (<http://www.w3.org/TR/webstorage/#the-localstorage-attribute>)
- ◆ The **SessionStorage** API (<http://www.w3.org/TR/webstorage/#the-sessionstorage-attribute>)
- ◆ The **ApplicationCache** interface (<http://www.w3.org/TR/2011/WD-html5-20110525/offline.html>)
- ◆ The **IndexedDB** API (<http://www.w3.org/TR/IndexedDB/>)

All the modern mobile browsers let developers handle the `online` and `offline` events through the `navigator` object (https://developer.mozilla.org/en/docs/Online_and_offline_events).



It's important to keep in mind that the specifications about the `onLine` attribute report that this attribute is inherently unreliable because a computer can be connected to a network without having Internet access.

A complete overview of the previous API, events, and interface is beyond the scope of this book. In the following sections I will discuss only those that are most relevant for building a PhoneGap app: LocalStorage and IndexedDB.

Exploring the PhoneGap LocalStorage API

There are two main web storage types: local storage and session storage. The LocalStorage API is part of the **WebStorage** API defined by the W3C in order to provide a guideline for persistent data storage of key-value pair data in web clients. The LocalStorage API is designed to support data that needs to be available between sessions. In other words, the data your app saves when using the LocalStorage API will be available again the next time the app runs.

In order to access the LocalStorage API, you have to refer to the `window` object to its `localStorage` property. If you type the following snippet in your browser console, you can take a look at the methods and properties of the `localStorage` object:

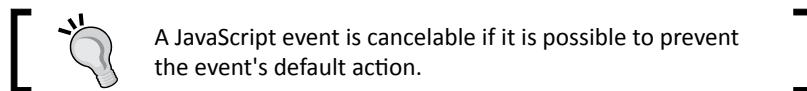
```
console.log(window.localStorage);
```

The following list summarizes the available methods and properties:

- ◆ `key`, returns the name of the key stored at a specific position; the `localStorage` object in fact allows you to access data by key or by index.
- ◆ `getItem`, returns the item identified by a key.
- ◆ `setItem`, saves data in a specific key (i.e., a string) of the `localStorage` object; the method needs a string as key and a value to store in the specific key.
- ◆ `removeItem`, removes the item identified by a key from the `localStorage`.
- ◆ `clear`, removes all of the key value pairs.
- ◆ `length`, returns the number of items stored in the `localStorage` object.

The `localStorage` object allows you to store only strings as key-value pairs. If you want to store more complex data, you have to use JSON or other string representations of the data you want to store. Each time the `localStorage` object is updated, a `StorageEvent` is fired. This event cannot be cancelled and contains the following properties:

- ◆ `key`, a string that represents the named key that was added, removed, or modified.
- ◆ `oldValue`, the previous value of the named key if it was updated or `null` if a new item was added.
- ◆ `newValue`, the new value of the named key or `null` if an item was removed.
- ◆ `url`, the address of the HTML page that called a method that triggered this change.



Keep in mind that storage events don't work for the same window or tab; they are fired only for other windows or tabs that use the same `localStorage` object.

The `localStorage` capabilities of PhoneGap allow you as a developer to write code as in the browser; it's the framework that handles the different platforms (Android, BlackBerry WebWorks OS 6.0 and higher, iOS, Windows Phone 7 and 8, and Tizen) on your behalf.

There are some drawbacks when using the `localStorage` object. For example, the API is synchronous and for this reason the app appears less responsive, because the time needed to access the `localStorage` object is greater than the one needed to access an object in memory. Also, as I already mentioned, complex data needs to be serialized and de-serialized, also impacting the responsiveness of the app. You can use JavaScript **WebWorker** to avoid any performance degradation, but the support depends on the platform browser implementation and not on PhoneGap.

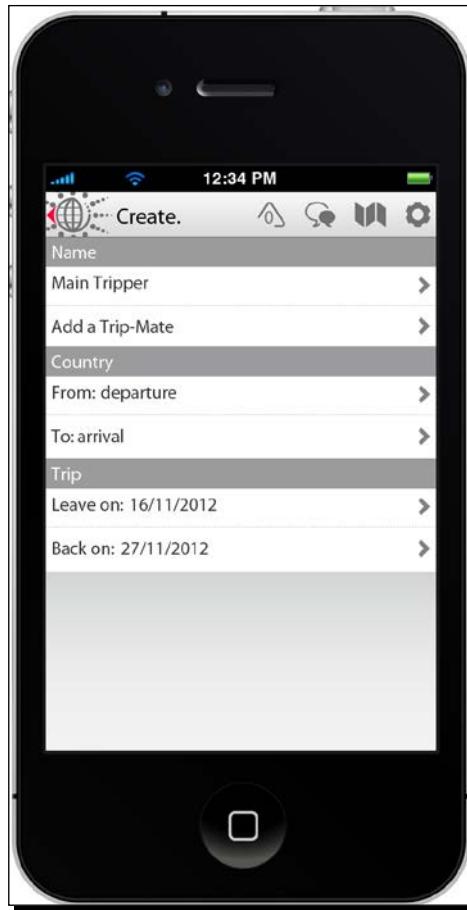
These drawbacks, however, should not prevent you from using the LocalStorage API because, as with most performance metrics, these performance hits really matter when you're performing the same operation multiple times in a row. More specifically, the performance degradation due to the time needed to access the `localStorage` object happens when different tabs/windows access the same object.

When the mobile app built on top of PhoneGap is running, it's almost impossible that the `localStorage` is accessed at the same time by other tabs/windows (you should have an issue only if your app starts to use several `InAppBrowser` instances in the `UIWebView` of the app).

Time for action – reading and writing data on the LocalStorage

The `itinero` app lets users create and plan a trip with friends. First, however, the user has to provide some basic information, including the names of "trip mates", the name of the country or countries to be visited, and the departure and return dates (as shown in the following screenshot).

Entering text using mobile phones is much more difficult than when using a desktop or laptop keyboard; for this reason it is always a good habit to provide the user with suggestions and autocomplete.



The following steps detail how to implement an autocomplete mechanism for text input.



In order to avoid long code listings, the steps don't refer to the *itinero* app specifically. If you want to take a look at the app implementation, refer to the GitHub branch at <https://github.com/GiorgioNatili/itinero/tree/features/ch06> or clone it in your development environment using the following command:

```
$ git clone -b features/ch06  
git://github.com/GiorgioNatili/itinero.git
```

1. Open a command-line tool and create a new PhoneGap app:

```
$ cordova create ~/the/path/to/your/source/ch06/storage  
SampleStorage
```

- 2.** Go to the app folder you just created and add the platform you want to use as the target for this sample (in my case Android):

```
$ cordova platform add android
```

- 3.** Go to the www folder the command-line tool just created, open the index.html file, and define a form with two input tags and a datalist tag:

```
<form>
  <input id='tripper' list = 'typedTrippers' type='text'
    placeholder='Main Tripper' />
  <datalist id='typedTrippers'>
    <!-- Dynamically content here -->
  </datalist>
  <input id='tripmate' type='text' placeholder='Add a Trip-
    Mate' />
</form>
```

- 4.** Open the index.js file you find in the www/js folder and inside the deviceready function add two listeners for the focusin and focusout event to the tripster input tag:

```
var tripster = document.querySelector('#tripper');
tripster.addEventListener('focusin', this.loadStoredData);
tripster.addEventListener('focusout', this.saveUserToStorage);
```

- 5.** The loadStoredData function is responsible for checking whether stored data exists and to add it to the typedTrippers datalist tag in order to allow the user to get useful suggestions each time the app runs. The function takes the event object as argument:

```
event.target.removeEventListener(event.type, arguments.callee);
```

```
var tripsterOptions = document.querySelector('#typedTrippers');
var fragment = document.createDocumentFragment();
```

```
var length = app.storage.length;
```

```
for (var i = 0; i <= length; i++) {
```

```
  var value = app.storage.getItem('tripper' + i);
  if (value) {
```

```
    var opt = document.createElement('option');
```

```
    opt.setAttribute('value', value);

    fragment.appendChild(opt);

}

}

tripperOptions.appendChild(fragment);
```

- 6.** The `saveUserToStorage` function checks whether the typed value is not already stored and, if it does not exist, adds the value to the `localStorage` object. The function takes the `event` object as argument:

```
var length = app.storage.length;
var newValue = event.target.value;
var toAdd = true;

for (var i = 0; i <= length; i++) {

    var value = app.storage.getItem('tripper' + i);
    if(value == newValue) {
        toAdd = false;
        break;
    }
}

if (toAdd) app.storage.setItem('tripper' + length, newValue);
```

- 7.** All the functions refer to the `app.storage` property. In order to access the `localStorage` object, the property contains a self-executing function that checks whether `localStorage` is supported:

```
(function() {
    var uid = new Date(),
    result;
    try {
        localStorage.setItem(uid, uid);
        result = localStorage.getItem(uid) == uid;
        localStorage.removeItem(uid);
        return result && localStorage;
    } catch(e) {}
}())
```

What just happened?

You saved persistent data on the device using the same JavaScript able to run in major modern browsers. Keep in mind that each platform stores this data in a different location and that this data may be cleared by the device. Depending on the platform, this data can be deleted when the app is closed or when the device is rebooted. For this reason I strongly encourage you to not use the `localStorage` object for storing crucial information.

Exploring the PhoneGap SQL storage

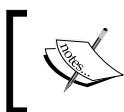
The client-side storage implementation in the different browsers on mobiles is pretty inconsistent right now. It's important to make an analysis of each browser implementation when working on a PhoneGap project because the app is rendered through `WebView` and on iOS this is the Objective-C `UIWebView` class, on Android it is `android.webkit.WebView`, and it differs on all other supported platforms. The web view simply exposes the underlying platform browser. For this reason it's important to know which client-side storage option is supported by the mobile browsers of your target platform.

The following table summarizes the storage support for the mobile version of the major browsers at the time of writing; the X sign indicates whether the feature is supported.

	Android browser	Firefox OS	iOS Safari	IE 10 mobile
IndexedDB	---	X	---	X
Web SQL	X	---	X	---

IndexedDB is a simple flat-file database with hierarchical key-value persistence and basic indexing. **Web SQL** is basically **SQLite** embedded in the browser. SQLite is a relational database contained in a small (~350 KB) library written in C that is used by software such as **Skype** or **Photoshop Lightroom**. The main difference between these storage options is that IndexedDB is a **NoSQL** database that lets you work with your JavaScript objects and indexes them based on your application needs, while Web SQL is a real, relational client-side database implementation. One of the advantages of using Web SQL is that you can share the same queries between that backend and the frontend. When running some tests you can see how much faster Web SQL can be.

DB type	Select single value by				
	Setup <input checked="" type="checkbox"/>	Insert <input checked="" type="checkbox"/>	Primary-Key <input checked="" type="checkbox"/>	Unique index <input checked="" type="checkbox"/>	
<input type="checkbox"/> LocalStorage					
<input checked="" type="checkbox"/> IndexedDB	✓ 0.05s	✓ 0.15s	✓ 0.18s	✓ 0.01s	
<input checked="" type="checkbox"/> Web SQL	✓ 0.02s	✓ 0.04s	✓ 0.01s	✓ 0.01s	



In order to run the same test on your machine, you can clone the GitHub repository at <https://github.com/scaljeri/indexeddb-vs-websql> and open the file `test.html` in your web browser.

The W3C dropped support for Web SQL on November 18, 2010, making IndexedDB the *de facto* standard. From a developer's point of view, IndexedDB may look like a huge step backward but it really isn't. For years, developers have stored data client-side using key-value pairs, and most of the time they use JSON querying the objects using **UQL** (refer to the online documentation of the Unstructured Query Language available at <http://unqlspec.org/>). For these reasons IndexedDB should be seen as the natural evolution of client-side storage.

PhoneGap provides storage API based on the deprecated Web SQL database specification. When a device already offers support for Web SQL, the app will use it. For devices that don't have storage support the app will use the PhoneGap one. As a developer you will not notice any difference, because you will not have to change any line of code.

Database storage with PhoneGap

In order to start working with a `Database` object it's enough to store in a variable what's returned by the `openDatabase` method:

```
var size = (1024 * 1024 * 2);
var db = window.openDatabase("test", "1.0", "Test DB", size);
```

The `openDatabase` method accepts the following four (self-explanatory) arguments:

- ◆ The **name** of the database file
- ◆ The **version** of the database
- ◆ The **display name** of the database
- ◆ The default **size** of the database

Keep in mind that an application can query the version number of the database in order to understand whether an upgrade to the database schema is required. The `openDatabase` method returns a reference to the currently open database.

In the previous snippet I allocated 2 MB of space in bytes. When allocating space, consider that mobile devices may have limitations on the size of the database they can support. For example, iOS only allows up to 5 MB for web-based applications; the same is true when using PhoneGap as a wrapper.

The returned database object between the others exposes two methods that take from one to three arguments: `transaction()` and `readTransaction()`. The main difference is that the `readTransaction()` method has to be used in read-only mode. The arguments that can be passed to these methods are:

- ◆ A function to execute one or more SQL statements
- ◆ A function to handle an exception raised by the app when opening the database
- ◆ A function to handle the successful opening of the database

Note that the `transaction()` and `readTransaction()` asynchronous methods are the only methods in the PhoneGap framework that want the failure handler function before the success one. Also, the success handler is the only one that doesn't receive any argument; the other handlers receive an `SQLTransaction` object.

The `SQLTransaction` object exposes the `executeSql` method. Using this method it's possible to run several SQL statements and pass some parameters to the statements to handle successful SQL query execution and SQL errors.



You can use several tokens in order to pass parameters to a SQL statement. For a complete overview of the available tokens, refer to the online documentation at http://www.sqlite.org/lang_expr.html.

The success handler receives two arguments: the first one is a reference to the transaction itself and the second one is a `SQLResultSet` object that contains the information, and optionally the results, of the executed query.



The `insertId` property of the `SQLResultSet` object returns an `Exception: DOMException` value when performing a `SELECT` statement. You can safely ignore it because it doesn't affect the app.

Time for action – populating a local database

In order to reinforce what you just learned, you will create a new local database, add a table to it, and write and read some data. In order to avoid a long code listing, I removed trivial portions from the code such as HTML tags injecting.

1. Return to the project you created for the previous example, open the `index.html` file, and add to the markup two `select` tags in order to let the user pick up a departure country and an arrival one. Make the two `select` tags required and set up a default option to explain to the user what to do:

```
<select id='departure' required>
  <option disabled selected value=''>Select a country</option>
```

```

<!-- All the other options here -->
</select>
<select id='arrival' required>
  <option disabled selected value=''>Select a country</option>
  <!-- All the other options here -->
</select>
```

2. Add the Contacts API plugin using the command line:

```
$ cordova plugins add https://git-wip-us.apache.org/repos/asf/cordova-plugin-contacts.git
```

3. Open the `index.js` file, create a function named `prepareDatabase`, and call it from the `deviceready` handler.

4. In the body of the function `prepareDatabase`, create Version 0.5 of a new database named *itinero* and assign 2 MB as size limit:

```
var size = (1024 * 1024 * 2);
database = window.openDatabase('itinero', '0.5', 'Trips',
                               size);
```

5. Always in the body of the `prepareDatabase` function, create a transaction in order to create a table. Don't forget to define the error and success handlers.

```
this.database.transaction(this.createTables,
  this.onCreateTableError, this.onCreateTableSuccess);
```

6. Define the function `createTables` and in the body, using the transaction object (i.e., `transactionObj`) you get as an argument; execute the SQL statement needed in order to create the `trips` table:

```
transactionObj.executeSql('CREATE TABLE IF NOT EXISTS trips
  (id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, tripper
  TEXT, tripMate TEXT, departure INTEGER, arrival,
  departureDate DATE, arrivalDate DATE)');
```

7. In order to save the data to the local database it is enough to register a listener for the `submit` event on the `body` tag; that way you will never lose the listener even when the form is reloaded via AJAX:

```
document.querySelector('body').addEventListener('submit',
  this.onSaveData, true);
```

8. Define the `onSaveData` function and in its body recover the data stored in the form fields and call another function passing the values as arguments. Prevent the `submit` event to propagate with its default behavior:

```
var tripper = document.querySelector('#tripper'),
  tripMate = document.querySelector('#tripmate'),
  departure = document.querySelector('#departure'),
```

```
arrival = document.querySelector('#arrival');

app.addTrip(tripper.value, tripMate.value, departure.value,
arrival.value);

evt.stopPropagation();
evt.preventDefault();
```



The `stopPropagation` method stops the event from bubbling up the event chain; the `preventDefault` method prevents the default action the browser makes on that event.

- 9.** The body of the `addTrip` function is the part of the code you use to store data to the local database. The function receives four arguments used as parameters within the `executeSql` method of the transaction object; the function used in order to run the SQL query is defined inline as the first argument of the `transaction` method of the database object.

```
this.database.transaction(function(transactionObj) {

    var sql = 'INSERT INTO trips (tripper, tripMate, departure,
        arrival) VALUES (?, ?, ?, ?);';
    transactionObj.executeSql(sql, [tripper, tripMate,
        departure, arrival]);
}, this.onTransactionFault, this.renderData);
```

- 10.** The handler for the successful execution of the insert the `renderData` function is also responsible for reading the data from the database and rendering it. The function defined inline as an argument of the `transaction` method of the `database` object is responsible to run the query and to render the data. In fact the successful handler is able to access the returned data using the `result` argument and its `item(index)` method.

```
this.database.transaction(function(transactionObj) {

    var sql = 'SELECT * FROM trips;';
    transactionObj.executeSql(sql, [], function(txObj, result){

        var limit = result.rows.length;
        for(i = 0; i < limit; i++){

            var obj = result.rows.item(i);
            console.log(obj.id + ' - ' + obj.tripper);
        }
    }, null);

}, this.onTransactionFault, null);
```

What just happened?

You created a local database in order to store and recover information relevant for your app and its users. The database will be automatically removed when the app is uninstalled.

Database limitations

There are some limitations to be aware of when using the WebSQL implementation of PhoneGap (refer to the SQLite documentation for a complete overview:

<http://www.sqlite.org/limits.html>). These limitations are not related to the framework itself but are due to the web view implementation of each target platform.

The limit you can easily hit when working on an app is the size limit of the database file. For example, on WebKit it varies depending on the operating system from 5 MB to 25 MB. Another limitation you can find is that, since iOS 5.1, both `localStorage` and `Web SQL` databases have been moved to the folder `~/Library/Caches` from `~/Library/WebKit`. Actually this change means that the information stored is not backed up anymore and that it can be arbitrarily deleted by the operating system when more space is needed (for more information about iOS data management, refer to the Apple Developer guide at <https://developer.apple.com/technologies/ios/data-management.html>).

In order to avoid the issues described, you can use the `Sqlite` Plugin available on GitHub for Android (<https://github.com/brodyspark/PhoneGap-sqlitePlugin-Android>) and iOS (<https://github.com/brodyspark/PhoneGap-sqlitePlugin-iOS>).



You will learn more about PhoneGap plugins throughout this book; for now it suffices to know that a plugin is typically a combination of HTML/CSS/JavaScript and native code used in order to extend the PhoneGap capabilities.

The main advantages you get when using this plugin are that you can keep the SQLite database in a user data location that is known and can be reconfigured, that there are no more size limits, and that the database can be encrypted using **SQLcipher** (for a complete reference refer to the online documentation at <http://sqlcipher.net/documentation/>).

From a developer's point of view there is no change in the API except the prefix; it means that instead of opening a database accessing the `window` object:

```
window.openDatabase();
```

You have to refer to the plugin:

```
sqlitePlugin.openDatabase();
```

The Contacts API

You can easily access the contact information stored on a device using the PhoneGap API. The **Contacts API** is an implementation of the W3C's Pick Contacts Intent API (an intent that enables access to a user's address book service from inside a web application). You can read more about the W3C specifications at <http://www.w3.org/TR/contacts-api/>).

In order to allow your app to access and manipulate the contacts stored on the device, you have to update the specific platform configuration files. The following table lists the information you need to add to the configuration files for Android, iOS, and Windows Phone apps. For a complete list of all the supported platforms refer to the online documentation at http://docs.phonegap.com/en/edge/cordova_contacts_contacts.md.html#Contacts.

Platform	Configuration file	Content
Android	app/res/xml/config.xml	<plugin name="Contacts" value="org.apache.cordova.ContactManager" />
	app/AndroidManifest.xml	<uses-permission android:name="android.permission.GET_ACCOUNTS" /> <uses-permission android:name="android.permission.READ_CONTACTS" /> <uses-permission android:name="android.permission.WRITE_CONTACTS" />
iOS	config.xml	<plugin name="Contacts" value="CDVContacts" />
Windows Phone	Properties/WPAppManifest.xml	<Capabilities> <Capability Name="ID_CAP_CONTACTS" /> </Capabilities>

In order to start to interact with the device contacts, you can use the `create` or the `find` methods defined in the `contacts` object stored in the navigator one:

```
var contact = navigator.contacts.create(properties);
navigator.contacts.find(contactFields, contactSuccess, contactError,
contactFindOptions);
```

In order to better understand how these methods work, let's explore the most relevant objects that are involved with them: `Contact`, `ContactName`, `ContactField`, `ContactFindOptions`, and `ContactError`.

The ContactName object

The `ContactName` object is used in the PhoneGap framework in order to store all the details of a contact name. The object is stored in the property `name` of the object `Contact`.

The properties of the `ContactName` object are all strings and are self-explanatory:

- ◆ `formatted` represents the complete name of the contact.
- ◆ `familyName` represents the contact's family name.
- ◆ `givenName` represents the contact's given name.
- ◆ `middleName` represents the contact's middle name.
- ◆ `honorablePrefix` represents the contact's prefix (e.g., Mr. or Dr.).
- ◆ `honorableSuffix` represents the contact's suffix (e.g., Esq.).



Until Version 2.4 the `formatted` property of the `ContactName` object is partially supported on Android and iOS. The property will return several data concatenated; it also doesn't store anything on the device.



The ContactField object

The `ContactField` object is a generic object used in the PhoneGap framework in order to represent a field of the `Contact` object. The generic nature of this object makes it reusable across several fields.

The properties of the `ContactField` object are:

- ◆ `type`: a string that represents the type of field; possible values are `home`, `work`, `mobile`, and so on.
- ◆ `value`: a string representing the value of the field such as a phone number or e-mail address.
- ◆ `pref`: a Boolean value that indicates whether in a specific field the user preferred value is returned.



When the `ContactField` object is used in the `photos` property of the `Contact` object the `type` property represents the type of a returned image (for example, a URL or a Base64-encoded string).

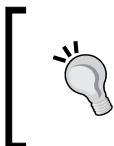


The ContactAddress object

The ContactAddress object is the object stored in the `addresses` property of the `Contact` object. The `addresses` property is an array so that multiple addresses can be associated with each contact.

The properties of the `ContactAddress` object are:

- ◆ `pref`: a Boolean that indicates whether the returned `ContactAddress` is the preferred value of the user for the `ContactAddress` object
- ◆ `type`: a string that indicates what type of address is stored in the `ContactAddress` object (for example, home, and office)
- ◆ `formatted`: a string that represents the complete address formatted for display
- ◆ `streetAddress`: a string that represents the complete street address
- ◆ `locality`: a string that represents the city or locality that is part of the `ContactAddress` object
- ◆ `region`: a string representing the state or region that is part of the `ContactAddress` object
- ◆ `postalCode`: a string that represents the zip code or postal code associated with the locality stored in the `ContactAddress` object
- ◆ `country`: a string representing the name of the country stored in the `ContactAddress` object



There is a limitation in Android 2.x (i.e., the `pref` property is not supported) and several on Android 1.x. Also, iOS does not support the `formatted` property. Always refer to the online documentation to verify the status of the support for the `ContactAddress` object.



The ContactOrganization object

The `ContactOrganization` object represents all the details of a company, organization, and so on that the stored `Contact` belongs to. The object is stored in the array contained in the `organizations` property of the `Contact` object.

The properties of the `ContactOrganization` object are:

- ◆ `pref`: a Boolean that indicates if the returned `ContactOrganization` object is the preferred value of the user for the `ContactOrganization` object
- ◆ `type`: a string that indicates what type of address is stored in the `ContactOrganization` object (for example, work and other)

- ◆ name: a string that represents the name of the organization stored in the ContactOrganization object
- ◆ department: a string that represents the department of the organization the contact works for
- ◆ title: a string that represents the contact's title in the organization

The properties `name`, `department`, and `title` are partially supported on iOS; the `pref` and `type` properties are badly supported on Android 1.x and Android 2.x.

The Contact object

The Contact object represents all the details of a contact stored in the device database. A Contact object can be saved, removed, and copied from the device contact database using the methods `save`, `remove`, and `clone` defined on the object itself. The `save` and `remove` methods accept two arguments in order to handle the success and the failure of the save or remove operation:

```
var contact = navigator.contacts.create({'displayName': 'Giorgio'});  
  
contact.save(onContactSaved, onContactSavedError);  
contact.remove(onContactRemoved, onContactRemovedError);
```

The error handlers receive the same `ContactError` object as an argument; the success handlers receive the saved contact or a snapshot of the current database when a contact is successfully removed.

The `ContactError` object contains in the `code` property the information about the occurred error. The values that can be returned are:

- ◆ `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`

When creating a new `Contact` object, you can define one by one the contact properties or pass them as an object when calling the `create` method. The properties of the `Contact` object are:

- ◆ `id`: a string used as a globally unique identifier
- ◆ `displayName`: a string that represents the name of the `Contact` object for display to end users
- ◆ `name`: an object containing all the information of a contact name; the object used to store this information is the `ContactName`
- ◆ `nickname`: a string that represents the casual name of a contact
- ◆ `phoneNumbers`: an array of all the contact's phone numbers; the array items are instances of the `ContactField` object
- ◆ `emails`: an array of all the contact's e-mail addresses; the array items are instances of the `ContactField` object
- ◆ `addresses`: an array of all the contact's addresses; the array items are instances of the `ContactAddresses` object
- ◆ `ims`: an array of all the contact's instant messages accounts; the array items are instances of the `ContactField` object
- ◆ `organizations`: an array of all the organizations the contact belongs to; the array items are instances of the `ContactOrganization` object
- ◆ `birthday`: a `Date` object that represents the birthday of the contact
- ◆ `note`: a string that represents a note about the contact
- ◆ `photos`: an array of all the contact's photos; the array items are instances of the `ContactField` object
- ◆ `categories`: an array of all the contact's defined categories; the array items are instances of the `ContactField` object
- ◆ `urls`: an array of all the web pages associated with the contact; the array items are instances of the `ContactField` object

The `Contact` object properties are not fully supported across all platforms. In fact operating system fragmentation makes it difficult to handle this information consistently.

For instance, the properties `name`, `nickname`, `birthday`, `photos`, `categories`, and `urls` are not supported on Android 1.x. Likewise the `categories` property is supported neither on Android 2.x. nor on iOS.

On iOS, the items returned in the photos array contain a URL that points to the app's temporary folder. This means that this content is deleted when the app exits and that you have to handle it if you want the user to find the app in the same status he/she left it. The displayName property is not supported on iOS and will be returned as null unless there is no ContactName defined. If the ContactName object is defined, then a composite name or nickname is returned.

Filtering contact data

I already mentioned the `find` method available on the `navigator.contacts` object. Using this method an app can find one or multiple contacts in the device's contact database. The `find` method accepts four arguments. The first one is an array that contains the name of the fields of the `Contact` object that have to be returned. The second and the third are the success and error handlers. The last one represents the filtering options you want to apply to the current research.

In order to apply a filter to the current search, it is enough to instantiate a new `ContactFindOptions` object and populate the `filter` and `multiple` properties. The `filter` property is a case-insensitive string that will act as a filter on the fields of the `Contact` objects returned by the `find` method. The `multiple` property is `false` by default and is the one to use in order to receive multiple `Contact` objects in the success handler.

Time for action – filtering device contacts

Let's see how to filter the device contacts when the user starts to type a contact name in a text input. Actually the filter will return only the contacts that have one or more e-mail addresses in order to let the user add a trip mate as a contact with whom to share the trip details.

1. Return to the project you created in the previous example and open the `index.html` file, and add another `datalist` tag for the `tripMate` input field:

```
<datalist id='typedTripmates'>
    <!-- Dynamically content here -->
</datalist>
```

2. Open the `index.js` file you already worked on and add a listener for the `input` event on the `tripMate` input field:

```
var tripMate = document.querySelector('#tripmate'),
    tripMate.addEventListener('input', onTripmateChange);
```

- 3.** Define the `onTripmateChange` function and in its body clear and start an interval (stored in the `delayedFind` variable) in order to call the `find` method only when the user stops typing:

```
if(delayedFind){  
    clearInterval(delayedFind);  
    app.delayedFind = null;  
}  
  
if(event.target.value.length > 3)  
    app.delayedFind = setInterval(filterContacts, 300,  
        event.target.value);
```

- 4.** The `filterContacts` function invoked by `setInterval` is the one in which you define how to run the `find` method and which are the filters. You are passing an argument to that function named `text`:

```
clearInterval(delayedFind);  
delayedFind = null;  
  
var options = new ContactFindOptions();  
options.filter = text;  
options.multiple = true;  
  
var fields = ['displayName', 'emails'];  
navigator.contacts.find(fields, onContactFindSuccess,  
    onContactFindError, options);
```

- 5.** The function `onContactFindSuccess`, that is the success handler defined in the `find` method, receives an array of `Contact` objects as argument. It's in the body of the function that you will filter the result checking that the e-mail array exists and eventually add them to `datalist typedTripmates`:

```
var typedTripmates = document.querySelector('#typedTripmates'),  
    fragment = document.createDocumentFragment(),  
    opt;  
  
for (var i in contacts){  
  
    if(contacts[i].emails){  
  
        opt = document.createElement('option');  
        opt.setAttribute('value', contacts[i].displayName);  
        fragment.appendChild(opt);  
  
    }  
}  
  
typedTripmates.appendChild(fragment);
```

What just happened?

You filtered the contact database based on the options defined by the `ContactFindOptions` object and you refined the result using the API provided by the PhoneGap framework.

Pop quiz – getting started with mobile apps

1. How can you avoid performance issues when working with `LocalStorage`?
 1. It's not possible at all.
 2. Using a custom JSON serializer.
 3. Using a WebWorker to handle the data writing and reading operations.
2. There are size limitations in PhoneGap when creating a database?
 1. No.
 2. Yes 5 MB.
 3. It depends on the target platform.

Have a go hero – create a Todo list app

Starting from one of the sample **Todo MVC** projects available at <http://todomvc.com/>, create a mobile app that stores the data into a local database.

Summary

In this chapter you learned how to save data on the device and how to handle the most common limitations. You also learned how the Contacts API works as well as about its limitations.

In the next chapter you will learn how to access the Device Sensor API in order to determine the device orientation, position, and so on.

7

Accessing Device Sensors

The use of device sensors opens the doors to sophisticated apps, able to improve user experience and to enhance the capabilities of a modern app. It's very important for a mobile developer to understand the power and limitations of device sensors to effectively use the APIs provided by the PhoneGap framework.

In this chapter you will:

- ◆ Learn which are the most common **device sensors** and how to use them in order to enhance the user experience
- ◆ Get an overview of the device orientation and device motion events using the accelerometer
- ◆ Learn how to work with device sensors directly with JavaScript
- ◆ Learn how to use the Compass API of PhoneGap
- ◆ Understand how to access the device compass data and use the available information in the app user interface

What are device sensors?

Humans have senses (touch, hear, smell, and so on); a phone has digital "senses": touch, geolocation, orientation, and motion. A sensor is a device component that measures a physical quantity and converts it into a signal that is understandable to software. Modern mobile phones come with a variety of sensors that can support users when completing their daily tasks. By tapping into a device sensor you can enhance the end user experience and develop sophisticated apps.

Sensors can be hardware-based or software-based. Hardware-based sensors are physical components built into a handset or tablet device. They derive their data by directly measuring specific environmental properties, such as acceleration, geomagnetic field strength, or angular change. Software-based sensors are not physical devices, although they mimic hardware-based sensors.

Typical device sensors are the **accelerometer**, the **gyroscope**, the **compass**, the **barometer**, the **orientation** sensor, and so on.

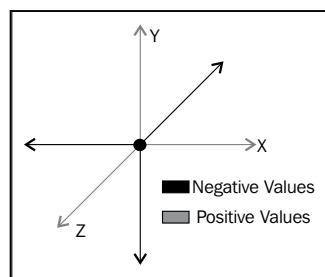
Not all devices, nor their operating systems, support the same sensors, so you have to know which devices you want to target before starting to consider which sensors to use in your app. The device sensors typically are divided into the following categories:

- ◆ Motion sensors
- ◆ Environmental sensors
- ◆ Position sensors

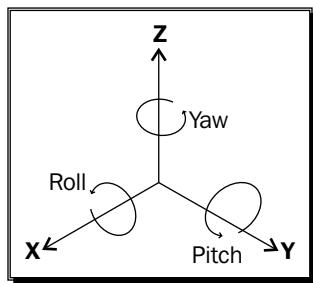
The **motion sensors** measure acceleration forces and rotational forces along three axes. Hardware parts such as the accelerometer, gravity sensors, gyroscopes, and rotational vector sensors belong to this category. The **environmental sensors** measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. The barometers, photometers and thermometers belong to this category of sensors. The **position sensors** measure the physical position of a device.

As already mentioned, each operating system offers different sensors. From a developer's point of view this means that to work on different platforms you have to understand how sensors work on each one. When working with PhoneGap you can safely use the **Accelerometer** and **Compass** APIs across different platforms. Furthermore, you can rely on the **onboard browser** capabilities to get additional sensor information such as the device orientation.

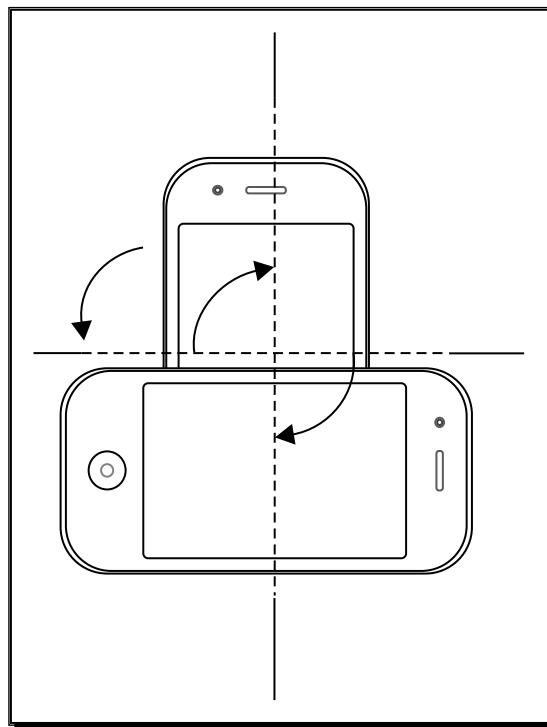
The accelerometer is actually made up of three accelerometers and each one measures the changes in velocity (i.e., linear acceleration) over time along the linear path on the axes **x**, **y**, and **z**. Combining the data of the three accelerometers, you can get device movement and orientation.



The gyroscope is always part of the motion sensors and measures the rate of rotation around the three axes usually **roll**, **pitch**, and **yaw**.



The magnetometer measures the strength of the magnetic field surrounding the device and in the absence of any strong local fields these measurements will refer to the magnetic field of the Earth. In this way the device is able to determine its **heading** with respect to the geomagnetic North Pole; using the heading values it's possible to determine the yaw of the device, too. Magnetic heading updates are available even if the user has switched off location updates in the settings application; the reported values are positive numbers from **0** to **360**. The real heading of the user, when they are holding the device in landscape mode, is the reported heading plus **90** degrees.



The iOS platform provides all the common sensors a developer can expect such as accelerometer, magnetometer, gyroscope, and the proximity sensor.

The Android platform provides four additional sensors that let you monitor various environmental properties: ambient **humidity**, **luminance**, ambient **pressure**, and ambient **temperature**. All the sensors are hardware-based and are available only if a device manufacturer has built them into a device.



You can find a complete demo of the Android sensors on the Google Play store; just search and install the app **Android Sensor Box**.



The Windows Phone 7.5/8 platform offers wide support for sensors. You can use the **Inclinometer** sensor to detect the pitch, roll, and yaw of the device or you can create complex 3D apps using the **Quaternion** sensor (quaternion is the quotient of two directed lines in a three-dimensional space). For a complete overview of the Windows Phone sensor APIs, please refer to the online documentation at <http://msdn.microsoft.com/library/windows/apps/windows.devices.sensors>.

The location capabilities of a device rely upon several sensors called position sensors. Devices normally use multiple positioning methods to provide different granularities of location data. The sources of position data vary in terms of accuracy, startup time, and power signature and include the following:

- ◆ GPS
- ◆ A-GPS
- ◆ Cell tower triangulation
- ◆ Wi-Fi triangulation
- ◆ IP address

With the continuous evolution of sensors, end user expectations are growing and the quality of the apps available on the market is increasing.



Lapka Electronics released a set of sensors and an app able to translate environmental data to read values easily. Using their sensors and app you can measure electromagnetic pollution, humidity, amounts of nitrates in raw foods and drinking water, and so on. More info about these sensors is available online at <http://mylapka.com/>.



Sensors and human-computer interaction

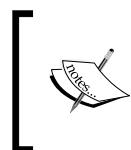
Sensors evolved and are still evolving very fast and are influencing how creative people are designing apps. The new generation of apps rely on voice commands, gestures, and more in order to allow the user to control apps in a more intuitive way. An app is now able to perceive the user's intentions based on the sensor data it collects. The use of sensors to make apps (and computers) more intuitive to control is known as **perceptual computing**. This initiative is led by Intel and has various applications including video conferencing, gaming, and so on.

By contrast, **augmented reality** is about extending how humans interface with the physical world through computers. Using an augmented reality interface, you can add additional information to the external environment and create amazing and useful apps. On mobile devices, the implementation of an augmented reality application heavily depends on the sensors on the devices, such as video cameras, and orientation sensor.

A nice example of the kind of interactions you can reach through sensors is an app for iOS named **Car Finder**. The app stores the position of a car when the user takes a picture of it and then provides to the user the information needed to find where he/she had parked the car.



The capability to use sensors and the data they return is increasingly paramount for mobile developers. The sensors supported by PhoneGap are limited but PhoneGap is simply a wrapper that makes it easier to separate your presentation layer from the native device code. For this reason you can start to write additional native code around the PhoneGap wrapper to extend its capabilities.



An interesting resource on sensor development is available on the Microsoft website at <http://research.microsoft.com/en-us/groups/sendev/>, where you can find papers and resources to help you get started with sensors.



Accelerometer

The PhoneGap Accelerometer API allows you to detect the device movement change values relative to device orientation. Keep in mind that the accelerometer detects the values as a delta movement relative to the current device position. Even more important, it takes into consideration the effect of gravity (i.e., 9.81 m/s^2), so that, when a device is lying flat on a table facing up, the value returned should be $x = 0$, $y = 0$, and $z = 9.81$.

You can detect the device acceleration data using the method `getCurrentAcceleration` or setting up a watcher through the method `watchAcceleration`. Both methods are available on the `navigator.accelerometer` object and accept similar arguments.

The `getCurrentAcceleration` method accepts a success and a failure callback function as argument and doesn't return anything. The `watchAcceleration` method accepts an additional argument in order to define the options and return a reference to the current watcher.

In order to constantly watch the acceleration data, you have to define the frequency at which you want to recover data and store the value returned by the `watchAcceleration` method in a variable.

```
var options = {frequency: 300};  
var currentAcceleration = navigator.accelerometer.watchAcceleration  
    (onSuccess, onFailure, options);
```

The `onSuccess` handler receives as an argument an `Acceleration` object, accessing its property, making it possible to read the acceleration on each axis.

```
function onSuccess(acceleration) {
    console.log('Acceleration X: ' + acceleration.x);
    console.log('Acceleration Y: ' + acceleration.y);
    console.log('Acceleration Z: ' + acceleration.z);
}
```

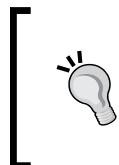
The failure handler doesn't receive any argument, but it's pretty useful to handle possible errors when accessing the device's accelerometer.

```
function onError() {
    console.log('Error accessing the accelerometer');
}
```

In order to stop watching the accelerometer data, it's enough to call the `clearWatch` method defined on the `accelerator` object passing the reference to the variable previously used to store the result of the `watchAcceleration` method.

```
navigator.accelerometer.clearWatch(currentAcceleration);
```

The method doesn't accept any additional handler.



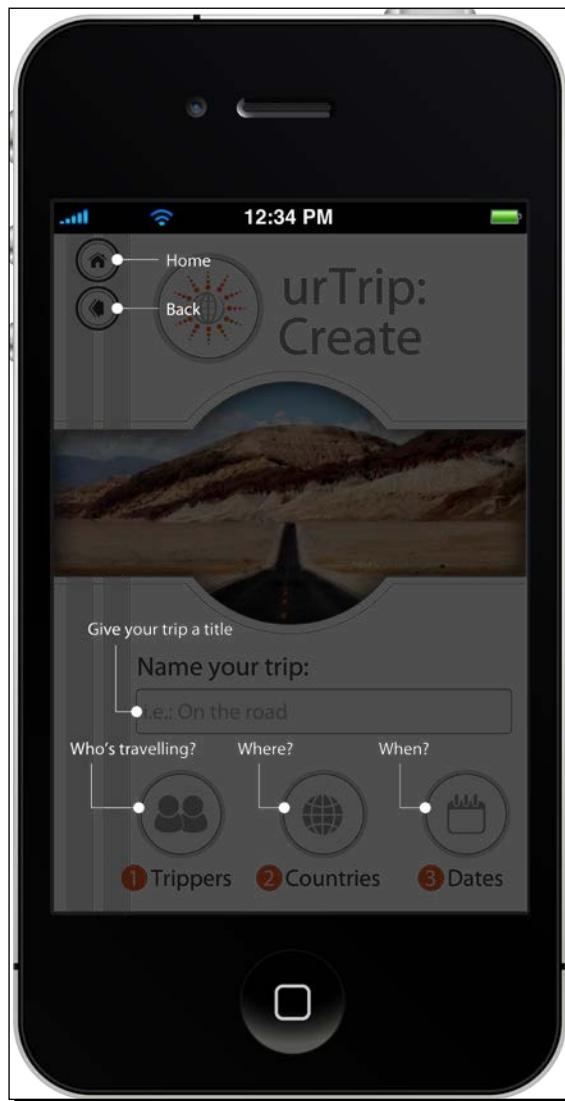
All the sensor APIs of PhoneGap work in a similar way; you will always have to use a `getCurrentSENSOR` and a `watchSENSOR` method (where `SENSOR` is the name of the sensor) to get data from the sensor. In order to stop watching a sensor you will always use the `clearWatch` method.

Detecting shakes

Using the information recovered from the accelerometer API, it's possible to understand whether the user is shaking the device. In order to implement this feature you have to load a new template in overlay on the app user interface, so that you can get the effect visible in the following image.

Accessing Device Sensors

In order to focus on the topics discussed so far, I will only touch on the styles needed to open and render the overlay. In the following steps you will see how to integrate a **shake detection module** in your app, how to render the contextual **help screen** loading the CSS only when needed, and how to remove it from the DOM.



Time for action – detecting shakes in your app

Execute the following steps:

1. Add the Accelerometer API plugin using the command line.

```
$ cordova plugins add https://git-wip-us.apache.org/repos/asf/cordova-plugin-device-motion.git
```

2. Go to the `tpl` folder and create a new **Mustache** template named `helpmain-tpl.html` containing two unordered lists; don't forget to assign to the root tag of the template an ID in order to be able to easily remove the help from the DOM.

```
<section id='help-index' class='help-overlay help-black'>
    <ul id='mainControls' class='help-items'>
        <li>{{home}}</li>
        <li>{{back}}</li>

    </ul>
    <ul id='sectionControls'>
        <li>{{tripname}}</li>
        <li>{{trippers}}</li>
        <li>{{countries}}</li>
        <li>{{dates}}</li>
    </ul>
</section>
```

3. Go to the `css` folder and create a new stylesheet named `help.css` in order to define the appearance of the help screens.

4. Go to the `js/utils` folder and create a new Require.js module named `shakesdetect.js`.

```
define('utils/shakesdetect', (function() {

    // The shake detection logic will go here

    return{

        // A reference to the exposed functions will go here

    }
}));
```

5. Create four variables at the very beginning of the module to store a reference to the x, y, and z values read from the accelerometer, a reference to the current sensor watcher, and a Boolean to understand whether the device has just been shaken.

```
var previousValue = {x: null, y: null, z: null},  
    newValues = {x: null, y: null, z: null },  
    watcher,  
    _shook;
```

6. Create a new function named start and in the body begin to monitor each for 300 milliseconds the values returned from the accelerometer and compare them with an arbitrary bound value; use the _shook flag in order to avoid executing the calculation if the device already has been shaken.

```
var bound = 3;  
watcher = navigator.accelerometer.watchAcceleration  
    (function (accelerometer) {  
  
        if (_shook) return;  
        // The shake detection will be defined here  
  
    }, null, {frequency: 300});
```

7. Once you have checked whether the device already has been shaken, it's important to determine if some information read from the accelerometer has been stored; if yes it's pretty easy to store in the newValues object the max value between the current acceleration and the last information stored.

```
if (previousValue.x !== null || previousValue.y != null) {  
  
    newValues.x = Math.max(previousValue.x, accelerometer.x);  
    newValues.y = Math.max(previousValue.y, accelerometer.y);  
    newValues.z = Math.max(previousValue.z, accelerometer.z);  
  
}
```

8. To determine if the device has been shaken it's enough to compare the values stored in the newValues object with the previously defined bound variable; if the x or y values are greater than the bound variable, then you can dispatch Event and suspend the shake detection for 3500 milliseconds.

```
if (newValues.x > bound || newValues.y > bound) {  
  
    var event = document.createEvent("Event");  
    event.initEvent("deviceshake", true, true);  
  
    _shook = true;
```

```
var delayed = setInterval(function() {  
  
    _shook = false;  
    clearInterval(delayed);  
    delayed = null;  
  
}, 3500);  
  
this.dispatchEvent(event);  
  
}
```

- 9.** Create a new function named `stop`, in its body clear the accelerometer watcher, and reset the initial values of the variables used in the calculations.

```
navigator.accelerometer.clearWatch(watcher);
```

```
previousValue = {  
  
    x: null,  
    y: null,  
    z: null  
  
};  
newValues = {};  
watcher = null;
```

- 10.** Expose the functions `start` and `stop` using the `return` object previously defined.

```
return{  
  
    start: start,  
    stop: stop  
  
}
```

- 11.** Open the file `approuter.js` stored in the folder `js/routers/` and define an object to store the information you need for each help screen of your app; each property of the object contains another object to pass the data to the Mustache template and the name of the template to use for each help screen.

```
var helps = {  
  
    index: new HelpContent({home: 'Home',  
                           back: 'Back',  
                           tripname: 'Give a name to your trip',  
                           trippers: 'Who\'s travelling',  
                           })  
};
```

```
        countries: 'Where?' ,
        dates: 'When?' ,
        'helpmain-tpl.html') ,
create: new HelpContent({}, 'helpcreate-tpl.html') ,
open: new HelpContent({}, 'helpopen-tpl.html') ,
share: new HelpContent({}, 'helpshare-tpl.html') ;
}
```

- 12.** Add a listener for the `deviceshake` event and in the event handler require and start the `Help` controller you will define in the next steps.

```
require(['controllers/Help'], function(controller) {

    var data = helps[_currentView];

    controller.init(data, 'help-' + _currentView);
    controller.start();

}) ;
```

- 13.** Go to the `js/controllers` folder and create a new Require.js module named `Help.js`; the controller depends on the `HelpView` view that you will create next.

```
define('controllers/Help',
['views/help/HelpView'], (function(view) {
    return {

        // A reference to the exposed functions go
        here

    }
})) ;
```

- 14.** Go to the `views` folder, create a new folder named `help`, and inside it a new Require.js module named `HelpView.js`; define the dependency from the `templateProvider` utility.

```
define('views/help/HelpView',
['utils/templateProvider'], (function(tplProvider) {

    return{

        // A reference to the exposed functions will go
        here

    }
})) ;
```

- 15.** In the same module create a function named `render` able to understand if the CSS file defined for the help is already loaded (and eventually to load it) and to load a template using the data previously defined.

```
var styles = document.styleSheets,
    skip = false;

for (var i = 0, max = styles.length; i < max; i++) {

    if (styles[i].href.indexOf('/css/help.css') >= 0) {

        tplProvider.loadTemplate(file, data);
        skip = true;
        break;

    }

}

if(skip) return;

require([
    'css!../../css/help'
], function(){

    tplProvider.loadTemplate(file, data);

});
```

 To dynamically load a CSS with Require.js you can use several plugins; the one used in this example is named **CSSLoader** and it's available on GitHub (<https://github.com/dimaxweb/CSSLoader>). Remember that, in order to use a plugin with Require.js, you have to save it in the folder `js/libs/require/plugins` and then define it as a path in the configuration object of your app.

- 16.** Expose the `render` function in the `return` object of the `HelpView` module.

```
return{

    render: render

}
```

- 17.** Go back to the `Help` module and define three variables to store the help template data and the template ID, and to determine whether there is a help template rendered on the screen.

```
var helpData, helpID, _opened;
```

- 18.** Create a new function named `init` that accepts two arguments (the template data and ID) and then in its body, define a listener for the `touchstart` and `templateready` events.

```
helpData = data;
helpID = id;
```

```
addEventListener('touchstart', closeHelp);
addEventListener('templateready', onTemplateReady);
```

- 19.** Create the `onTemplateReady` function already defined as a listener for the `deviceready` event; the role of the function is to inject the template in the DOM and to ensure the handler is not executed again when the help is opened.

```
document.querySelector('body').innerHTML += event.detail.html;
_opened = true;
removeEventListener('templateready', onTemplateReady);
```

- 20.** Create the `closeHelp` function and inside the function body remove the previously added help screen and set up a listener for the `templateready` event.

```
event.preventDefault();
```

```
if(helpID) {

    var currentDiv = document.getElementById(helpID);

    while (currentDiv.firstChild) {

        currentDiv.removeChild(currentDiv.firstChild);

    }

    helpID = null;
    _opened = false;

}
```

- 21.** Similarly to the other views already created in the previous chapters, create a function named `start` that, after checking that there is no other help screen visible, renders the view (the `init` and `start` functions are the ones called from the handler of the `deviceshake` event defined in the `approuter.js` file).

```
if(!_opened){  
  
    view.render(helpData.template, helpData.data);  
  
}
```

- 22.** Expose the `init` and `start` functions in the `return` object of the `Help` module.

```
return{  
  
    init: init,  
    start: start  
  
}
```

What just happened?

You created the backbone of a help system that the user can activate when shaking the device. Using device sensors to enhance the end user experience is a very good practice especially because on mobile devices the input operations are not straightforward for the end user.

Device orientation events

The PhoneGap framework supports only the compass and accelerometer. In order to handle the orientation changes you have to rely on the JavaScript APIs of the target platform browser. The user interface updates due to the device orientation changes are handled through CSS media queries; any other business logic can be handled using JavaScript due to the fact that PhoneGap uses the web view to render the app user interface.

Using JavaScript, you can set up a listener for the `orientationchange` event and another listener for the `deviceorientation` event in order to handle the device orientation. The first event is fired each time the orientation of the device changes; the second one is fired when the physical orientation of the device changes. Both the listeners have to be registered to the `window` object.

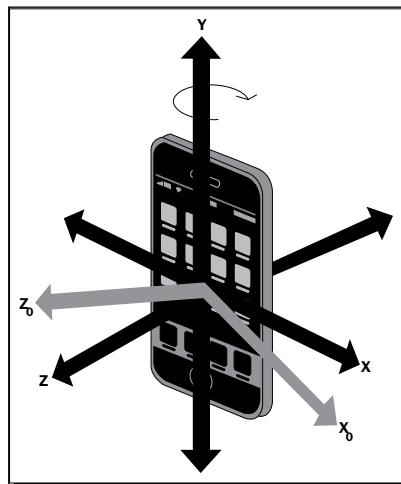
```
window.addEventListener('orientationchange', EVENT_HANDLER);  
window.addEventListener('deviceorientation', EVENT_HANDLER);
```

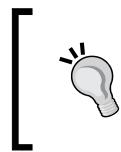
The `orientationchange` event handler is commonly used to detect the screen orientation after it has changed. Once the orientation changes, the app receives a notification for several events. The following table summarizes these events and the orientation property value:

Device and user gesture	Events fired	Orientation
iPad to landscape	resize	0
	orientationchange	90
iPad to portrait	resize	90
	orientationchange	0
iPhone to landscape	resize	0
	orientationchange	90
iPhone to portrait	resize	90
	orientationchange	0
Android phone to landscape	orientationchange	90
	resize	90
Android phone to portrait	orientationchange	0
	resize	0

The `deviceorientation` event is very powerful. It returns to the handler an instance of the `DeviceOrientationEvent` event with the following information:

- ◆ `alpha` returns the rotation of the device around the Z axis
- ◆ `beta` returns the rotation of the device around the X axis
- ◆ `gamma` returns the rotation of the device around the Y axis





In order to improve the performance of your app, consider using the event-handler function to do no more than save current values from the sensor data into variables. Then move your calculations or DOM manipulations into a new function executed at a fixed time.

Handling orientation with JavaScript

It's time to put into practice what you learned about the device orientation events. Let's work on a very basic sample that is able to show the screen orientation in a `div` rotated accordingly to the device physical orientation.

Time for action – handling device orientation with JavaScript

Execute the following steps:

1. Open the command-line tool and create a new PhoneGap project named `orientationevents`, and add the platforms you want to target for this example.
2. Go to the `www` folder, open the `index.html` file, and add `div` with the `#orientation` ID inside the main `div` of the app beneath `#deviceready`.

```
<div class="app">
    <h1>Apache Cordova</h1>
    <div id="deviceready">
        .....
    </div>
    <div id = "orientation">
        </div>
    </div>
```

3. Go to the `css` folder and define two new rules inside the `index.css` file to give to `div` and its content a border and a bigger font size.

```
#orientation{
    width: 230px;
    border: 1px solid rgb(10, 1, 1);
}

#orientation p{
    font-size: 36px;
    font-weight: bold;
    text-align: center;
}
```

4. Go to the js folder, open the index.js file, and define a new function to easily detect if the device can handle the orientationchange and the deviceorientation events.

```
orientationSupported: function() {  
  
    try {  
        return 'DeviceOrientationEvent' in window &&  
            window['DeviceOrientationEvent'] !== null;  
    } catch (e) {  
        return false;  
    }  
  
}
```

5. In the deviceready function, add two listeners if the device supports the orientationchange and the deviceorientation events.

```
if(app.orientationSupported){  
  
    window.addEventListener('orientationchange',
        app.orientationChanged);
    window.addEventListener('deviceorientation',
        app.updateOrientation);  
  
}else{  
  
    navigator.notification.alert('Orientation not supported!',
        null, 'Attention!', 'OK');  
  
}
```

6. Define the orientationChanged event handler and use it to print on screen the current device orientation.

```
orientationChanged: function() {  
  
    var element = document.querySelector('#orientation');
    element.innerHTML = '<p>' + window.orientation + '</p>';  
  
}
```

7. Define the handler for the deviceorientation event and use the information provided by the device's sensor to change the 3D transformation of the orientation div.

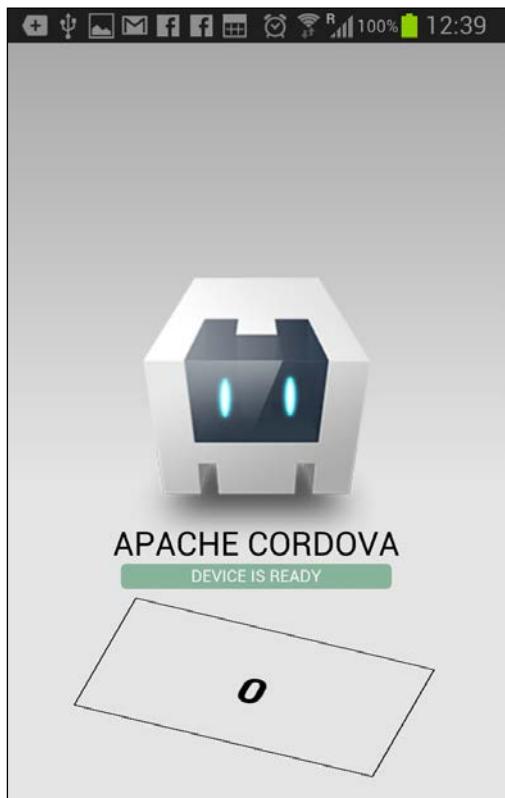
```
updateOrientation: function(event){  
  
    var alpha = event.alpha,
```

```
beta = event.beta,  
gamma = event.gamma;  
  
var element = document.querySelector('#orientation');  
var rotation = 'rotateZ(' + alpha + 'deg) rotate  
(' + beta + 'deg) rotateY(' + gamma + 'deg)';  
// For brevity the browser prefixes have been removed  
element.style.transform = rotation;  
  
}
```

8. Open the command-line tool again, locate the main project folder, and then compile the app and test it on every platform you previously added.

What just happened?

You handled the orientation events using JavaScript and deployed the result to a device using PhoneGap. The app is able to get the device screen orientation and the current position in real time as shown in the next image:



Compass

The PhoneGap Compass API allows you to obtain the direction that the device is pointing to. The compass is a sensor that detects the direction or heading in which the device is pointed and returns the heading of the device in degrees using values from 0 to 359.99. The Compass API works similarly to the Accelerometer API; in fact you can read the current device heading or you can define a watcher in order to continuously read the heading value.

The Compass API is available on the `compass` property of the `navigator` object and exposes the following functions:

- ◆ `compass.getCurrentHeading`, reads the current compass heading through a handler.
- ◆ `compass.watchHeading`, reads the compass heading at a specific time interval through a handler and returns a reference to it.
- ◆ `compass.clearWatch`, stops a previously defined time interval reading handler.

The `getCurrentHeading` and `watchHeading` functions accept very similar arguments; the only difference is the last argument of the `watchHeading` function that allows you to configure it. In order to read the current heading of the device, it suffices to execute the `getCurrentHeading` function, specifying a success and an error handler.

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

The `onSuccess` handler receives as argument a `CompassHeading` object with the following properties:

- ◆ `magneticHeading`, the heading in degrees from 0 to 359.99.
- ◆ `trueHeading`, the heading relative to the geographic North Pole in degrees.
- ◆ `headingAccuracy`, the deviation in degrees between the reported heading and the true heading.
- ◆ `timestamp`, the time at which this heading was determined.

The error handler receives as an argument a `CompassError` object; the `CompassError` object has a property named `code` that returns two possible values: `CompassError.COMPASS_INTERNAL_ERR` or `CompassError.COMPASS_NOT_SUPPORTED`.

```
function onError (error) {  
    switch(true) {  
  
        case error.code == CompassError.COMPASS_INTERNAL_ERR:  
            // handle error  
    }  
}
```

```
navigator.notification.alert('Compass Error!', null, 'Info',
    'OK');
break;

case error.code == CompassError.COMPASS_NOT_SUPPORTED:
navigator.notification.alert('Compass Unavailable!', null,
    'Info', 'OK');
break;

default:
navigator.notification.alert('Generic Error!', null, 'Info',
    'OK');

}
}
```

The `watchHeading` function works like the `getCurrentHeading` function. The only difference is that it accepts an additional `CompassOption` object that allows you to set up how often to retrieve the compass heading in milliseconds (i.e., frequency) and the change in degrees required to initiate the success handler (i.e., filter).

```
var options = {frequency: 300};
var currentHeading = navigator.compass.watchHeading(
    onSuccess, onError, options);
```

In order to stop watching the heading value changes, it suffices to use the `clearWatch` function and the reference to the current heading watcher.

```
clearWatch(currentHeading);
```



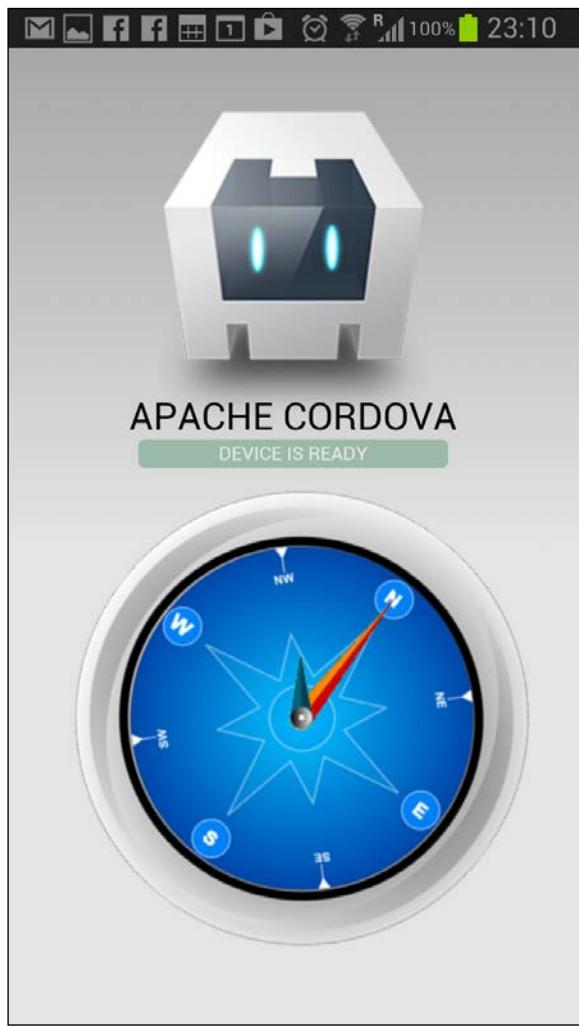
The `trueHeading` property of the `CompassHeading` object is not supported on Android. It returns the same value as `magneticHeading`, and the `headingAccuracy` will always be 0 as there is no difference between `magneticHeading` and `trueHeading`.

On iOS the `trueHeading` property is returned only when location services are running using the `watchLocation` function.

Creating a compass

Reading the current heading of a device is a common task for a developer in several use cases such as traffic apps, augmented reality apps, or any app that incorporates a sense of direction. Let's see how to create a complete compass with PhoneGap that will look like the following image:

[ The images used to render the compass are available under the Creative Commons License at <http://commons.wikimedia.org/wiki/File:Compass.svg>. Before starting to work on this example, download the image and create three separate PNG files.]



Time for action – using the Compass API

Execute the following steps:

1. Open the command-line tool and create a new PhoneGap project named `compass` and add the platforms you want to target this example.

2. Add the Compass API plugin using the command line.

```
$ cordova plugins add https://git-wip-us.apache.org/repos/asf/cordova-plugin-device-orientation.git
```

3. Go to the `www` folder, open the `index.html` file, and add a `section` tag with the ID `#compass` inside the main `div` of the app beneath `#deviceready`; in the `section` tag, add three `div` tags in order to handle the compass arrows and the background.

```
<section id="compass">
    <div id="compassbg"></div>
    <div id="north"></div>
    <div id="arrow"></div>
</section>
```

4. Go to the `css` folder, open the `index.css` file, and define the rules needed to have a separate background for each element of the compass.

```
#compassbg {
    background-image: url(../img/Compass.png);
}
#north {
    background-image: url(../img/arrow_direction.png);
}

#arrow {
    background-image: url(../img/arrow_beta.png);
}

#compass, #arrow, #north, #compassbg {
    background-repeat: no-repeat;
    background-size: cover;
    position:fixed;
    width: 286px;
    height: 286px;
}
```

5. Go to the `js` folder, open the `index.js` file, and add a property to the `app` object in order to store a reference to the watcher you will define to monitor the device heading.

```
var app = {  
  
    currentHeading: null,  
    .....  
}
```

6. Locate the `deviceready` function and add inside it the snippet of code needed in order to check the device heading every 100 milliseconds.

```
var options = {frequency: 150};  
app.currentHeading = navigator.compass.watchHeading  
(app.onCompassSuccess, app.onCompassError, options);
```

7. Create a new function in a property on the `app` object named `onCompassSuccess` and inside its body start to read the heading data stored in the received argument; use it to rotate the compass elements.

```
onCompassSuccess: function(heading) {  
  
    var magneticHeading = heading.magneticHeading,  
    trueHeading = heading.trueHeading;  
  
    var compass = document.querySelector('#compassbg'),  
        north = document.querySelector('#north');  
  
    var compassRotation = 'rotate(' + magneticHeading + 'deg)',  
        northRotation = 'rotate(' + trueHeading + 'deg)';  
    var compassStyle = compass.style,  
        northStyle = north.style;  
  
    // For brevity all the browser prefixes have been removed  
  
    compassStyle.transform = compassRotation;  
    northStyle.transform = northRotation;  
  
}
```

8. Open the command-line tool again, locate the main project folder, compile the app, and test it on every platform you previously added.

What just happened?

You implemented a real, cross-platform compass using the PhoneGap API. In the process you learned how to use a pretty complex feature of mobile device sensors.

Pop quiz – getting started with mobile apps

Q.1. How can you detect a change in the orientation of the device?

1. It's not possible at all.
2. Using JavaScript.
3. Using a custom plugin.

Q.2. Are the sensor APIs asynchronous?

1. No.
2. Yes.
3. It depends on the target platform.

Have a go hero – rendering the device orientation

Create a sample app to deploy on different target platforms in order to render the orientation of the device and to better understand how the values change according to the device.

Summary

In this chapter you learned how to work with device sensors to enhance the functionality of your app. Furthermore, you continued to gain an understanding of the PhoneGap APIs that allow you to create powerful native apps.

In the next chapter, you will start to work with geolocation data to improve the quality of information available in your app.

8

Using Location Data with PhoneGap

Location data allow a mobile developer to tag every piece of information with the device's position. This kind of meat tagging opens the doors to very contextualized apps. The PhoneGap framework provides a Geolocation API that is simple to use, easy to understand, and very powerful.

In this chapter you will:

- ◆ Learn about geolocation and how its data are available in the device
- ◆ Explore the differences between the HTML5 and the PhoneGap Geolocation APIs
- ◆ Learn how to use the PhoneGap Geolocation API and how to integrate the Google Maps API in an app
- ◆ Learn how to use Geolocation data in conjunction with external service providers such as Google Places

An introduction to Geolocation

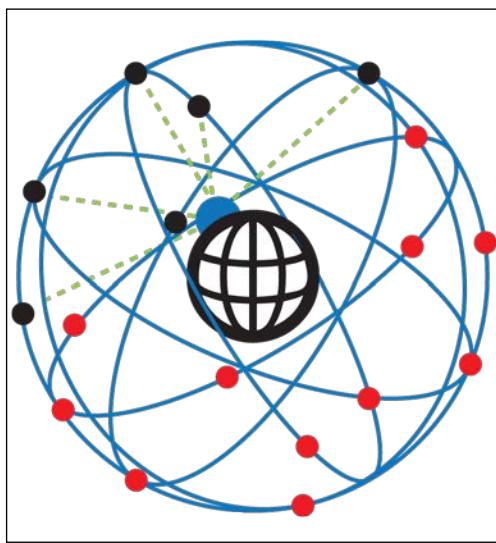
The term **geolocation** is used in order to refer to the identification process of the real-world geographic location of an object. Devices that are able to detect the user's position are becoming more common each day and we are now used to getting content based on our location (**geo targeting**).

Using the **Global Positioning System (GPS)**—a space-based satellite navigation system that provides location and time information consistently across the globe—you can now get the accurate location of a device. During the early 1970s, the US military created Navstar, a defense navigation satellite system. Navstar was the system that created the basis for the GPS infrastructure used today by billions of devices. Since 1978 more than 60 GPS satellites have been successfully placed in the orbit around the Earth (refer to http://en.wikipedia.org/wiki/List_of_GPS_satellite_launches for a detailed report about the past and planned launches).

The location of a device is represented through a point. This point is comprised of two components: latitude and longitude. There are many methods for modern devices to determine the location information, these include:

- ◆ Global Positioning System (GPS)
- ◆ IP address
- ◆ GSM/CDMA cell IDs
- ◆ Wi-Fi and Bluetooth MAC address

Each approach delivers the same information; what changes is the accuracy of the device's position. The GPS satellites continuously transmit information that can parse, for example, the general health of the GPS array, roughly, where all of the satellites are in orbit, information on the precise orbit or path of the transmitting satellite, and the time of the transmission. The receiver calculates its own position by timing the signals sent by any of the satellites in the array that are visible.





The process of measuring the distance from a point to a group of satellites to locate a position is known as **trilateration**. The distance is determined using the speed of light as a constant along with the time that the signal left the satellites.

The emerging trend in mobile development is GPS-based "people discovery" apps such as Highlight, Sonar, Banjo, and Foursquare. Each app has different features and has been built for different purposes, but all of them share the same killer feature: using location as a piece of metadata in order to filter information according to the user's needs.

The PhoneGap Geolocation API

The Geolocation API is not a part of the HTML5 specification but it is tightly integrated with mobile development. The PhoneGap Geolocation API and the W3C Geolocation API mirror each other; both define the same methods and relative arguments. There are several devices that already implement the W3C Geolocation API; for those devices you can use native support instead of the PhoneGap API.



As per the HTML specification, the user has to explicitly allow the website or the app to use the device's current position.

The Geolocation API is exposed through the `geolocation` object child of the `navigator` object and consists of the following three methods:

- ◆ `getCurrentPosition()` returns the device position.
- ◆ `watchPosition()` watches for changes in the device position.
- ◆ `clearWatch()` stops the watcher for the device's position changes.

The `watchPosition()` and `clearWatch()` methods work in the same way that the `setInterval()` and `clearInterval()` methods work; in fact the first one returns an identifier that is passed in to the second one. The `getCurrentPosition()` and `watchPosition()` methods mirror each other and take the same arguments: a success and a failure callback function and an optional `configuration` object. The `configuration` object is used in order to specify the maximum age of a cached value of the device's position, to set a timeout after which the method will fail and to specify whether the application requires only accurate results.

```
var options = {maximumAge: 3000, timeout: 5000,
               enableHighAccuracy: true };
navigator.geolocation.watchPosition(onSuccess, onFailure, options);
```



Only the first argument is mandatory; but it's recommended to handle always the failure use case.

The success handler function receives as argument, a `Position` object. Accessing its properties you can read the device's coordinates and the creation timestamp of the object that stores the coordinates.

```
function onSuccess(position) {  
  
    console.log('Coordinates: ' + position.coords);  
    console.log('Timestamp: ' + position.timestamp);  
  
}
```

The `coords` property of the `Position` object contains a `Coordinates` object; so far the most important properties of this object are `longitude` and `latitude`. Using those properties it's possible to start to integrate positioning information as relevant metadata in your app.

The failure handler receives as argument, a `PositionError` object. Using the `code` and the `message` property of this object you can gracefully handle every possible error.

```
function onError(error) {  
    console.log('message: ' + error.message);  
    console.log ('code: ' + error.code);  
  
}
```

The `message` property returns a detailed description of the error, the `code` property returns an integer; the possible values are represented through the following pseudo constants:

- ◆ `PositionError.PERMISSION_DENIED`, the user denies the app to use the device's current position
- ◆ `PositionError.POSITION_UNAVAILABLE`, the position of the device cannot be determined



If you want to recover the last available position when the `POSITION_UNAVAILABLE` error is returned, you have to write a custom plugin that uses the platform-specific API. Android and iOS have this feature. You can find a detailed example at <http://stackoverflow.com/questions/10897081/retrieving-last-known-geolocation-phonegap>.

- ◆ `PositionError.TIMEOUT`, the specified timeout has elapsed before the implementation could successfully acquire a new `Position` object



JavaScript doesn't support constants such as Java and other object-oriented programming languages. With the term "pseudo constants", I refer to those values that should never change in a JavaScript app.

One of the most common tasks to perform with the device position information is to show the device location on a map. You can quickly perform this task by integrating Google Maps in your app; the only requirement is a valid API key. To get the key, use the following steps:

1. Visit the APIs console at <https://code.google.com/apis/console> and log in with your Google account.
2. Click the **Services** link on the left-hand menu.
3. Activate the Google Maps API v3 service.

Time for action – showing device position with Google Maps

Get ready to add a map renderer to the PhoneGap default app template. Refer to the following steps:

1. Open the command-line tool and create a new PhoneGap project named `MapSample`.

```
$ cordova create ~/the/path/to/your/source/mapmample com.gnstudio.pg.MapSample MapSample
```
2. Add the Geolocation API plugin using the command line.

```
$ cordova plugins add https://git-wip-us.apache.org/repos/asf/cordova-plugin-geolocation.git
```
3. Go to the `www` folder, open the `index.html` file, and add a `div` element with the `id` value `#map` inside the main `div` of the app below the `#deviceready` one.

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

4. Add a new `script` tag to include the Google Maps JavaScript library.

```
<script type="text/javascript"
       src="https://maps.googleapis.com/maps/api/js?key=
YOUR_API_KEY&sensor=true">
</script>
```

5. Go to the `css` folder and define a new rule inside the `index.css` file to give to the `div` element and its content an appropriate size.

```
#map{

    width: 280px;
    height: 230px;
    display: block;
    margin: 5px auto;
    position: relative;

}
```

6. Go to the `js` folder, open the `index.js` file, and define a new function named `initMap`.

```
initMap: function(lat, long){

    // The code needed to show the map and the
    // device position will be added here

}
```

7. In the body of the function, define an `options` object in order to specify how the map has to be rendered.

```
var options = {

    zoom: 8,
    center: new google.maps.LatLng(lat, long),
    mapTypeId: google.maps.MapTypeId.ROADMAP

};
```

- 8.** Add to the body of the `initMap` function the code to initialize the rendering of the map, and to show a marker representing the current device's position over it.

```
var map = new google.maps.Map(document.getElementById('map'),  
    options);  
  
var markerPoint = new google.maps.LatLng(lat, long);  
  
var marker = new google.maps.Marker({  
  
    position: markerPoint,  
    map: map,  
    title: 'Device\'s Location'  
});
```

- 9.** Define a function to use as the success handler and call from its body the `initMap` function previously defined.

```
onSuccess: function(position) {  
  
    var coords = position.coords;  
    app.initMap(coords.latitude, coords.longitude);  
  
}
```

- 10.** Define another function in order to have a failure handler able to notify the user that something went wrong.

```
onFailure: function(error) {  
  
    navigator.notification.alert(error.message, null);  
  
}
```

- 11.** Go into the `deviceready` function and add as the last statement the call to the Geolocation API needed to recover the device's position.

```
navigator.geolocation.getCurrentPosition(app.onSuccess, app.  
onError, {timeout: 5000, enableAccuracy: false});
```

- 12.** Open the command-line tool, build the app, and then run it on your testing devices.

```
$ cordova build  
$ cordova run android
```

What just happened?

You integrated Google Maps inside an app. The map is an interactive map most users are familiar with—the most common gestures are already working and the Google Street View controls are already enabled.



To successfully load the Google Maps API on iOS, it's mandatory to whitelist the `googleapis.com` and `gstatic.com` domains. Open the `.plist` file of the project as source code (right-click on the file and then **Open As | Source Code**) and add the following array of domains:

```
<key>ExternalHosts</key>
<array>
    <string>*.googleapis.com</string>
    <string>*.gstatic.com</string>
</array>
```



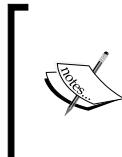
Other Geolocation data

In the previous example, you only used the `latitude` and `longitude` properties of the `position` object that you received. There are other attributes that can be accessed as properties of the `Coordinates` object:

- ◆ `altitude`, the height of the device, in meters, above the sea level.
- ◆ `accuracy`, the accuracy level of the latitude and longitude, in meters; it can be used to show a radius of accuracy when mapping the device's position.
- ◆ `altitudeAccuracy`, the accuracy of the altitude in meters.
- ◆ `heading`, the direction of the device in degrees clockwise from true north.
- ◆ `speed`, the current ground speed of the device in meters per second.

Latitude and longitude are the best supported of these properties, and the ones that will be most useful when communicating with remote APIs. The other properties are mainly useful if you're developing an application for which Geolocation is a core component of its standard functionality, such as apps that make use of this data to create a flow of information contextualized to the geolocation data. The `accuracy` property is the most important of these additional features, because as an application developer, you typically won't know which particular sensor is giving you the location and you can use the `accuracy` property as a range in your queries to external services.

There are several APIs that allow you to discover interesting data related to a place; among these the most interesting are the Google Places API and the Foursquare API.



The Google Places and Foursquare online documentation is very well organized and it's the right place to start if you want to dig deeper into these topics. You can access the Google Places docs at <https://developers.google.com/maps/documentation/javascript/places> and Foursquare at <https://developer.foursquare.com/>.

The `itinero` reference app for this book implements both the APIs. In the next example, you will look at how to integrate Google Places inside the RequireJS app you built in the previous chapter.



In order to include the Google Places API inside an app, all you have to do is add the `libraries` parameter to the Google Maps API call. The resulting URL should look similar to `http://maps.google.com/maps/api/js?key=SECRET_KEY&sensor=true&libraries=places`.

The itinero app lets users create and plan a trip with friends. Once the user provides the name of the trip, the name of the country to be visited, and the trip mates and dates, it's time to start selecting the travel, eat, and sleep options.



When the user selects the **Eat** option, the Google Places data provider will return bakeries, take-out places, groceries, and so on, close to the trip's destination. The app will show on the screen a list of possible places the user can select to plan the trip.



For a complete list of the types of place searches supported by the Google API, refer to the online documentation at https://developers.google.com/places/documentation/supported_types.

Time for action – discovering places with Google Places

Execute the following steps:

1. Add the Geolocation API plugin using the command line.

```
$ cordova plugins add https://git-wip-us.apache.org/repos/asf/cordova-plugin-geolocation.git
```

2. Download the `async` RequireJS plugin from GitHub at <https://github.com/millermedeiros/requirejs-plugins/blob/master/src/async.js> and save it to the `js/libs/require/plugins` folder. This plugin allows you to asynchronously load external libraries or dependencies. In this example, it will be used to load the Google Places API.

3. Add the path to the `async` plugin into the configuration of RequireJS you already defined in the `main.js` file.

```
require.config({
    paths: {
        // All the already defined paths
        async: 'libs/require/plugins/async'
    },
    // Other configurations
});
```

4. Go to the `js/utils` folder and create a new RequireJS module named `gmaps`. This module allows your app to download the Google Places API only when requested by another module; use the module's name to access the Google Maps API.

```
;define('utils/gmaps', ['async!http://maps.google.com/maps/api/js?key=YOUR_KEY&sensor=true&libraries=places'],
function() {
    return window.google.maps;
});
```

5. Create a new Mustache template named `create-find-places-tpl.html` to show to the user the available data providers (i.e. Google and FourSquare), to search for and return places in the selected city/country, and to save it in the `tpl` folder.

```
<section id='create-find-places' class='gray-background'>

    <div class="results-vertical">
        <h1 class='resultsText'>{{searchingFor}}</h1>
        <h2 class='resultsText'>{{place}}</h2>
        <ul class='placeproviders'>
            <li id='google'><a href='google'>{{google}}</a></li>
            <li id='square'><a href='square'>{{square}}</a></li>
        </ul>
        <div id='map'></div>
    </div>
    <div id='places-results'></div>
</section>
```

6. Create a new RequireJS module named `FindPlaceView.js`, define its dependencies, and save it into the `js/views/create/findplaces` folder (the module uses the `templateProvider` utility to load a new template, Mustache to add the results on the screen, and the `spinner` utility to show when the app is loading new data). The content of the file will look like the following snippet of code.

```
;define('views/create/findplaces/FindPlacesView',
['utils/templateProvider', 'mustache', 'utils/spinner'],
(function(tplProvider, mustache, spinner) {

    // The module variables and functions will go here
});
```

7. Create another RequireJS module named `FindPlaces.js` with an explicit dependency to the module `FindPlaceView.js` and save it into the `js/controllers` folder. The content of the file will look like the following snippet of code:

```
;define('controllers/FindPlaces',
['views/create/findplaces/FindPlacesView'], (function(view) {

    // The module variables and functions will go here
});
```

- 8.** Open the `FindPlaces.js` module and define a function to `init` the module and another one to start it and expose both functions using a `return` object. The `init` function stores the data needed by the template and the city used as search target; the `start` function initializes the view. The `init` function also registers a listener in order to enable the interactivity on the Google Places and Foursquare selectors only when the template is loaded.

```
function init(data, city){

    section = data;
    targetCity = city;
    addEventListener('initFinders', onInitFinders);

}

function start(){

    view.render(section.template, section.data);

}

return{

    init: init,
    start: start

}
```

- 9.** Define the `onInitFinders` listener and add to its body a call to the `Array.prototype.forEach` method in order to define the `click` event listener for all the `<a>` tags used to render the data provider options.

```
function onInitFinders(evt) {

    var links = document.getElementsByClassName
        ('placeproviders')[0].getElementsByTagName('a');

    forEach.call(links, function(link){

        link.addEventListener('click', getPlaces);

    });

}
```

- 10.** Open the `FindPlaceView.js` module previously created, define the `render` function, and expose it using the `return` object. The `render` function loads a new template and defines a listener for the `templateReady` event.

```
function getPlaces(evt) {  
  
    var provider = this.href;  
  
    if(provider.indexOf('google') >= 0){  
  
        loadFromGoogle();  
  
    }else if(provider.indexOf('foursquare') >= 0){  
  
        // The code to use FourSquare is available online  
  
    }  
  
    evt.preventDefault();  
  
}
```

- 11.** Define the `onTemplateReady` function in the `FindPlaceView.js` file and inside its body add the code needed to populate the template and to notify the `FindPlaces.js` controller that it can define the listeners for the data source selectors.

```
function onTemplateReady(event) {  
  
    var app = document.querySelector('div.app');  
  
    app.innerHTML = event.detail.html;  
  
    var event = document.createEvent("Event");  
    event.initEvent("initFinders", true, true);  
  
    this.dispatchEvent(event);  
  
}
```

- 12.** To complete the `FindPlacesView.js` module, define and expose a function to append data to the `places-results` div tag part of the `create-find-places-tpl.html` template.

```
function results (value) {  
  
    var tpl = '<ul>{{#places}}<li>';  
    tpl += '<a href="#">{{name}} - {{vicinity}}</a>';  
    tpl += '</li>{{/places}}</ul>';  
  
    var target = document.getElementById('places-results');  
    target.innerHTML = mustache.to_html(tpl, value);  
  
}
```

- 13.** Go back to the `FindPlace.js` module, define the `getPlaces` event listener, and call the method to load data using the Google Places API or the Foursquare API, depending on the user's selection.

```
function getPlaces(evt) {  
  
    var provider = this.href;  
  
    if(provider.indexOf('google') >= 0){  
  
        loadFromGoogle();  
  
    }else if(provider.indexOf('foursquare') >= 0){  
  
        // The code to use FourSquare is available online  
  
    }  
  
    evt.preventDefault();  
  
}
```

- 14.** In the same file, add the function to load the data using the Google Places API. The first thing the function has to do is to require the `gmaps` module already defined and waiting until the API is available.

```
function loadFromGoogle() {  
  
    require(['utils/gmaps'], function(gmaps) {  
  
        // Do some stuff when the API is available  
    }) ;  
}
```

- 15.** When the API is available, the anonymous function used as an argument in the `require` function has to create a new `Geocoder` object in order to recover the latitude and longitude of the target city. The `Geocoder` object is created in the `js/controllers/FindPlaces.js` file.

```
var geocoder = new gmaps.Geocoder();  
var address = targetCity;  
  
geocoder.geocode({ 'address': address },  
                  function(results, status) {  
                      // Start to search data  
                  }) ;
```

- 16.** When the value of the `status` argument used within the `geocode` method handler is `OK` you can recover the latitude and the longitude needed to define the center of the map to be used to perform the places search.

```
if (status == gmaps.GeocoderStatus.OK) {  
  
    var latitude = results[0].geometry.location.lat();  
    var longitude = results[0].geometry.location.lng();  
  
    var center = new gmaps.LatLng(latitude, longitude);  
  
    var mapDiv = document.getElementById('map');  
    var options = {  
        mapTypeId: gmaps.MapTypeId.ROADMAP,  
        center: center,  
        zoom: 9,  
        radius: 50000  
    };  
    var map = new gmaps.Map(mapDiv, options);  
  
    // The request to the API will go here  
  
}
```

- 17.** Inside the conditional statement, add a `request` object to specify the center of the request, the radius, and the types of places. Use the object to start a **Nearby Search** (a search for places within a specified area by keyword or type) and define a listener for the results.

```
var request = {  
    location: center,  
    radius: '500',  
    types: ['bakery', 'meal_takeaway', 'cafe']  
};  
  
var service = new gmaps.places.PlacesService(map);  
service.nearbySearch(request, onResult);
```

- 18.** Create the `onResult` function. The function receives two arguments, one is the search result and the other is the status of the research; according to the request status, populate the `places` property of the `currentData` object and then populate the view with the recovered data.

```
function onResult(results, status) {  
  
    if(status == google.maps.places.PlacesServiceStatus.OK) {  
  
        var currentData = {places: []};  
  
        for (var i = 0; i < results.length; i++) {  
  
            var place = results[i];  
  
            var data = {};  
            data.name = place.name;  
            data.vicinity = place.vicinity;  
            data.icon = place.icon;  
  
            currentData.places.push(data);  
  
        }  
  
        view.results(currentData);  
  
    }  
}
```



The properties name of the `currentData` object and the variables defined in the template snippet used in the `FindPlacesView.js` module match.

- 19.** Open the `Create.js` file saved in the `js/controllers` folder and add the snippet of code needed to load the new controller.

```
var section = {template: 'create-find-places-tpl.html',
               data: {
                  searchingFor: 'Places to eat something',
                  place: 'Rome - IT',
                  google: 'Google Places',
                  foursquare: 'FourSquare'
               }};

require(['controllers/FindPlaces'], function(controller){

  controller.init(section);
  controller.start();

});
```

What just happened?

The itinero app is now able to get information about places near the destination of the trip and the user can now add them to the trip's plan and notes.

Pop quiz – getting started with mobile apps

Q1. How can you detect a change in the position of the device?

1. It's not possible at all.
2. Using the PhoneGap API.
3. Using a custom plugin.

Q2. Is the Geolocation API asynchronous?

1. No.
2. Yes.
3. It depends on the target platform.

Have a go hero – improving the Google API usage

Rebuild the section of the app just discussed using the `goog` plugin available at <https://github.com/millermedeiros/requirejs-plugins> instead of the `async` plugin.

Summary

In this chapter, you learned how to get the Geolocation information from a device and how to integrate external Geolocation service in the app. In the next chapter, you will learn how to manipulate a file, access the device Camera, and perform some basic image manipulation.

9

Manipulating Files

Accessing the filesystem is a critical part of an app's business logic. The PhoneGap framework offers a very sophisticated API to access and read files, even download and upload data over the Internet. One of the strengths of the PhoneGap framework is that it widely relies on the W3C open standard. If you are familiar with the W3C FileSystem and FileUpload APIs, you will be impressed how quickly you will get used to the device filesystem.

In this chapter you will:

- ◆ Learn about the Files API, how it works, and how to organize your code to keep it clear and maintainable
- ◆ Use the Files API to explore the device filesystem
- ◆ Learn how to read and render data inside a file
- ◆ Learn how to load and save a file to a device's persistent storage

Understanding the Files API

The PhoneGap Files API is an implementation of two different W3C APIs, the Directories and System API and the File API (you can find the complete specifications on the W3C website at <http://www.w3.org/TR/file-system-api/> and <http://www.w3.org/TR/file-upload/>.) The PhoneGap Files API is not a complete implementation of the W3C specification; the missing piece is the synchronous filesystem interface implementation. Asynchronous JavaScript APIs are a bit more complex to use because you have to work with multiple nested functions but this should not be a big issue; in fact it's something web developers are all too familiar with.



The main difference between asynchronous and synchronous JavaScript execution is that in the first case you can run several processes simultaneously and avoid "freezing" the user interface. With the introduction of web workers in JavaScript, it's possible to avoid this issue but this is totally out of the scope for this book; you can find more information about web workers on the Mozilla website at https://developer.mozilla.org/en-US/docs/DOM/Using_web_workers.

In order to access the device filesystem, you can use the `requestFileSystem` method of the `LocalFileSystem` object; all the methods of this object are defined in the `window` object. The method accepts the following four arguments:

- ◆ The type of storage (temporary or persistent)
- ◆ The amount of space in bytes to be allocated on the device storage
- ◆ The success handler
- ◆ The error handler

When you want to access the device filesystem the resulting code looks like the following snippet:

```
window.requestFileSystem(*storage*, /*size*, onSuccess, onError);
```

For the `storage` argument, you need to specify one of the following two pseudo constants defined in the `LocalFileSystem` object:

- ◆ `LocalFileSystem.PERSISTENT`, indicates that the storage cannot be removed by the user agent without the app's or user's permission.
- ◆ `LocalFileSystem.TEMPORARY`, indicates that the files stored in the requested space can be deleted by the user agent or by the system without the app's or user's permission

The size of the requested sandbox storage is expressed in bytes; in order to make the code more readable you can use the syntax (4 x 1024 x 1024) to allocate 4 KB instead of the bytes number 41,94,304.



The device hard disk is not completely open to the app's view. A limited portion of the hard disk is dedicated to a single app alone; this is the app **sandbox**. The idea behind the app sandbox is that each app can only access its own sandbox and some higher-level directories owned by the operating system. The structure of the high-level directories varies depending on the operating system.

The `onSuccess` handler receives a `FileSystem` object as an argument. The two properties defined for this object are `name` and `root`. Accessing the `name` property of the object makes it possible to read the name of the filesystem; accessing the `root` property allows you to get a reference to the `root` directory of the app sandbox.

```
function onSuccess(fileSystem) {  
  
    console.log(fileSystem.name);  
    var currentRoot = fileSystem.root;  
  
}
```

The `onError` handler receives a `FileError` object as argument; this object represents different errors using several pseudo constants defined in the object itself.

```
function onError(fileError) {  
  
    console.log(fileError.code);  
  
}
```

If the location of the file or the directory is known, you can use the `resolveLocalFileSystemURI` method of the `LocalFileSystem` object to access it. This method accepts the following three arguments:

- ◆ The URI of the file or directory
- ◆ The success handler (`onSuccess`)
- ◆ The error handler (`onError`)

If you want to access, for instance, the external storage of an Android device, you can use the following syntax:

```
window.resolveLocalFileSystemURI('file:///mnt/sdcard',  
                                onSuccess, onError);
```

The `onSuccess` function receives as argument a `DirectoryEntry` or a `FileEntry` object depending on the kind of path entered (i.e., a directory or a file); the `onError` handler receives a `FileError` object as argument.

The values of the `code` property are summarized by the following pseudo constants:

- ◆ `FileError.NOT_FOUND_ERR` (returned value 1), the file or directory required by the app cannot be found
- ◆ `FileError.SECURITY_ERR` (returned value 2), the file or directory is outside the app sandbox or the app has not the rights to access it

- ◆ `FileError.ABORT_ERR` (returned value 3), thrown when the `abort` method of the reader or writer is called
- ◆ `FileError.NOT_READABLE_ERR` (returned value 4), the file or directory required by the app cannot be read
- ◆ `FileError.ENCODING_ERR` (returned value 5), a path or local URI used as argument in the `resolveLocalFileSystemURI` method of the `LocalFileSystem` object is malformed
- ◆ `FileError.NO_MODIFICATION_ALLOWED_ERR` (returned value 6), the app attempted to write to a file or directory that cannot be modified due to the actual state of the filesystem
- ◆ `FileError.INVALID_STATE_ERR` (returned value 7), the app is accessing a file that is used by another process
- ◆ `FileError.SYNTAX_ERR` (returned value 8), self explanatory; occurs due to the syntax error
- ◆ `FileError.INVALID_MODIFICATION_ERR` (returned value 9), the modification requested by the app is invalid; an example of such error is moving a directory into its own child
- ◆ `FileError.QUOTA_EXCEEDED_ERR` (returned value 10), the app requested a storage amount greater than the allowed storage quota
- ◆ `FileError.TYPE_MISMATCH_ERR` (returned value 11), the app attempted to access a file or directory but the entry is not of the expected type (i.e., a directory is returned instead of a file)
- ◆ `FileError.PATH_EXISTS_ERR` (returned value 12), the app failed to create a file or directory due to the existence of a file or directory with the same path

Reading directories and files

Only after getting access to the filesystem is it possible to read the device directories, subdirectories, and content. Again, the `onSuccess` handler used as argument in the `requestFileSystem` method receives a `FileSystem` object. Through the `root` property of this object it's possible to access a `DirectoryEntry` object and then create a `DirectoryReader` object able to read all the entries available in the current directory.

```
function onSuccess(fileSystem) {  
  
    var currentRoot = fileSystem.root;  
    var reader = currentRoot.createReader();  
  
}
```

The `DirectoryReader` object exposes one method named `readEntries`. It can be used in order to read the entries and, due to the asynchronous nature of the File API, it accepts a success and a failure handler. Similar to what's happening for the `resolveLocalFileSystemURI` method, the success handler receives an array of `DirectoryEntry` or `FileEntry` objects according to the kind of item that is listed (i.e., a directory or a file).

Time for Action – listing folders and files recursively

Get ready to explore the folders and their contents of the device's persistent storage. Use the following steps:

1. Open the command-line tool and create a new PhoneGap project named `FileSystem`.

```
$ cordova create ~/the/path/to/your/source/filesystem com.
gnstudio.pg.FileSystem FileSystem
```

2. Add the File API plugin using the command line.

```
$ cordova plugins https://git-wip-us.apache.org/repos/asf/cordova-
plugin-file.git
```

3. Go to the `www` folder, open the `index.html` file, and add a `div` element with the `id` value `#fileslist` inside the main `div` of the app, below the `#deviceready` one.

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

4. Go to the `www/js` folder, open the `index.js` file, and define a new function named `requestFileSystem`; the function has to be called once the `deviceready` event is fired.

```
requestFileSystem: function() {
```

```
// The file system access request will go here
```

```
}
```

5. In the body of the function, request access to the device filesystem specifying the success and failure handlers you will define next and request a persistent storage of 0 KB; you need to specify a quota only when writing to the device filesystem.

```
var size = 0;
window.requestFileSystem(LocalFileSystem.PERSISTENT, size,
                        app.onFileSystemSuccess,
                        app.onFileSysError);
```

- 6.** Define the error handler that will notify you when the code throws an error.

```
onFileSysError: function(error) {  
  
    navigator.notification.alert(error.code, null);  
  
}
```

- 7.** Define the success handler and inside its body create a new `DirectoryReader` object and use this object as an argument when calling the function `readDirAndFiles`.

```
onFileSystemSuccess: function(fileSystem) {  
  
    var currentRoot = fileSystem.root;  
    app.readDirAndFiles(currentRoot.createReader());  
  
}
```

- 8.** Define the function `readDirAndFiles` and add in its body a call to the `readEntries` method of the `DirectoryReader` object, specifying the success and failure handlers.

```
readDirAndFiles: function(reader) {  
  
    reader.readEntries(app.parseDirectories,  
                      app.onFileSysError);  
  
}
```

- 9.** Define in the `app` object an array you will use in order to store the directories and files information in the success handler used as an argument in the previous step.

```
var app = {  
    queuedDir: [],  
    // The rest of the index.js file is here
```

- 10.** Define the `parseDirectories` function, and in its body, iterate through the entries storing the data for later parsing. At the end of the loop the function `parseQueuedDirs` will be called in order to start the recursive reading of the device's external storage.

```
parseDirectories: function(entries) {  
  
    for (var i = 0, fs; fs = entries[i]; i++) {  
  
        app.queuedDir.push(app.parseEntries(fs));  
  
    }  
  
}
```

```
    app.parseQueuedDirs(0);

}
```



The syntax used in the `for` loop is a compact form that first declares a variable used to store the content of the current index of the array and exits from the loop when the variable is `undefined`.



11. The `parseEntries` function contains the logic needed in order to create and return an object used to store the information of each filesystem entry; the function creates a new `DirectoryReader` object for each directory in order to allow multiple asynchronous reading operations and initializes the `children` property in order to clearly identify the folders in the data structure.

```
parseEntries: function (fs) {

  var obj = {reader: null, html: '', children: null};

  if (fs.isDirectory) {

    obj.reader = fs.createReader();
    obj.html = '<p><strong>' + fs.name + '</strong>:' +
    + fs.fullPath + '</p><br>';
    obj.children = [];

  } else {

    obj.html = '<p>' + fs.name + ':' + fs.fullPath +
    '</p><br>';

  }

  return obj;

}
```

- 12.** The `parseQueuedDirs` function parses the entries stored in the `app.queuedDir` array using the previously created `DirectoryReader` to get the contents of the children folders and then calls the function needed to render the results; if an entry contains other folders, the `for` loop will start to read them.

```
parseQueuedDirs: function(start) {  
  
    for(var i = start,entry; entry = app.queuedDir[i]; i++) {  
  
        if (entry.children){  
  
            entry.reader.readEntries(function (entries) {  
  
                for (var j = 0, fs; fs = entries[j]; j++) {  
  
                    // The directory check will go here  
  
                }  
  
                app.parseQueuedDirs(i + 1);  
  
            }, app.onFileSysError);  
  
            return;  
        }  
  
    }  
  
    this.printDir(this.queuedDir);  
}
```

- 13.** In order to explore the nested folders in the `for` loop there is a conditional statement that checks if the entry is a directory. When the condition is met, a new `DirectoryReader` object is created and the `readEntries` method is called using a success handler argument of the function currently executed. When the condition is not met, the data is pushed into the `children` array.

```
if (fs.isDirectory) {  
  
    var reader = fs.createReader();  
    reader.readEntries(arguments.callee, app.onFileSysError);  
  
    return;  
  
} else {  
  
    entry.children.push(app.parseEntries(fs));  
  
}
```

- 14.** Define the `printDir` function and add in its body the code needed to print the HTML into the `div` element with the `id` value `fileslist`.

```
printDir: function(data) {  
  
    var element = document.querySelector('#fileslist');  
  
    for (var i = 0, item; item = data[i]; i++) {  
  
        element.innerHTML += item.html;  
  
        if (item.children.length > 0) {  
  
            for (var j = 0, nested;  
  
                nested = data[i].children[j]; j++) {  
  
                element.innerHTML += nested.html;  
  
            }  
  
        }  
  
    }  
  
}
```

What just happened?

The app is now able to read the content of the device's persistent storage using the asynchronous Files API provided by the PhoneGap framework.



As you have seen, the callback syntax can be complex to read, especially if you don't break down the code into small functions. There is an interesting experiment on GitHub that changes syntax highlighting to scope colorizing available at <https://github.com/daniellmb/JavaScript-Scope-Context-Coloring>. It can represent a solution to the nested callback reading issue; in fact this experiment address the reading difficulties you can have with highly nested asynchronous calls.

Writing and reading a file's data

To write data to a file it suffices that the app gets access to the file using the `FileWriter` object. In order to get a `FileWriter` object, you first have to get access to a `DirectoryEntry` object or to a `FileEntry` object using the `requestFileSystem` method of the `LocalFileSystem` object.

Once you successfully get access to the filesystem, you can request a file specifying that you want to create it using the `create` flag.

```
onFileSystemSuccess: function(fileSystem) {  
  
    var root = fileSystem.root;  
    root.getFile('data.txt', {create: true},  
                onGetFile, onGetFileError);  
  
}
```



There are two flags available within the Files API that can be used as arguments of the `getFile` and `getDirectory` methods: `create` and `exclusive`. The `create` flag is used to indicate that the file or directory should be created; the `exclusive` flag takes effect only when the `create` flag is set to `true` and it causes the file or directory creation to fail if it already exists.

As with the other Files API, the `getFile` method is asynchronous and requires a success and a failure handler. Once in the success handler, it's possible to create a `FileWriter` object using the `createWriter` method of the `FileEntry` object received as an argument. The `createWriter` method also requires the success and failure handlers.

```
function onGetFile(file) {
    file.createWriter(onGetWriter, onGetWriterError);
}
```

Once again you have two other handlers, which means only after three callback functions can you write some content into the file you just created.

```
function onGetWriter(writer) {
    writer.write('Hello PhoneGap Files API!');
}
```



You can't write binary data from JavaScript in PhoneGap using a `FileWriter` object; this is a limitation of the framework because it passes data between the native and JavaScript layers as a string. One possible solution is to write a plugin that translates a Base64 string into binary data.

During the write operation several events occur; for each event there is a corresponding property defined on the `FileWriter` object:

- ◆ `onwritestart` is called when the `FileWriter` object starts to write the file; it receives as argument a `ProgressEvent` object.
- ◆ `onwrite` is called when the `FileWriter` object has completed successfully the write operation; it receives as argument a `ProgressEvent` object.
- ◆ `onabort` is called when the write operation has been interrupted by calling the `abort` method of the `FileWriter`; it receives as argument a `ProgressEvent` object.
- ◆ `onerror` is called when the write operation fails; it receives as argument the `ProgressEvent` object. In order to understand why the error occurs, you can access the `FileError` object stored in the `target.error` property of the event object.

The `FileWriter` object contains other properties as well. For a complete overview refer to the online guide available at http://docs.phonegap.com/en/edge/cordova_file_file.md.html#FileWriter.

Using the `ProgressEvent` object, you can access the bytes loaded, the bytes total, and the nature of the event (i.e., `abort`, `writeend`, and so on) using the properties `loaded`, `total`, and `type`.

```
function onGetWriter(writer){  
  
    writer.onwrite = function(evt){  
  
        console.log(evt.loaded, evt.total, evt.type);  
  
    }  
  
    writer.write('Hello PhoneGap Files API!');  
  
}
```



If you try to call sequentially the `write` method of the `FileWriter` object, only the first string will be added to the file. You have to wait until the `writeend` event is fired in order to write other data to the file.



When you want to read a file you can use a `FileReader` object. This object works similarly to the `FileWriter` object. When using it, several events occur: The `onabort` and `onerror` properties of the `FileWriter` object act similarly to the `FileReader` ones. The properties related only to the `FileReader` object are:

- ◆ `onloadstart`, the function stored in the property is called when the `FileReader` object starts to read a file; it receives a `ProgressEvent` object as an argument.
- ◆ `onload`, the function stored in the property is called when the read operation has successfully completed; it receives a `ProgressEvent` object as an argument.
- ◆ `onloadend`, the function stored in the property is called when the read operation is completed (regardless of whether it succeeded or failed); it receives a `ProgressEvent` object as argument.

The `FileReader` object allows you to read the file data in the following four different ways:

- ◆ `readAsDataURL` reads the file and returns the content of the specified file as a Base64-encoded data URL
- ◆ `readAsText` reads a file and returns the data as a string encoded by default in UTF-8

- ◆ `readAsBinaryString` reads the file as binary and returns the data as a binary string
- ◆ `readAsArrayBuffer` reads the file and returns the data as `ArrayBuffer`

In order to put into practice what you have just learned, you will see now how to parse the device's persistent storage, recover the first available image, and render it in the app web view as shown in the following screenshot:



Time for Action – reading and rendering an Image

Get ready to render the first available image in the device's storage into the PhoneGap default app template. Refer to the following steps:

1. Open the command-line tool and create a new PhoneGap project named `ReadingFile`.
2. Go to the `www` folder, open the `index.html` file, and add an `img` tag with the `id` value `firstImage` inside the main `div` of the app below the `deviceready` one.
``

- 3.** Go to the `www/js` folder, open the `index.js` file, and define a new function named `requestFileSystem`.

```
requestFileSystem: function () {  
  
    // The request of access to the file system will go here  
  
}
```

- 4.** Define the error handler in order to get the code of every possible error.

```
onError: function(error) {  
  
    navigator.notification.alert(error.code, null);  
  
}
```

- 5.** In the body of the function `requestFileSystem`, access the device filesystem using the `requestFileSystem` function of the `LocalFileSystem` object, define the success and failure handlers, and, inside the success handler, access the filesystem root.

```
window.requestFileSystem(LocalFileSystem.PERSISTENT, 0,  
  
    function(fileSystem) {  
  
        var root = fileSystem.root;  
  
        }, this.onError);
```



You are requesting a 0 bytes quota because you are just reading a file; you need to specify a quota only when writing to the device filesystem.

- 6.** Once you get access to the filesystem `root` you can create a `DirectoryReader` object in the success handler and start to explore the `root` filesystem using the `readEntries` asynchronous method of the object.

```
var reader = root.createReader();  
  
reader.readEntries(function(entries){  
  
    for (var i = 0, entry; entry = entries[i]; i++){  
  
        // Here The logic to check if the file is an image  
  
    }  
  
}, app.onError);
```

- 7.** In order to determine if a file is an image in the `for` loop you can first check the `isFile` property of the entry and then use a simple regular expression; when the condition is met you access the file using the `getFile` method of the root `DirectoryEntry` object specifying the success and the failure handlers.

```
if (entry.isFile && (/\.gif|jpg|jpeg|png$/.i).test(entry.name)) {

    root.getFile(entry.name, {create: false},
        app.onGetFile, app.onError);
    break;

}
```

- 8.** In the `index.js` file, always define the `onGetFile` function and in its body access the real file by using the `file` method of the `FileEntry` object. Once you get access to the file, specify the `onload` and `onerror` handlers and read the file using the `readAsDataURL` method in order to assign the result as the `src` attribute of the `img` tag.

```
onGetFile: function(fileEntry) {

    fileEntry.file(function(file) {

        var reader = new FileReader();

        reader.onload = function(evt) {

            var img = document.querySelector('#firstImage');
            img.src = evt.target.result;

        };

        reader.onerror = function(evt) {

            navigator.notification.alert(evt.target.error.
                code, null);

        };

        reader.readAsDataURL(file);

    }, app.onError);

}
```

- 9.** Always in the `index.js` file and in the `deviceready` handler add a call to the `app.requestFileSystem` function. Now test the project on a real device.

What just happened?

You explored the filesystem of the device and rendered the first image found as a Base64 data stream in your app. Now that you are somewhat familiar with the File API, it's time to learn how to transfer files from and to a device.

Transferring files

The PhoneGap File API also includes the `FileTransfer` object. As the name suggests, this object allows you to develop apps able to download and upload files over the Internet. The methods exposed by the `FileTransfer` object are self-explanatory: `upload`, `download`, and `abort`.

The `upload` method accepts several arguments: the path of the file on the device, a URL to receive the file, the success and the failure handlers, an option object, and a Boolean to force the method accepting all the security certificates. (I omitted the Boolean in the next snippet because using it for production is not recommended; an app should accept only the protocols it was designed to deal with.)

```
var fileTransfer = new FileTransfer();
fileTransfer.upload(fileURI, URL, onSucces, onError, options);
```

The `options` argument is a `FileUploadOptions` object. This object allows you to provide additional information using the following properties:

- ◆ `chunkedMode`, a Boolean value that indicates if the streaming of the HTPP request is performed without internal buffering. (For a more detailed description of the chunked transfer encoding you can refer to http://en.wikipedia.org/wiki/Chunked_transfer_encoding).
- ◆ `fileKey`, a string that indicates the name of the form element under which the file is uploaded to the server; the default value is 'file'.
- ◆ `fileName`, a string that represents the name of the uploaded file; the default value is 'image.jpg'.
- ◆ `mimeTye`, a string representing the MIME type of the file that will be uploaded; by default the value is 'image/jpg'.
- ◆ `params`, an object that represents key/value pairs to be included in the HTTP request header.

The `onSuccess` handler receives a `FileEntry` object as an argument so you can immediately access information such as the file name and full path on the device. The `onError` handler receives a `FileTransferError` object, the properties of this object are as follows:

- ◆ `code`, a number that represents one of the four possible error codes stored in the `FileTrasnferError` pseudo constants (i.e., `FILE_NOT_FOUND_ERR`, `INVALID_URL_ERR`, `CONNECTION_ERR`, `ABORT_ERR`).
- ◆ `source`, a string representing the URI to the source file.
- ◆ `target`, a string representing the URI to the target file.
- ◆ `http_status`, a number representing the HTTP status code.

The download method works similarly; the only differences are that the first two arguments are switched and are respectively the URL to download the file and the system URI (i.e., path) to use to store it on the device; also, the options parameter accepts only HTTP headers.

```
var fileTransfer = new FileTransfer();
fileTransfer.download(URL, filePath, onSucces, onError, options);
```

The abort method can be used to stop a download or an upload operation, once the `onError` handler is called, and the value of the `code` property of its argument is the pseudo constant `FileTransferError.ABORT_ERR`.



Also, when you set up the wrong path of the file to download, the `code` property of the `FileTrasnferError` object is equal to `FileTransferError.FILE_NOT_FOUND_ERR` (i.e., the value 1).

Only the `onprogress` property is defined on the `FileTrasnfer` object. As the name suggests, this property is used to store a function called whenever a chunk of data is transferred from or to the device.

Next, in order to put into practice what you just learned, you will download a file, show the download progress, and add a link to the file once the download is completed as shown in the following screenshot:



Time for Action – downloading and saving a file

Get ready to download a file and display in the PhoneGap default app template a progress bar and a link to the file. Refer to the following steps:

1. Open the command-line tool and create a new PhoneGap project named DownloadFile.

```
$ cordova create ~/the/path/to/your/source/downloadfile com.  
gnstudio.pg.DownloadFile DownloadFile
```

2. Add the FileTransfer API plugin using the command line.

```
$ cordova plugins add https://git-wip-us.apache.org/repos/asf/  
cordova-plugin-file-transfer.git
```

- 3.** Go to the www folder, open the index.html file, and add a progress tag with the id value progress inside the main div element of the app below the deviceready tag; assign 1 to the value attribute and 100 to the max attribute.

```
<progress id='progress' value='1' max='100'></progress>
```

- 4.** Go to the www/js folder, open the index.js file, and define a new function named requestFileSystem and request access to the filesystem.

```
requestFileSystem: function () {  
  
    var size = 1024 * 1024 * 5;  
    window.requestFileSystem(LocalFileSystem.PERSISTENT, size,  
        function(fileSystem) {  
  
            // FileTransfer will go here  
  
        }, app.onError);  
  
}
```

- 5.** Define the error handler and, in order to clearly get the code of every possible error, show an alert using the notification object.

```
onError: function(error) {  
  
    navigator.notification.alert(error.code, null);  
  
}
```

- 6.** Once you get access to the filesystem, create a new FileTransfer object and call the download method specifying the remote URL, the system filepath URI, and the success failure handlers.

```
var fileTransfer = new FileTransfer();  
var url = 'http://s3.amazonaws.com/mislav/Dive+into+HTML5.pdf';  
  
fileTransfer.download(url,  
    fileSystem.root.fullPath + '/html5.pdf',  
    app.fileDownloaded,  
    app.fileTransferError);
```

- 7.** Before moving to the success handler, let's define the progress handler; in order to show the download progress, all you have to do is store a function in the `FileTransfer` object's `onprogress` property and use the `loaded` and `tot` properties of the `ProgressEvent` object to determine the percentage of the file already downloaded.

```
fileTransfer.onprogress = function(evt) {  
  
    if (evt.lengthComputable) {  
  
        var tot = (evt.loaded / evt.total) * 100;  
  
        var element = document.querySelector('#progress');  
        element.value = Math.round(tot);  
  
    }  
};
```

- 8.** Define the success handler and, in its body, add the JavaScript needed to render a link in the app.
- 9.** Go to the `www/js` folder, open the `index.js` file, and in the `deviceready` handler add a call to the `app.requestFileSystem` function. Now run your project on a real device.

What just happened?

You initiated a file download, displayed a progress bar, and rendered a link to the file. You will notice a problem because most platforms don't provide a PDF reader inside the `WebView`. In short, the user will not be able to read the file, neither in the app nor in the external browser. In order to open a native app to read the file, you have to use an external plugin. You will discover in the next chapters how to integrate a plugin in your app and how to solve this problem.

Pop quiz – working with files

Q1. How much storage space do you have to reserve when reading a file?

1. 1 KB
2. 0 KB
3. 1 MB

Q2. Is the FileTrasfer API part of the File API?

1. No
2. Yes
3. Only in PhoneGap 2.x

Have a go hero – uploading files

Using the FileTransfer plugin, create an app the user can use to upload images and documents on his/her own server.

Summary

In this chapter you learned how to manipulate files on a device. At this point, you should be able to store information and to read files on a device. In the next chapter, you will learn how to access the Camera and Capture APIs, how to apply nice effects on user pictures, and then upload them to a server using what you learned so far.

10

Capturing and Manipulating Device Media

Mobility and portability of media are key points in our new social media life. The capability to access the device's media enables us to create apps that integrate different media in the communication flow. Using the device Camera your app can capture pictures and videos or access to the existing ones. Through the device's default audio record application the app can capture or play audio tracks.

In this chapter you will:

- ◆ Learn about the Camera API and how to configure the way the app accesses device pictures
- ◆ Understand the differences between the PhoneGap Camera and Capture APIs
- ◆ Learn how to control the position of the camera roll dialog box on an iPad
- ◆ Learn how to use the PhoneGap Capture API to access existing pictures, video, and audio files or to get a new one through the default device applications
- ◆ Learn how to integrate HTML5 and JavaScript to manipulate pictures in your native app

Camera API or Capture API?

The PhoneGap framework implements two different APIs to access media in a device: the Camera API and the Capture API. The main difference between these APIs is that the Camera API can access only the default device camera application whereas the Capture API can also record audio or video using the default audio and video recording application. Another important difference is that the Capture API allows multiple captures with a single API call.



The Capture API is an implementation of an abandoned W3C standards draft. As you can see there are several similarities between the draft and the actual PhoneGap implementation.



Accessing the camera using the Camera API

The Camera API provides access to the device's camera application. This means that an app can get a picture or get access to a media file stored in the photo library and in the albums the user created on the device. The Camera API exposes the following two methods defined in the `navigator.camera` object:

- ◆ `getPicture`, which opens the default camera application or lets the user browse the media library depending on the options specified in the `configuration` object the method accepts as argument.
- ◆ `cleanup`, which cleans up the `image` file stored into the temporary storage location (supported only on iOS).

The `getPicture` method accepts as arguments a success handler, a failure handler, and optionally an object used to specify several camera options through its properties:

- ◆ `quality`, a number between 0 and 100 used to specify the quality of the saved image.
- ◆ `destinationType`, a number used to define the format of the value returned in the success handler. The possible values are stored in the following `Camera.DestinationType` pseudo constants:
 - `DATA_URL(0)` : indicates that the `getPicture` method will return the image as a base64-encoded string
 - `FILE_URI(1)` : indicates that the method will return the file URI
 - `NATIVE_URI(2)` : indicates that the method will return a platform-dependent file URI (for example, `assets-library://` on iOS or `content://` on Android)

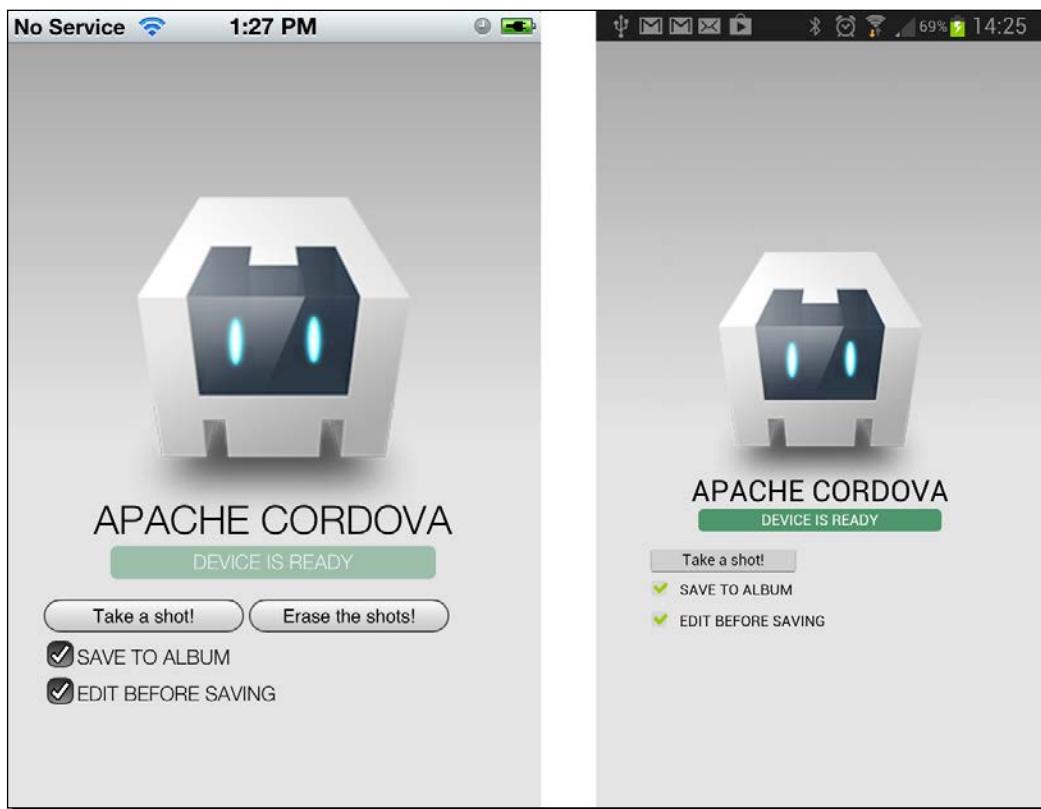
- ◆ `sourceType`, a number used to specify where the `getPicture` method will get an image. The possible values are stored in the `Camera.PictureSourceType` pseudo constants `PHOTOLIBRARY` (0), `CAMERA` (1), and `SAVEDPHOTOALBUM` (2):
 - `PHOTOLIBRARY`: This pseudo constant indicates that the method will get an image from the device's library
 - `CAMERA`: This indicates that the method will grab a picture from the camera
 - `SAVEDPHOTOALBUM`: This indicates that the user will be prompted to select an album before picking a an image
- ◆ `allowEdit`, a Boolean value (the value is `true` by default) used to indicate that the user can make small edits to the image before confirming the selection; it works only in iOS.
- ◆ `encodingType`, a number used to specify the encoding of the returned file. The possible values are stored in the `Camera.EncodingType` pseudo constants `JPEG` (0) and `PNG` (1).
- ◆ `targetWidth` and `targetHeight`, the width and height in pixels to which you want the captured image to be scaled; it's possible to specify only one of the two options. When both are specified, the image will be scaled to the value that results in the smallest aspect ratio (the aspect ratio of an image describes the proportional relationship between its width and its height).
- ◆ `mediaType`, a number used to specify which kind of media files have to be returned when the `getPicture` method is called using as `sourceType` the `Camera.PictureSourceType.PHOTOLIBRARY` or the `Camera.PictureSourceType.SAVEDPHOTOALBUM` pseudo constant; the possible values are stored in the `Camera.MediaType` object as pseudo constants and are `PICTURE` (0), `VIDEO` (1), and `ALLMEDIA` (2).
- ◆ `correctOrientation`, a Boolean value that forces the device camera to correct the device orientation during the capture.
- ◆ `cameraDirection`, a number used to specify which device camera has to be used during the capture. The values are stored in the `Camera.Direction` object as pseudo constants and are `BACK` (0) and `FRONT` (1).
- ◆ `popoverOptions`, an object supported on iOS to specify the anchor element location and arrow direction of the popover used on an iPad when selecting images from the library or album.
- ◆ `saveToPhotoAlbum`, a Boolean value (the value is `false` by default) used in order to save the captured image in the device default photo album.

Capturing and Manipulating Device Media

The success handler receives an argument that contains the URI to the file or the data stored in the file Base64-encoded string depending on the value stored in the `encodingType` property of the `options` object. The failure handler receives as an argument a string containing the device's native code error message.

Similarly, the `cleanup` method accepts a success and a failure handler. The only difference between the two is that the success handler doesn't receive any argument. The `cleanup` method is supported only on iOS and can be used when the `sourceType` property value is `Camera.PictureSourceType.CAMERA` and the `destinationType` property value is `Camera.DestinationType.FILE_URI`.

In order to put into practice what you learned about the Camera API, let's create an app that can get a picture from the device's default camera application and eventually delete it. I just mentioned that the `cleanup` method works only in iOS. This means that your app should look and work differently depending on the target platform. As you can see in the following screenshot, the **Erase the shots!** button is not available on Android:



When creating a new project with the Cordova command-line tool, a folder named `merges` is created in the root of the project. This folder contains a separate folder for each platform you add to the project; the `root` folder of a PhoneGap project looks as follows:

```

|-merges
|---android
|---ios
|-platforms
|---android
|---ios
|-plugins
|-www
|---css
|---img
|---js

```

When you have to handle different user interface elements or business logic for a specific target platform, you can place the files you want to merge in the `merges/TARGET_PLATFORM` folder. In the next example you will create the `index.html` and `index.js` files specifically targeted for the iOS platform.

Time for action – accessing the device camera

Get ready to access the device's camera, show the user the captured picture, and allow the user to erase the selection (in iOS only). Refer to the following steps:

1. Open the command-line tool and create a new PhoneGap project named `AccessCamera`.

```
$ cordova create ~/the/path/to/your/source/camera com.gnstudio.pg.AccessCamera AccessCamera
```

2. Add the Camera API plugin using the command line.

```
$ cordova plugins add https://git-wip-us.apache.org/repos/asf/cordova-plugin-camera.git
```

3. Using the command-line tool, add the Android and the iOS platforms to the project.

```
$ cordova platforms add android
$ cordova platforms add ios
```

4. Go to the www folder, open the index.html file, and add a button tag with the id #getPicture, two input tags with attribute type equal to checkbox, and an img tag with the id value #shot inside the main div of the app below the #deviceready tag; the id values of the two input tags matches exactly to the properties saveToPhotoAlbum and allowEdit of the optional configuration argument of the getPicture method.

```
<button id='getPicture'>Take a shot!</button> <br>
<input id='saveToPhotoAlbum' checked type='checkbox'>
    Save to Album
</input> <br>
<input id='allowEdit' checked type='checkbox'>
    Edit before saving
</input> <br>
<img id='shot' />
```

5. Go to the www/js folder, open the index.js file, define a new property named cameraOptions, and then use it to store an object with the options you want to use when getting a picture (for example., saveToPhotoAlbum and allowEdit between the others).

```
cameraOptions: {targetWidth: 300, targetHeight: 400,
    saveToPhotoAlbum: true, allowEdit: true}
```

6. In the body of the deviceready function, define a listener for the #getPicture button for the two checkboxes.

```
var element;

element = document.querySelector('#getPicture');
element.addEventListener('touchstart', app.takePicture);

element = document.querySelector('#saveToPhotoAlbum');
element.addEventListener('change', app.updatePreferences);

element = document.querySelector('#allowEdit');
element.addEventListener('change', app.updatePreferences);
```

7. Define the takePicture function and in its body access the device camera using the getPicure method.

```
takePicture: function(evt) {

    evt.preventDefault();

    navigator.camera.getPicture(app.onCameraSuccess,
        app.onCameraError,
        app.cameraOptions);

}
```

- 8.** Define the success handler passed as argument to the `getPicture` method and in its body set up the source of the `img` tag previously added to the `index.html` file.

```
onCameraSuccess: function(imageData) {  
  
    document.querySelector('#shot').src = imageData;  
  
}
```

- 9.** Define the error handler that will notify you when the code throws an error.

```
onCameraError: function(error) {  
  
    navigator.notification.alert(error, null);  
  
}
```

- 10.** Define the `updatePreferences` function in order dynamically to update the `cameraOptions` object using the id of the selected checkbox and its status.

```
updatePreferences: function(evt) {  
  
    evt.preventDefault();  
    app.cameraOptions[evt.target.id] = evt.target.checked;  
  
}
```

- 11.** Now that the Android version is completed, create a copy of the `index.html` and `index.js` files in the folder `merges/ios`, retaining the same folder structure of the `www` folder.

- 12.** Go to the `merge/ios` folder, open the `index.html` file, and add a `button` tag with the `id` value `#cleanpictures` below the `button` tag previously added.

```
<button id='cleanpictures'>Erase the shots!</button>
```

- 13.** Open the `index.js` file and add in the body of the `deviceready` function the definition of an event handler for this new `button` tag.

```
element = document.querySelector('#cleanpictures');  
element.addEventListener('touchstart', app.cleanPicture);
```

- 14.** Define the `cleanPicture` function; in the body of the function, call the `cleanup` method defining two inline event handlers in order to get notified about the result.

```
cleanPicture: function(evt) {  
  
    evt.preventDefault();  
  
    navigator.camera.cleanup(function(message) {  
  
        navigator.notification.alert(message, null);  
  
    },  
    function(errorMessage) {  
  
        navigator.notification.alert(errorMessage, null);  
  
    }) ;  
  
}
```

- 15.** Open the command-line tool and launch the `prepare` command, and then the build one from the root of the project.

```
$ cordova prepare  
$ cordova compile
```



The commands `prepare` and `compile` can be executed from any folder of the project. The command `build` is a shorthand for the commands `prepare` and `compile`.



- 16.** Run the project on a real device for each target platform (unfortunately emulators don't support the camera).

What just happened?

You accessed the device camera and handled the different implementation of the Camera API on Android and iOS using the merge feature of the Cordova command-line tool. If you go to the platform-specific folders (`platforms/android/assets/www` and `platforms/ios/www`) and open the `index.js` file, you will see that they differ from each other.

Controlling the camera popover

The method `getPicture` in iOS (and specifically on iPad) returns a `CameraPopoverHandle` object when the `sourceType` property value is one of the following pseudo constants defined in the `Camera.PictureSourceType` object: `SAVEDPHOTOALBUM` or `PHOTOLIBRARY`. Using this object it's possible to control the position of the popover dialog box created when the `getPicture` method is called.

The `CameraPopoverHandle` object exposes only the `setPosition` method that requires a `CameraPopoverOptions` object as argument. This object allows you to specify the coordinates, the dimensions, and the position of the arrow of this dialog box.

```
var popoverOptions = new CameraPopoverOptions();
popoverOptions.x = 220;
popoverOptions.y = 600;
popoverOptions.width = 320;
popoverOptions.height = 480;
popoverOptions.arrowDir = Camera.PopoverArrowDirection.ARROW_DOWN;
```

You can reach the same result with a more compact syntax, specifying the properties in the constructor of the `CameraPopoverOptions` object.

```
var popoverOptions = new CameraPopoverOptions(220, 600, 320, 480,
Camera.PopoverArrowDirection.ARROW_DOWN);
```

It's also possible to specify coordinates, position, and arrow direction using the `popoverOptions` property of the `cameraOptions` object.

```
cameraOptions.popoverOptions = {

    x : 220,
    y : 600,
    width : 320,
    height : 480,
    arrowDir : Camera.PopoverArrowDirection.ARROW_DOWN

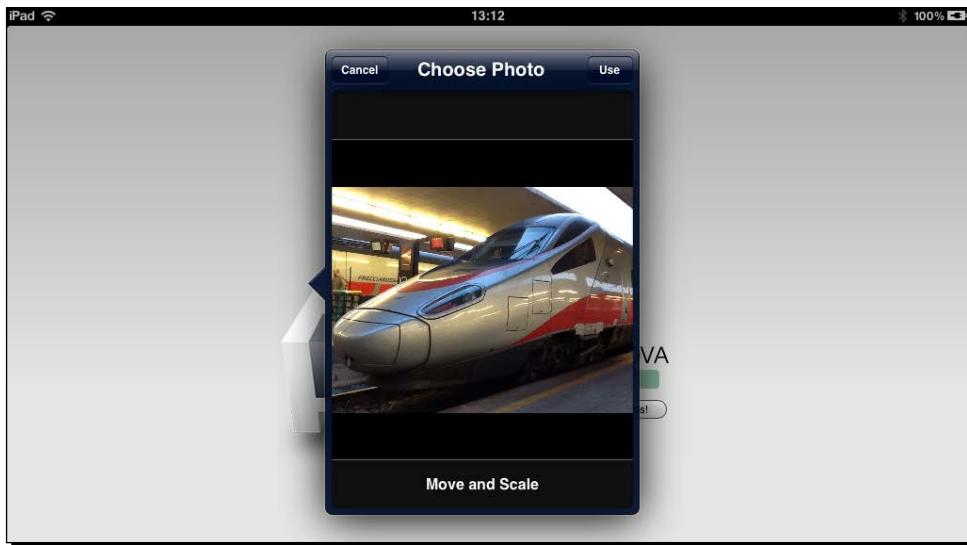
};
```

The pseudo constants defined in the `PopoverArrowDirection` object match the native iOS constants defined in the `UIPopoverArrowDirection` class.

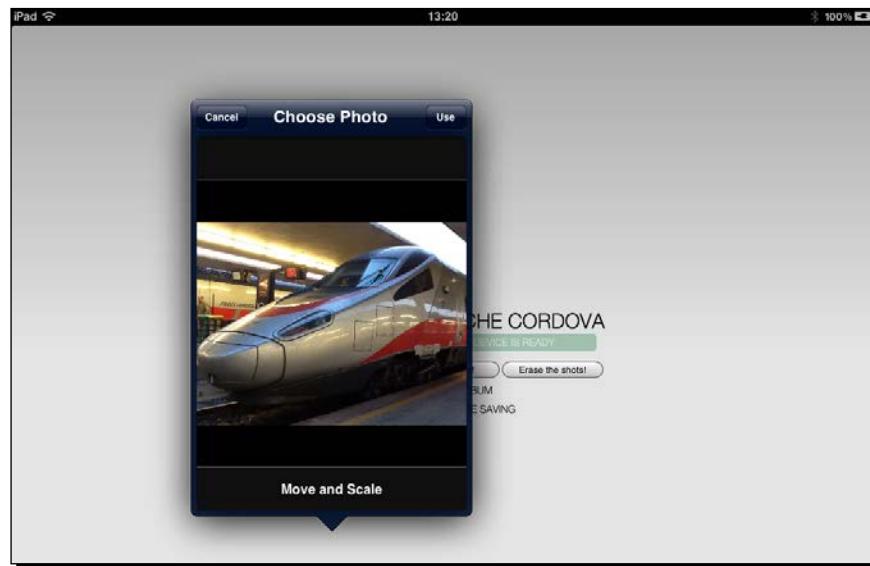
```
Camera.PopoverArrowDirection = {
    ARROW_UP: 1,
    ARROW_DOWN: 2,
    ARROW_LEFT: 4,
    ARROW_RIGHT: 8,
    ARROW_ANY: 15
};
```

Capturing and Manipulating Device Media

This option is extremely useful when you want to have control over the position of the dialog box when the orientation of the device changes. In the following example, you will control the position and size of the dialog box in order to override the default behavior as shown in the screenshots that will follow. The following screenshot shows the default position of the dialog box:



In the following screenshot, the position differs due to the options defined when opening the dialog box:



If you want to use different iOS devices (for example, an iPhone 5 and 4S to test different screen sizes) for development, you have to create a separate provisioning file and add it to the device. In order to perform this task, you have to:



- ◆ Read the device ID by clicking on the serial number you get on the first screen you get in iTunes when connecting the device.
- ◆ Go to the Apple developer portal, log in, and add a new development device to your account.
- ◆ Create and download a new provisioning file (you should already have a certificate available; otherwise read the information available in the Apple Developer portal at <https://developer.apple.com/support/technical/certificates/>).
- ◆ Add the provisioning file to your device using the Xcode Organizer window (in **Xcode Window | Organizer**).

Time for action – controlling the position of the camera roll

Use the following steps to change the position of the default camera roll dialog box on an iPad:

1. Go to the `merges/ios/js` folder of the PhoneGap project you previously created, open the `index.js` file, and add a new function named `initAdditionalOptions`.

```
initAdditionalOptions: function() {
    // Additional options will be defined here
}
```

2. In the body of the `initAdditionalOptions` function, specify the device photo album as source of the `getCamera` method as well as the size and position of the popover dialog box.

```
app.cameraOptions.sourceType = Camera.PictureSourceType.
    SAVEDPHOTOALBUM;

app.cameraOptions.popoverOptions = new CameraPopoverOptions(220,
    600, 320, 480, Camera.PopoverArrowDirection.ARROW_DOWN);
```

3. Add a call to the `initAdditionalOptions` function at the end of the `deviceready` event handler.

```
deviceready: function() {  
    // All the events handling initializations are here  
    app.initAdditionalOptions();  
  
}
```



When you refer to PhoneGap-specific objects such as the `CameraPopoverOptions` you have to defer the use of this object after the `deviceready` event is fired.

4. Open the command-line tool and run the `prepare` and `compile` commands (or the `build` one) in order to test the project on a real device.

What just happened?

You handled the size and position of a default dialog box on iOS on an iPad using only JavaScript.

The Capture API

Modern devices offer to the user a huge range of media capabilities; right now people can register a video, record some audio, take a picture, and use all of these media in their communication flow.

The Capture API works asynchronously as most of the PhoneGap APIs and provides access to the audio, image, and video capture capabilities of the device. In order to start working with this API you have to access the `capture` object stored in the `navigator.device` object.

```
var capture = navigator.device.capture;
```

Once you get access to the `capture` object, it's possible to detect which video, audio, and image formats are supported by the device through the following properties:

- ◆ `supportedAudioModes`
- ◆ `supportedImageModes`
- ◆ `supportedVideoModes`

Each property returns an array of `ConfigurationData` objects, each item of the array represents a supported media type. There are three properties defined in the `ConfigurationData` object you can use to clearly identify the media types supported by the device:

- ◆ `type`, a lower case string representing the supported media type following the RFC2046 standard explained at <http://www.ietf.org/rfc/rfc2046.txt> (i.e., `video/3gpp`, `video/quicktime`, `image/jpeg`, `audio/amr`, etc.).
- ◆ `height`, a number that represents the height of the supported image or video in pixels (the property returns 0 when the object represents a supported audio format).
- ◆ `width`, a number that represents the width of the supported image or video in pixels (the property returns 0 when the object represents a supported audio format).



At the time of writing the `ConfigurationData` object is not implemented in any supported platform. In fact, each of the arrays stored into the `supportedAudioModes`, `supportedImageModes`, and `supportedVideoModes` properties is empty.

The `capture` object exposes three methods in order to access the video, audio, and image capture capabilities of the device:

- ◆ `captureVideo`
- ◆ `captureAudio`
- ◆ `captureImage`

These methods have the same syntax; each one accepts as arguments a success handler, a failure handler, and an option object. The success handler is invoked upon a successful media capture operation and receives as argument an array of `MediaFile` objects describing each captured file. The error handler is invoked if an error occurs during a media capture operation or when the user cancels the operation and receives as an argument a `CaptureError` object. The following snippet captures an image using the device camera:

```
var capture = navigator.device.capture;

capture.captureImage(function(files) {
    console.log(files);
}, function(error) {
    console.log(error);
});
```

The `MediaFile` object stored in the `files` array returned by the success handler describes the captured media. The properties of the `MediaFile` object are:

- ◆ `fullPath`, a string representing the file path on the device including the file name.
- ◆ `lastModifiedDate`, the modification date of the file expressed as the number of milliseconds since January 1, 1970 (refer to the online reference for more information about the `Date` object in JavaScript https://developer.mozilla.org/en/docs/JavaScript/Reference/Global_Objects/Date).
- ◆ `name`, a string representing the name of the file. The name is composed by the `lastModificationDate` value and the file extension.
- ◆ `size`, a number representing the size of the file in bytes.
- ◆ `type`, a string representing the mime type of the captured file (i.e., 'image/jpeg').

The `CaptureError` object returned to the error handler only has one property, `code`. The property contains an integer equal to one of the following pseudo constants defined in the `CaptureError` object:

- ◆ `CaptureError.CAPTURE_INTERNAL_ERR` (returned value 0), the device failed to capture a video, an image, or sound
- ◆ `CaptureError.CAPTURE_APPLICATION_BUSY` (returned value 1), the capture application is currently serving an other capture request
- ◆ `CaptureError.CAPTURE_INVALID_ARGUMENT` (returned value 2), the app is using invalid arguments when invoking the API (for example, the `limit` parameter has a value of less than 1)
- ◆ `CaptureError.CAPTURE_NO_MEDIA_FILES` (returned value 3), the user exited the camera application or the audio capture application before capturing anything
- ◆ `CaptureError.CAPTURE_NOT_SUPPORTED` (returned value 20), the requested capture operation is not supported

The option object varies for each method. In fact, the Capture API defines a different object for each kind of capture: `CaptureVideoOptions`, `CaptureAudioOptions`, and `CaptureImageOptions`. All of these objects have in common the same properties, `limit` and `mode`; the `duration` property is defined only in the `CaptureVideoOptions` and `CaptureAudioOptions` objects. The default value of the `limit` property is 1 and it's used to specify the number of captures the user can do before returning to the app. The `duration` property is the maximum length of a capture in seconds. The `mode` property represents the selected video or audio mode.



Support for the configuration options is very fragmented. For instance, the `limit` property of the `CaptureImageOptions` and `CaptureVideoOptions` objects is not supported in iOS. Refer to the online documentation to check the actual status of the implementation at http://docs.phonegap.com/en/edge/cordova_media_capture_capture.md.html#Capture.

One of the features of the Itinero open source app distributed with this book is to take a picture or to select one from the gallery and to create a relationship between a day of the trip and the image. Among other options, the end user can add some text to the image and apply some filters to a picture in order to make it look nicer.



The most efficient way to manipulate an image is through native code; however, you can also perform simple image manipulations via JavaScript. In order to avoid overcomplicating the following example, I extracted the relevant code from the Itinero app source code.

Time for action – manipulating images with a canvas

Get ready to apply a sepia effect to an image acquired using the Capture API. Execute the following steps:

1. Open the command-line tool and create a new PhoneGap project named ImageEffect.

```
$ cordova create ~/the/path/to/your/source/imageEffect com.  
gnstudio.pg.ImageEffect ImageEffect
```

2. Add the Capture API plugin using the command line.

```
$ cordova plugins add https://git-wip-us.apache.org/repos/asf/  
cordova-plugin-media-capture.git
```

3. Add to the existing markup a canvas tag with the id value #manipulatedImage in order to use it to render the manipulated image.

```
<canvas id='manipulatedImage' />
```

4. Once the deviceready event has been fired, access the device camera, allowing the user to get only one image.

```
var capture = navigator.device.capture;  
capture.captureImage(onGetImage, onImageError, {limit: 1});
```

5. Define the success handler in the index.js file and access to the file information stored in the array returned as an argument.

```
onGetImage: function(files){
```

```
    var currentFile = files[0];  
  
    // Canvas access logic will go here  
}
```

6. Get access to the canvas and store a reference to the 2d context of the canvas in order to be able to draw content on it.

```
var canvas = document.querySelector('#manipulatedImage');  
var context = canvas.getContext('2d');  
  
// Image object definition and load will go here
```

- 7.** Create an `image` object, assign a handler to the `onload` property, and define the `src` property of the object using the `fullPath` property of the `MediaFile` object.

```
var image = new Image();  
  
image.onload = function(evt) {  
  
    // The image manipulation logic will go here  
  
};  
  
image.src = currentFile.fullPath;
```

- 8.** In the function stored in the `onload` property draw the image on the canvas, get the pixels, manipulate them, and reassign the pixels to the canvas.

```
var width = canvas.width;  
var height = canvas.height;  
  
context.drawImage(this, 0, 0, width, height);  
  
var imgPixels = context.getImageData(0, 0, width, height);  
context.putImageData(grayscale(imgPixels), 0, 0, 0, 0, width,  
height);
```



You can find the grayscale function and several image effects at <http://www.html5rocks.com/en/tutorials/canvas/imagefilters/>.

What just happened?

You applied an effect over an image using only client-side code in a native app. This is one of the key strengths of PhoneGap: you can be productive and create sophisticated apps without a deep knowledge of the target platform.

Pop quiz – getting started with media

Q1. What is the difference between the Camera and Capture APIs?

1. The Camera API is deprecated.
2. The Capture API works only with PhoneGap 3.x.
3. The Camera API can capture one media per time while the Capture can capture more than one.

Q2. The Capture API can capture audio?

1. No.
2. Yes.
3. It depends on the target platform.

Have a go hero – applying effects

Starting from the last example, add a radio button and let the user select the effect he/she wants to apply to the image.

Summary

In this chapter, you learned how to access the device camera and all the other capturing tools available in the device. In the next chapter, you will learn how to extend PhoneGap using native code, and you will see how to integrate existing plugins in your app.

11

Working with PhoneGap Plugins

Using web standards and JavaScript to build a native app may have its limitations because apps developed using native code can interact deeply with the operating system. This is only partially true when dealing with PhoneGap apps because its architecture allows developers to extend the framework capabilities with the help of custom plugins.

In this chapter you will:

- ◆ Learn what a PhoneGap plugin is and how to install and configure the plugins you want to use in your project
- ◆ Understand how to manage project plugins and their dependencies using Plugman
- ◆ Discover the components needed to create a custom plugin
- ◆ Learn how to create a custom plugin from the existing source code
- ◆ Discover how to implement push notifications in a PhoneGap-based app

Introduction to plugins

In order to be productive quickly with PhoneGap plugins, it's important to keep in mind how the framework works. A PhoneGap app consists of three main layers:

- ◆ The user interface, developed using HTML, CSS, and JavaScript
- ◆ The business logic, developed in JavaScript
- ◆ The PhoneGap framework, native code exposed to the business logic through a JavaScript API

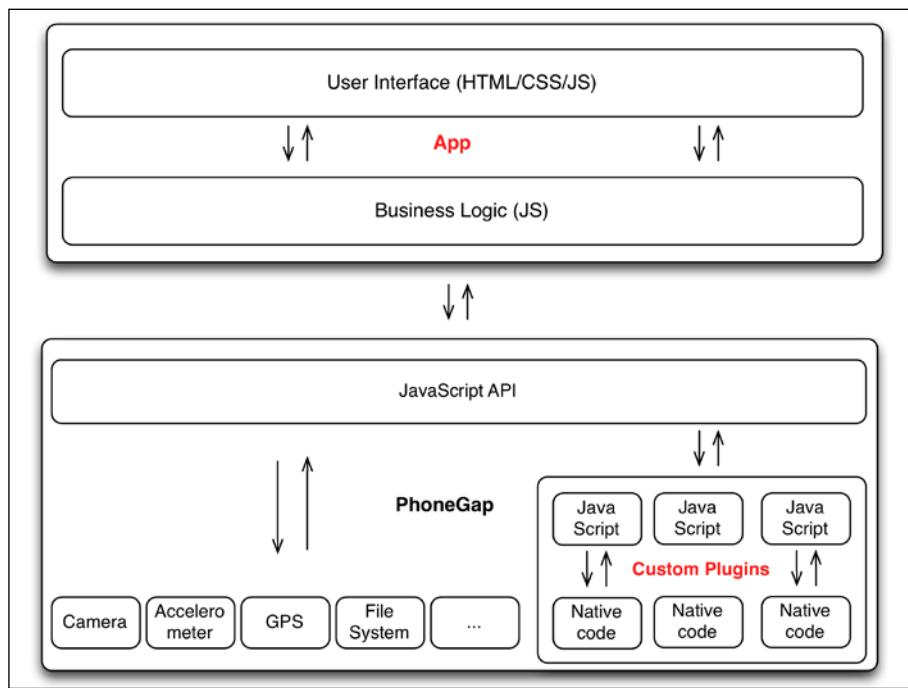


Since Version 3.0, all the PhoneGap APIs have been converted to plugins, which means having a deep understanding of the plugins is now even more important for a PhoneGap developer.

The user interface and the business logic are the app's main source code and are the parts on which most developers concentrate their development efforts. The custom plugins are strictly integrated in the framework and are exposed to the user through JavaScript. PhoneGap provides a callback architecture that allows a developer to design a custom plugin and extend its functionalities. You can imagine a plugin as an additional component of the framework that works like all the PhoneGap APIs: it acts as a bridge between the WebView and the native platform on which the app is running.



All the PhoneGap APIs that you have learned so far rely on the same architecture so you can safely consider them as native plugins.



When you plan to write a custom plugin, keep in mind that you have to write the native code from scratch for all the platforms you want to support. Before starting to write a plugin you should check if what you are searching for has been implemented already at <https://github.com/phonegap/phonegap-plugins>. This repository is not a comprehensive list but the community around PhoneGap is working hard to keep it up-to-date.

Getting started with plugins

In order to use the features implemented in a custom plugin, you have to install it in your project. Depending on the target platform this involves different steps; most of the time, however, the accompanying `readme.md` file available in the plugin repository will help.

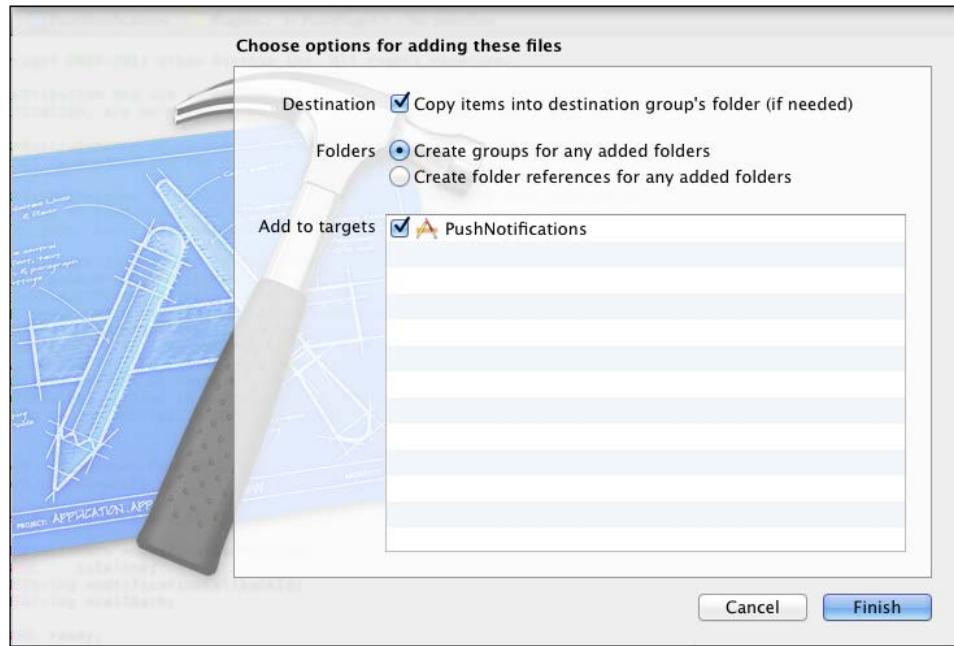
When working on the Android platform the first step is to copy the plugin source code to the `platforms/android/src` folder of the project and register the plugin by adding it to the `config.xml` file stored in the folder `platforms/android/res/xml`. The `config.xml` file contains several other settings and all the plugins, as an XML node that indicates the name and the package of the plugin:

```
<plugin name='HelloWorld' value='com.gnstudio.pg.HelloWorld' />
```

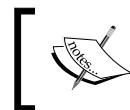
Depending on the type of the plugin, you should also update the app permissions in the `platforms/android/AndroidManifest.xml` file. For instance, when using a plugin that is able to send SMS you have to explicitly grant permission to the app adding the following XML node to the `AndroidManifest.xml` file:

```
<uses-permission android:name="android.permission.SEND_SMS" />
```

In order to add a plugin to the iOS platform, manually copy the plugin source code (the `.h` and `.m` files only) to the Xcode project by dragging the files on the `plugins` folder you find in the Xcode project navigator. When you drop the files, a dialog box pops up; check the option to copy the files into the destination group.



When working with the Windows Phone 8 platform, it's enough to add the source code to the `platforms/windows/plugins` folder and add the plugin to the `config.xml` file stored in the `platforms/windows` folder.



When the plugin includes a JavaScript file, you have to store it in the `www` folder of the target platform and include it in the `index.html` file after the `cordova.js` file inclusion.



It's not an easy task to manually manage the plugins in a PhoneGap project; thankfully there is a command-line tool that makes your life easier.

Using plugins with Plugman

The Apache Cordova Plugman project <http://github.com/apache/cordova-plugman> is an open source command-line utility distributed as an `npm` module to facilitate the installation and uninstallation of plugins. Plugman supports the Android BlackBerry 10 and iOS platforms. The installation process is the same as any other `npm` module; remember that if you install it globally (using the `-g` option) you have to run the command as root:

```
$ npm install plugman -g
```

Once installed, you can use several commands from your command-line tool to get the source code of a plugin, install and uninstall it, and package the plugin to be distributed with your app (Plugman is part of the `cordova-cli` tool, you don't have to install it if you are already using it).

- ◆ `--fetch`, retrieves a plugin from a directory, a Git repository, or by name into the specified `plugins` directory.

```
$ plugman --fetch https://github.com/phonegap-build/GAPPlugin.git  
--plugins_dir PATH_TO_YOUR_PLUGINS_DIR
```

- ◆ `--install`, installs a plugin for a specific target platform in a PhoneGap project. The plugin can be installed by name or by URL.

```
$ plugman --platform android --project PLATFROM_PROJECT_PATH  
--plugin https://github.com/phonegap-build/GAPPlugin.git
```



The `--plugin` argument can be the name of the plugin or the path to a Git repository. By default, Plugman launches the `install` command and fetches the plugin if it doesn't exist in the `plugins` directory. For this reason the `--install` argument is optional.



- ◆ --uninstall, uninstalls by name a previously installed plugin.
`$ plugman --uninstall --platform android --project PLATFORM_PROJECT_PATH --plugin PLUGIN_NAME`
- ◆ --list, lists all the plugins previously fetched using Plugman.
- ◆ --prepare, sets up the plugin, properly injecting the needed JavaScript files and defining the appropriate permissions. The --prepare command is implicitly called when you install or uninstall a plugin.

Plugman is integrated into the Cordova command-line tool so that you can achieve similar results using the `plugin add`, `plugin remove`, and `plugin list` commands. In order to add a plugin to your project, it's enough to run the `add` command from your project folder specifying the path to the plugin. The plugin source files are then copied to the `plugins` folder.

```
$ cordova plugin add PATH_TO_THE_PLUGIN
```

In order to add the plugin to a target platform, you can run the `compile` command or the `prepare` command.

When using Plugman as a standalone utility you can specify variables at install time using the `--variable` argument. Such variables are necessary for plugins requiring API keys or other custom, user-defined parameters.

The anatomy of a plugin

A PhoneGap plugin is a bridge between the `WebView` and the native platform the app is running on. Plugins are composed of a single JavaScript interface used across all platforms and native implementations following platform-specific plugin interfaces that the JavaScript will call into. There are no restrictions about how to develop the JavaScript interface. The only mandatory implementation is the way to communicate between JavaScript and the native environment using the `cordova.exec` function.

```
var arguments = /* Optional arguments will go here */;  
cordova.exec(onSuccess, onError, 'ClassName', 'method', arguments);
```

The `cordova.exec` function accepts as arguments a success handler, a failure handler, the name of the native class (i.e., the name specified in the `config.xml` file previously mentioned), the method to call, and an array with the arguments required by the native code. The success and failure handlers will get back as arguments the parameters returned by the native code.



Due to the asynchronous nature of PhoneGap, most of the time the examples refer to the `ClassName` argument as the service and to the `method one` as the action.



When working with Android as the target platform, you have to be familiar with Java to write a custom plugin. A PhoneGap plugin in fact has to extend the `CordovaPlugin` class and override the `execute` method.

```
@Override  
public boolean execute(String action, JSONArray args, CallbackContext  
callback) {  
  
    // The calls to private methods will be defined here  
    return false;  
}
```

This method accepts the following three arguments:

- ◆ `action`, a string used in order to understand in the native code how to handle the request.
- ◆ `args`, an ordered sequence of values (i.e., a Java `JSONArray`
<http://www.json.org/javadoc/org/json/JSONArray.html>).
- ◆ `callback`, an instance of the `org.apache.cordova.api.CallbackContext` class to use in order to call the success and the failure handlers. It's using the `callback` argument that you will be able to execute functions inside your JavaScript.

When working with iOS as the target platform, you have to be familiar with Objective-C to write a custom plugin. The native part of your plugin will consist of at least two files: a header file (i.e., a `.h` file) and a source file (a `.m` file) which together implement the logic you need. These two files together define a class in Objective-C; more information about Objective-C is available in the online reference at http://developer.apple.com/library/ios/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/.

The plugin interface extends the Cordova `CDV.h` interface and contains the definition of all the public methods you can call from JavaScript.

```
#import <Cordova/CDV.h>

@interface MyPluginClass : CDVPlugin

- (void)someMethod:(CDVInvokedUrlCommand*) command;

@end
```

The implementation of the interface completes the class definition in Objective-C and contains the native code that will be executed through the JavaScript layer.

```
#import 'Plugin.h'
#import 'AppDelegate.h'

@implementation MyPluginClass

- (void)someMethod:(CDVInvokedUrlCommand *)command{

    NSLog(@"YOU ARE READING THIS NATIVELY FROM A PLUGIN");

}
@end
```

When Windows Phone 8 is your target platform, you have to be familiar with C# to create a custom plugin. In fact, the plugin is a C# class that extends the Cordova `BaseCommand` class and implements the public methods that can be executed through the JavaScript layer.

```
using WPPluginClassLib.Cordova;
using WPPluginClassLib.Cordova.Commands;
using WPPluginClassLib.Cordova.JSON;

public class Echo: BaseCommand{

    public void echo(string options)  {

        // The native code to be executed will go here

    }
}
```



All the methods exposed by a Windows Phone plugin must have the same signature: public, returning void, and one argument as a string.

Regardless of the native target platform, it's a good habit to organize the source code of a custom plugin following a precise pattern. The contributors of the Plugman project suggest using a folder structure as follows:

```
| -plugin.xml  
| ---src  
|   | ---android  
|   |   | -CustomPlugin.java  
|   | ---ios  
|   |   | -CustomPlugin.h  
|   |   | -CustomPlugin.m  
|   | ---windows  
|   |   | -CustomPlugin.cs  
| -README.md  
| ---www  
|   | -customplugin.js  
|   | -plugin.png
```

The `plugin.xml` file is the manifest of the plugin; it's where a developer can specify the name of the plugin, the versions of PhoneGap supported by the plugin, the assets to copy in the `www` directory of the PhoneGap project during the installation, the supported platforms, and so on. All this information will be used when installing a plugin using the available command-line tools; if you are interested in a complete list of tags allowed in the `config.xml` file, refer to the online documentation at <https://github.com/apache/cordova-plugman>.

A good plugin is always distributed with a very well-organized JavaScript interface. Take a look at the example at <https://github.com/shazron/KeychainPlugin/blob/master/www/keychain.js>, which exposes the plugin API with a `JavaScript object` and the `prototype` property.

Working with plugins

Using plugins you can extend the PhoneGap framework in order to meet the needs of your app. This means that there are no compelling limitations but it also means that the source code of your app has to be maintained for different platforms. On GitHub, there is a big list of plugins categorized by platform (<https://github.com/phonegap/phonegap-plugins>), each one run on specific versions of PhoneGap. In order to make sure that an existing plugin fits your needs, you have to double-check the compatibility with the PhoneGap version you are using in your project and eventually update the source code to be compliant. One of the strengths of PhoneGap is the continuous release model because it speeds up the release of new features and bug fixing, but it means that a plugin should be maintained in order to meet the deprecation policy of the framework. Refer to the online Wiki for updated information about upcoming deprecations at <http://wiki.apache.org/cordova/DeprecationPolicy>.

The Push Notifications plugin

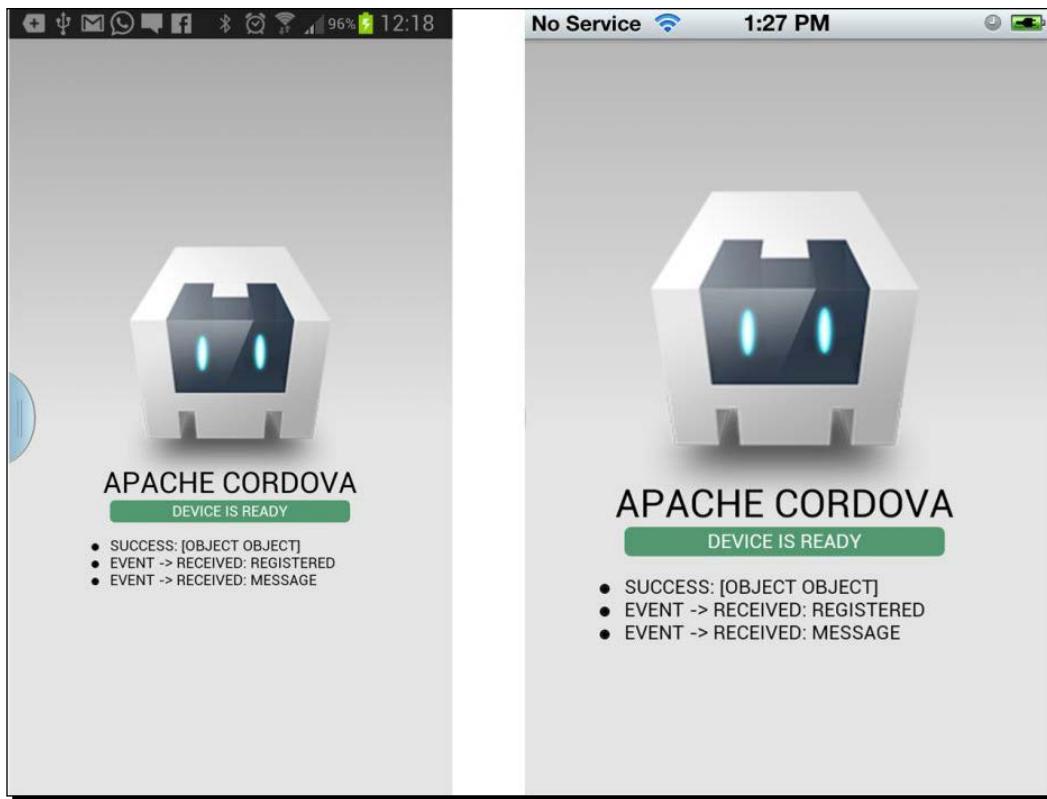
A push notification is a way for an app to send information to your phone (via a badge, alert, or notification bar message) even when the app isn't in use. For example, if you have a sports app with push notifications enabled, that app can send you the latest score of your favorite team even if the app isn't running. The notification will send you a message, and when you touch it, it will direct you back to the app for more information.

Push notifications are a feature added to iOS 3.0 and more recently also available on Android and Windows Phone. There are several online websites that offer cross-platform push notification services such as <http://openpush.im/>, <http://urbanairship.com/products/push-notifications/>, <http://push.io/>, and many others; anyway you can implement your own service using the PhoneGap Push Plugin at <https://github.com/phonegap-build/PushPlugin>.



At the time of writing, there is a pull request on GitHub to fix the Android native code of the plugin in order to run with the last version of PhoneGap, <https://github.com/GiorgioNatali/PushPlugin>.

In the following example, you will see how to implement push notifications in a PhoneGap-based app showing the communication logs on the device screen. Unfortunately, the Push Plugin is not yet compliant with Plugman, so you have to install it manually.



The very first step is to set up the push notifications for each target platform. To enable push notifications in iOS, you need the app to be signed with a provisioning profile that is configured for push. In addition, your server needs to sign its communications to APNS with an SSL certificate. For more information refer to the online guide at <http://developer.apple.com/library/ios/>.

In order to use push notifications on Android, you have to activate the Google Cloud Messaging service and create a server key to use in your app. For more details refer to the Android guide at <http://developer.android.com/google/gcm/gs.html>.

To use your local development environment as a server that uses the Google and Apple notification services, you have to install Ruby gems <http://rubygems.org/>. If you already installed it be sure to update gems using the following command:

```
$ gem install rubygems-update
```

Time for action – using push notifications on Android

Use the following steps to enable cross-platform push notifications on your app:

1. Open the command-line tool and create a new PhoneGap project called pushnotifications.

```
$ cordova create ~/the/path/to/your/source/pushnotifications com.gnstudio.pg.pushnotifications PushNotifications
```

2. Using the command-line tool add the Android and the iOS platforms to the project.

```
$ cd pushnotifications
$ cordova platforms add android
$ cordova platforms add ios
```

3. Clone the Push Plugin GitHub repository to your system.

```
$ git clone https://github.com/phonegap-build/PushPlugin.git
```

4. Open the Xcode project you find in the folder platforms/ios and drag-and-drop into the plugins folder available in the Xcode Project Navigator the following files: AppDelegate+notification.h, AppDelegate+notification.m, PushPlugin.h, PushPlugin.m



All the files are stored in the src/ios folder of the plugin repository.

5. Open the config.xml file you find in the root of the Xcode project and add an entry to enable the plugin.

```
<plugin name='PushPlugin' value='PushPlugin' />
```

6. Change the directory to platforms/android/src and then copy to it the files you find in the src/android folder of the plugin repository.

7. Go to the res/xml folder, open the config.xml file, and add the plugin to the Android project configuration.

```
<plugin name='PushPlugin' value='com.plugin.GCM.PushPlugin' />
```

8. Add the PushNotification.js file you find in the www folder of the plugin repository to your project www folder.

9. Build the app in order to check if the configuration is working properly.

```
$ cordova build
```

- 10.** Go to the www folder, open the index.html file, and add a ul tag with the class logs inside the main div of the app, below the #deviceready one.

```
<ul class='logs'></ul>
```

- 11.** In the same file add, immediately after the inclusion of PhoneGap, the script tag needed to embed the JavaScript interface of the plugin.

```
<script type='text/javascript' src='PushNotification.js'></script>
```

- 12.** Go to the www/js folder, open the index.js file, and add in the body of the deviceready function the script needed to initialize the push notifications on Android and iOS.

```
var pushNotification = window.plugins.pushNotification;

if (device.platform == 'android' || device.platform == 'Android')
{
    pushNotification.register(app.successHandler, app.errorHandler,
    {'senderID': '570783355289', 'ecb': 'app.onNotificationGCM'});
} else {
    pushNotification.register(app.tokenHandler,
    app.errorHandler, {'badge': 'true', 'sound':
    'true', "alert": 'true', 'ecb': 'app.onNotificationAPN'});
}
```

- 13.** Define a function named addLogs in order to append the logs to the device screen.

```
addLogs: function(message, data) {

    var logs = document.querySelector('.logs');

    var log = document.createElement('li');
    log.innerHTML = message + data;

    logs.appendChild(log);
}
```

- 14.** Define the successHandler function to be used in Android to notify the user about the successful registration of the device to the Google Cloud Messaging service.

```
successHandler: function (result) {

    navigator.notification.vibrate(300);
    app.addLogs('success: ', result);

}
```

- 15.** Define the `tokenHandler` function to be used in iOS to notify the user about the successful registration to the Apple Push Notification service.

```
tokenHandler: function (result) {  
  
    navigator.notification.vibrate(300);  
    app.addLogs('token: ', result);  
  
}
```

- 16.** Define the `errorHandler` function used on both platforms that logs an eventual error.

```
errorHandler: function (error) {  
  
    navigator.notification.vibrate(300);  
    app.addLogs('error: ', error);  
  
}
```

- 17.** Define the `onNotificationAPN` function that will be executed in iOS each time a push notification has been received.

```
onNotificationAPN: function (evt) {  
  
    if (evt.alert) {  
  
        navigator.notification.vibrate(300);  
        app.addLogs('EVENT -> RECEIVED: ', evt.alert);  
  
    }  
}
```

- 18.** Define the `onNotificationGCM` function that will be executed in iOS each time a push notification has been received.

```
onNotificationGCM: function (evt) {  
  
    navigator.notification.vibrate(300);  
    app.addLogs('EVENT -> RECEIVED: ', evt.event);  
  
}
```

- 19.** Build the project and install the app on an iOS and on an Android device.

20. Go to the `Example/server` folder of the plugin repository and open the files `pushGCM.rb` and `pushAPNS.rb` in a text editor. In the `pushGCM.rb` file, add the API key you created with Google Cloud Messaging and the device registration ID. In the `pushAPNS.rb` file, add the developer certificate password and the path to the certificate.

21. Open the command-line tool and install the gem `pushmeup`.

```
$ gem install pushmeup
```

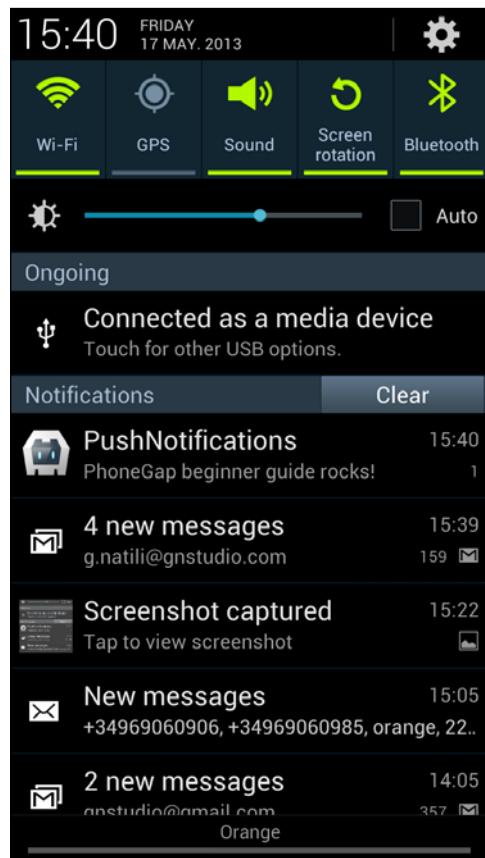
22. Run the Ruby file from your command-line tool.

```
$ ruby pushGCM.rb
```

```
$ ruby pushAPNS.rb
```

What just happened?

You developed an app able to react to a push notification on iOS and Android, and your device should have received another notification when the app is paused.



Pop quiz – getting started with PhoneGap plugins

Q1. How can you write a plugin for multiple platforms?

1. Using JavaScript.
2. Using JavaScript and the target platform native code.
3. Using the PhoneGap build services.

Q2. Is a Plugin asynchronous?

1. No.
2. Yes.
3. It depends on the target platform.

Have a go hero – improving the push notifications example

Create a sample app that is able to send SMSes using the SMS-plugin you can find at <https://github.com/phonegap/phonegap-plugins> for Android and iOS. When installing the plugin, check whether there are compatibility issues with your current version of PhoneGap and eventually fix them.

Summary

In this chapter, you learned how to extend the PhoneGap capabilities using native code; you are now ready to work on a real project! PhoneGap is a really vibrant project and it's updated several times per year. I will do my best to update the example repository at <https://github.com/GiorgioNatali/PhoneGapGettingStarted> and the Itinero application at <https://github.com/GiorgioNatali/itinero>. Feel free to open issues online to keep the learning process going.

A

Localizing Your App

In computing, internationalization and localization are means of adapting computer software to different languages, regional differences, and the technical requirements of a target market. The term **localization** refers to all the activities needed before your app can be deployed in different languages and according to local cultural conventions. Before starting to localize an app, you have to **internationalize** your code by removing any language and cultural dependencies and designing your code in order to be adapted to various languages without engineering changes. You can then localize your app, translate client-facing content and labels, and otherwise adapt it so that it works well in a particular locale. The term **locale** refers to a collection of settings or preferences to be used in localization. A locale is often described as a language and country pair such as en-US, de-AT, it-IT, and so on. The term **globalization** stands for the combination of internationalization and localization.

There are some odd-looking abbreviations in which a number is used to indicate the number of letters between the first and last letter used to refer to internationalization, localization, and globalization:

- ◆ i18n stands for internationalization
- ◆ l10n stands for localization
- ◆ g11n stands for globalization

From a development point of view, the common practice is to place the text in resource strings that are loaded at the execution time depending on the user settings. There are several techniques you can use to globalize your app such as storing the translations in **PO (portable object)** files, creating a JSON object containing all of them, or loading the localization files dynamically when the app starts. The goal is to deploy an app able to select the relevant language resource file at runtime and to handle culture-aware number and date parsing and formatting, plurals, currencies, special characters, validation, and so on.



Software internationalization is a very huge topic because it covers plurals, dates, special characters, and so on. Discussing all of them is beyond the scope of this book. If you are interested in learning more about internationalization, take a look at the GNU project gettext (<http://www.gnu.org/software/gettext/manual/>), the globalize project (<https://github.com/jquery/globalize>), or the jed project (<http://slexaxton.github.io/Jed/>). For an overview of the most common challenges to address during the internationalization process, watch this great talk at http://www.youtube.com/watch?v=uXS_-JRsB8M.

PhoneGap offers great support for localization through the Globalization API accessible through the `globalization` object. The `globalization` object is a child of the `navigator` object and therefore has global scope. Therefore, in order to access the `globalization` object, it's enough to type the following code snippet:

```
var globalization = navigator.globalization;
```

The `globalization` object exposes several asynchronous methods that have a similar signature. In fact, usually most methods accept an argument, a success and a failure handler, and optionally an `options` object:

```
globalization.methodName(argument, onSuccess, onError, options);
```

Not all the methods accept an argument and the `option` object; some of them accept only a success and a failure handler. The failure handler receives a `GlobalizationError` object as an argument. There are two properties defined on this object message and code. The first one contains a string describing the error details; the second one contains an integer equal to one of the following pseudo constants defined in the `GlobalizationError` object:

- ◆ `GlobalizationError.UNKNOWN_ERROR` (return value 0), a generic error occurred
- ◆ `GlobalizationError.FORMATTING_ERROR` (return value 1), an error occurred during a formatting operation
- ◆ `GlobalizationError.PARSING_ERROR` (return value 2), an error occurred during a parsing operation
- ◆ `GlobalizationError.PATTERN_ERROR` (return value 3), an error occurred recovering a currency, date or number pattern

The Globalization API exposes the following methods defined in the `navigator.globalization` object:

- ◆ `getPreferredLanguage`, returns the string identifier for the device's current language; the string is stored in the `value` property of the object received as an argument in the success handler (i.e. `{value: 'English'}`).

- ◆ `getLocaleName`, returns the locale identifier according to the device's current language; the string is stored in the `value` property of the object received as an argument in the success handler (i.e., `{value: 'en'}`).
- ◆ `dateToString`, returns a date formatted as a string according to the client's locale and time zone; the method accepts a `Date` object as the first argument and an optional `options` object as the last argument:

```
var globalization = navigator.globalization;

var today = new Date();
globalization.dateToString(today, onSuccess, onError);
```

The returned result is stored in the `value` property of the object received as an argument in the success handler (i.e., `{value: '06/14/2013 12:49 PM'}`).

- ◆ `stringToDate`, parses a date formatted as a string, and depending on the device's preferences and calendar, returns the corresponding `Date` object as an argument in the success handler.
- ◆ `getDatePattern`, returns an object received as an argument in the success handler containing:
 - A pattern string to format and parse dates according to the device's preferences
 - The time zone of the device
 - The difference in seconds between the device time zone and the universal time and the offset in seconds between the device's non-daylight saving's time zone
 - The client's daylight saving's time zone (i.e. `{pattern: 'dd/MM/yyyy HH:mm', timezone: 'CEST', utc_offset: 3600, dst_offset: 3600}`)

The method accepts an optional `options` object through which it's possible to specify the format length (i.e., `short`, `medium`, `long`, or `full`) and the data to be returned (i.e., `date`, `time`, or `date and time`).

- ◆ `getDateNames`, returns an array of names of the months or days of the week depending on the device's settings; the array is stored in the `value` property of the object received as an argument in the success handler (i.e., `{value: Array[12]}`).
- ◆ `isDayLightSavingsTime`, returns a Boolean stating whether daylight saving time is in effect for a given `Date` object passed as the first argument using the device's time zone and calendar; the value is stored in the `dst` property of the object received as an argument in the success handler (i.e., `{dst: true}`).

- ◆ `getFirstDayOfWeek`, returns as a number the first day of the week depending on the device's user preferences and calendar, assuming that the days of the week are numbered starting from 1 (= Sunday). The string is stored in the `value` property of the object received as an argument in the success handler (i.e., `{value: 1}`).
- ◆ `numberToString`, returns the number passed as the first argument formatted as a string according to the client's locale and preferences; the number is stored in the `value` property of the object received as an argument in the success handler (i.e., `{value: '12,456,246'}`).
- ◆ `stringToNumber`, returns the string passed as the first argument formatted as a number according to the client's locale and preferences; the number is stored in the `value` property of the object received as an argument in the success handler (i.e., `{value: 1250.04}`).
- ◆ `getNumberPattern`, returns an object received as an argument in the success handler containing:
 - A pattern string to format and parse numbers according to the device's preferences
 - The number of fractional digits to use when parsing and formatting numbers, the rounding increment to use when parsing and formatting, and so on (i.e., `{decimal: '.', fraction: 0, grouping: ',', negative: '-', pattern: '#,##0.###', positive: '', rounding: 0, symbol: '.'}`).
- ◆ `getCurrencyPattern`, returns an object received as an argument in the success handler containing a pattern string to format and parse currencies according to the currency code passed as the first argument and the device's preferences; the number of fractional digits to use when parsing and formatting numbers, the rounding increment to use when parsing and formatting, the ISO 4217 currency code for the pattern, and so on (i.e., `{code: 'EUR', decimal: '.', fraction: 2, grouping: ',', pattern: '$#,##0.00;(¤#,##0.00)', rounding: 0}`).



Both the `numberToString` and `stringToNumber` methods accept an optional `options` object; through the `type` property of this object, you can specify the format of the number (i.e., `decimal`, `percent`, or `currency`).

Through the combination of the data provided by the methods of the globalization object, it's possible to handle very complex scenarios and provide a highly localized app to the end user. In the following example you will learn how to load different strings depending on the device settings using Require.js and its i18n plugin (both of them are available for download at <http://requirejs.org/docs/download.html>).

Time for action – rendering localized messages

Refer to the following steps to render different messages in your app according to the device's language settings:

1. Open the command-line tool and create a new PhoneGap project called globalization.

```
$ cordova create ~/the/path/to/your/source/globalization com.gnstudio.pg.globalization Globalization
```

2. Add the Globalization API plugin using the command line.

```
$ cordova plugins add https://git-wip-us.apache.org/repos/asf/cordova-plugin-globalization.git
```

3. Using the command-line tool, add the platform you want to use for this test (Android, Blackberry, iOS, or Windows Phone 8).

```
$ cordova platforms add android
```

4. Go to the www/js folder, create the libs folder and the subfolders needed to store the require.js file and the i18n plugin.

```
| -libs  
| ---require  
| ----plugins
```

5. Download and save the minified version of Require.js available at <http://requirejs.org/docs/download.html#requirejs> in the js/libs/require folder and the i18n plugin available at <http://requirejs.org/docs/download.html#i18n> in the js/libs/require/plugins folder.

6. Go to www/js and create a new folder named nls (the plugin will search for a folder named in this way to load language files). In this folder create a subfolder to store the data for each supported locale and a root folder to be used in case the user locale is not yet implemented (i.e., root).

```
| -nls  
|   |---en-us  
|   |---es-es  
|   |---fr-fr  
|   |---it-it  
|   |---root
```

- 7.** Create a file named `connection.js` in the `nls/root` folder containing all the default messages to render in case the app is having connection issues. The syntax required by the `i18n` plugin is very similar to the one used to define a new `Require.js` module except that the name of the module is not required.

```
// root
;define({  
  
    'cannotConnect': 'The app can\'t connect to internet.',  
    'connectionSlow': 'The connection is slow be patient.',  
    'connectionError':'Please try again later.'  
  
});
```

- 8.** Create a language file for each supported locale using the same name and the same syntax but with different content so that you can easily get the differences when testing your app. Save the file in the language-specific folder (for example, `fr-fr`).
- 9.** Go to the `nls` folder and create another JavaScript file named `connection.js` in order to specify the supported locales.

```
;define({  
  
    'root': true,  
    'fr-fr': true,  
    'en_us': true,  
    'es-es': true,  
    'it-it': true  
  
});
```

- 10.** Go to the `js` folder, create a new JavaScript file named `main.js`, and define a new `Require.js` module specifying a dependency to the `connection.js` language files.

```
;define('main', ['i18n!nls/connection'],
(function(connection){  
  
    // All the methods and variables will go here  
  
}));
```

- 11.** Create in the `main` module a function named `init` and define in its body a listener for the `deviceready` event.

```
var init = function(){  
  
    document.addEventListener('deviceready', onDeviceReady);  
  
};
```

- 12.** Return the `init` function at the end of the module in order to be able to call it from other modules.

```
return{  
  
    init: init  
  
}
```

- 13.** Once the `deviceready` event is triggered and the PhoneGap framework is loaded, it's possible to access the Globalization API and discover the device current locale using the `getLocaleName` method.

```
var onDeviceReady = function(evt){  
  
    var g = navigator.globalization;  
    g.getLocaleName(onLocaleName, onGlobalizationError);  
  
};
```

- 14.** Define the handler `onLocaleName` and use the values returned by the `i18n` plugin to display all of them on the screen.

```
var onLocaleName = function(locale){  
  
    console.log('onLocaleName', locale.value);  
  
    var deviceready = document.querySelector('#deviceready');  
    var element = document.createElement('span');  
  
    element.innerHTML += 'The cannotConnect message is:<i>' +  
        connection.cannotConnect + '</i><br>';  
    element.innerHTML += 'The connectionSlow message is:<i>' +  
        connection.connectionSlow + '</i><br>';  
    element.innerHTML += 'The connectionError message is:<i>' +  
        connection.connectionError + '</i><br>';  
  
    deviceready.appendChild(element);  
  
};
```

- 15.** Define the `onGlobalizationError` handler and notify the user that an issue has occurred.

```
var onGlobalizationError = function(error) {  
  
    var message = 'code: ' + error.code + '\n';  
    message += 'message: ' + error.message;  
  
    navigator.notification.alert(message, null);  
  
};
```

- 16.** In the `js` folder create a new JavaScript file named `app.js` and use it to configure the paths for `Require.js` to load the `main` module and call the `init` method.

```
require.config({  
  
    paths: {  
  
        i18n: 'libs/require/plugins/i18n'  
    },  
});  
  
require(['main'], function(main){  
  
    main.init();  
  
});
```

- 17.** Open the `index.html` file and add a script tag in the head section to load `Require.js` and the `app.js` file.

```
<script data-main="js/app" src="js/libs/require/require.js"></script>
```

- 18.** Open the command-line tool, go to the project folder, and build and run the app on a real device or an emulator.

```
$ cordova build  
$ cordova run
```

What just happened?

You developed an app i.e. able to render different text messages based on the user's device language settings.

Summary

In this Appendix, you learned how to create a localized app using PhoneGap. The Globalization API is a very powerful tool that allows you to work in conjunction with other JavaScript libraries.

B

Publishing Your App

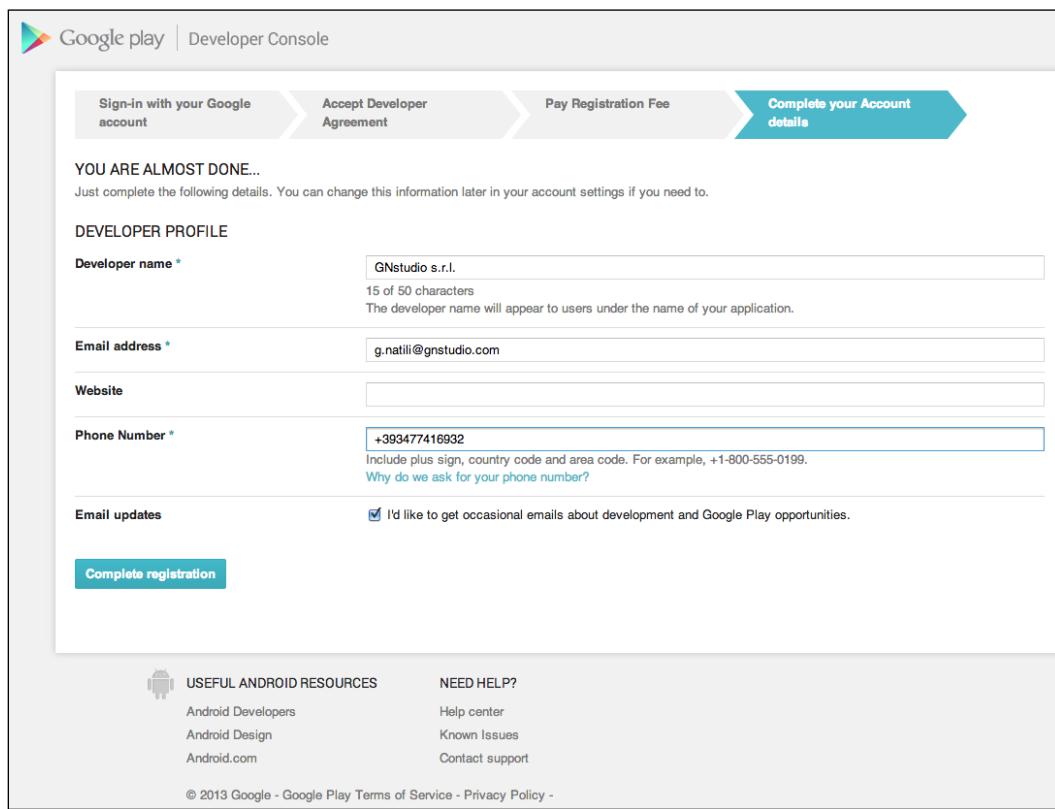
One of the best features of PhoneGap is that it allows you to create a cross-platform app using the same code base. It means that you can reuse most of the code but you still have to build the app for each target platform. You can do this by configuring the development environment for each platform you want to support or you can use online services such as PhoneGap Build Services <http://build.phonegap.com> or Icenium <http://www.icenium.com/>. Both are cloud-based services; the main difference is that PhoneGap Build Services supports all the platforms whereas Icenium supports only the Android and iOS platforms but does come with a very nice online editor. Once the builds are ready, you have to follow a different workflow for each target platform.



Also, if you are dealing with cross-platform development when using PhoneGap, it's always a good habit to use a testing device for each platform during the development phase and when preparing the build.

Publishing on Google Play

Google Play, formerly known as the **Android Market**, is the digital application distribution platform for Android apps. In order to publish an app on Google Play, you have to log in with your Google account and follow the steps outlined at <https://play.google.com/apps/publish>. When registering, remember to have a credit card available because you will be required to pay a fee using Google Wallet before adding the developer details (that is, name, telephone number, e-mail, and so on).



Once you complete the registration process, you can add your apps to the developer console. For each app, you can define the countries in which you want to distribute it, the carriers you want to target, specify whether it's a free app (if you want to sell an app you have to provide a valid Google Wallet merchant account), set up an alpha and beta group for testing and the staged rollouts, and so on. In order to get new users to download and install the app, it's very important to provide detailed information, icons, screenshots, and so on. The Android online guide available here <https://support.google.com/googleplay/androiddeveloper/answer/1078870> is the place to start in order to get a better understanding of the required graphic assets.

In addition, your app has to be less than 50 MB in size and signed using the keystore tool as described at <http://developer.android.com/tools/publishing/app-signing.html>. For apps that require more than 50 MB see the APK expansion details at <http://developer.android.com/google/play/expansion-files.html>.



Once uploaded, your app will be available on the Google Play market within 60 minutes or less.



Publishing on the BlackBerry World

The **BlackBerry World** (previously **BlackBerry App World**) is an application distribution service and application by BlackBerry that allows users to browse, download, and update third-party applications. In order to publish an app on the BlackBerry World market, you need to have a BlackBerry developer account (you can create this for free at <https://developer.blackberry.com/>). You also need to apply to become a vendor providing your BlackBerry ID information (a PayPal account is required to complete the application), which you can do at <https://appworld.blackberry.com/isvportal/home.do>.

The screenshot shows the BlackBerry Vendor Registration interface. At the top, there's a navigation bar with links for North America, Worldwide, Partners, Developers, Why BlackBerry, and a search bar. Below the navigation is a main menu with categories: SMARTPHONES, APPS & SOFTWARE, SUPPORT & SERVICES, SOLUTIONS, and WHERE TO BUY. On the left, a sidebar titled 'BlackBerry World™' includes 'Vendor Portal', 'Logout', and 'Help'. The main content area is titled 'Vendor Registration' and indicates 'Step 1' is active. It says 'Create an administrative user for your Vendor account' and 'Enter your company contact information'. A note states: 'This is the name that will be displayed as the vendor name for your products. Please use the full legal name of your company; or, if you are not part of a company, your own full legal name.' There's a field for 'Vendor name' containing 'GNstudio s.r.l.', a 'Vendor Logo' section with a logo for 'gn studio' and a link to 'www.gnstudio.com', and dropdown menus for 'Legal Status' (Corporation), 'Street address' (Piazza del Popolo 18), 'City' (Rome), 'Country' (Italy), 'State/Province' (Please Select a State), 'ZIP / Postal Code' (00187), and 'Phone' (+390636712855). A note below the logo says: 'Vendor Logo should be 480x480 for square images. For non-square, height is to be 480px and width is to be 480px min and 1440px wide max.'

After you submit your application as a vendor, you will receive a confirmation e-mail asking you to provide official documentation to validate your company information or a copy (front and back) of an official government-issued identification card in case you applied as an individual to the vendor portal.



The verification process can take up to two days, so you have to consider it carefully when you are planning a release for a specific date.

When your account is confirmed, you can add an app (that is, a product) providing a name, a description, the logo, the screenshots, and any other required details about your application. For details on the requirements, refer to the online information available at the BlackBerry World app store https://developer.blackberry.com/devzone/blackberryworld/preparing_your_app_for_blackberry_world.html.

Add Product

Please complete the wizard to submit your product.
* indicates required fields

Step 1 Step 2 Step 3 Step 4 Step 5

Product Details

Describe how your product will appear in BlackBerry World™ and your licensing model.

By default your product will be installed using your Vendor Company Name and Product Name, if your JAD file uses a different product and company name you can override them here.

NOTE: Only for advanced users, use this ONLY if your existing download JAD file is different than your BlackBerry World account.

* Product Name: ?
Override Names: Override JAD file product and vendor names ?

The release of an app to the BlackBerry World also involves a signing process. This process assumes that you already have downloaded and installed the BlackBerry 10 WebWorks SDK available at <https://developer.blackberry.com/html5/downloads/>. Before you can get your BlackBerry 10 app signed, you have to complete the web form at <https://www.blackberry.com/SignedKeys>. When your application is accepted, you will receive two .csj registration files by e-mail. Each file arrives in a separate e-mail message with information about the purpose of the file attached (one is to generate a debug token and one to sign the app for the marketplace). In order to register with the RIM Signing Authority, you have to run the .bar file stored in the \dependencies\tools\bin folder located in the BlackBerry 10 WebWorks SDK installation folder from your command-line tool. This tool creates the following files needed to digitally sign the app: author.p12, barsigner.csk, and barsigner.db.

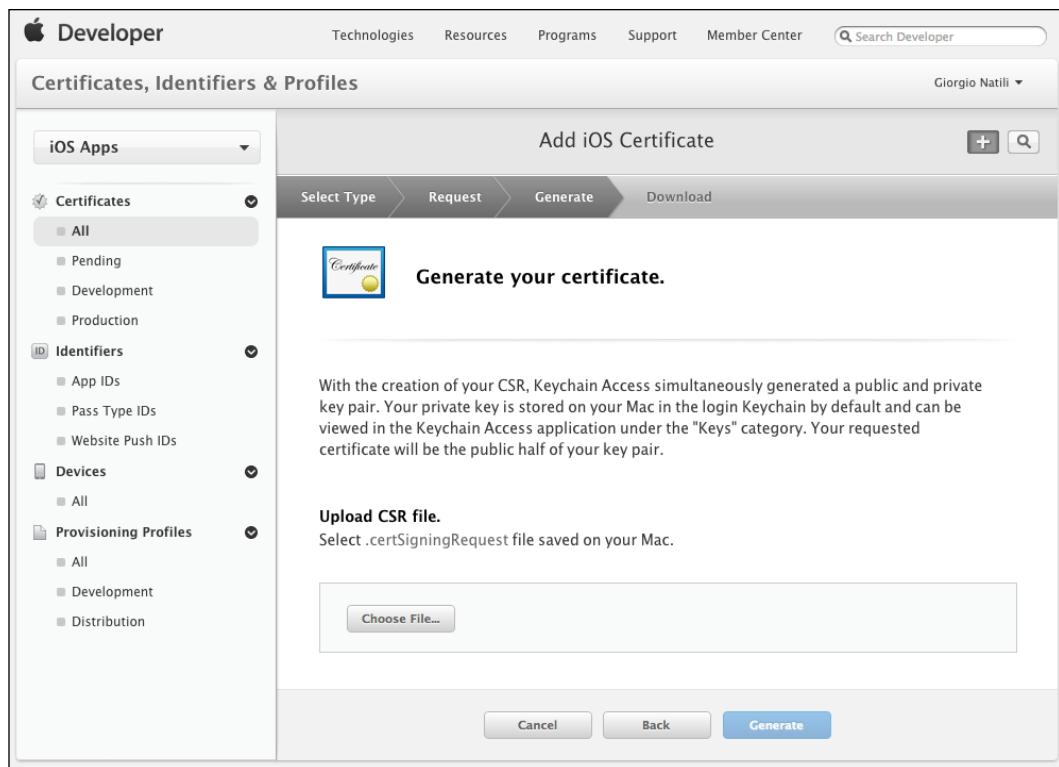
Be very careful about the app naming; an application name that starts with a brand/company/product implies an association and that it's an authorized or an official application. If you name your app YouTube Player it will be rejected for sure (instead name it Player for YouTube).

Publishing on the Apple App Store

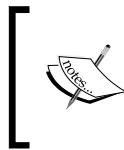
The **Apple App Store** is a digital application distribution platform for iOS apps maintained by Apple Inc. Users can browse through the App Store and install applications directly to an iOS device. Although Apple envisions the App Store to be a global product, in reality its market is restricted by national boundaries. In other words, there are potentially as many distinct App Stores as there are countries in the world. To publish an app through the Apple App Store, you need to have an Apple Developer account (<http://developer.apple.com/programs/register>) and be a member of the iOS Developer Program (<http://developer.apple.com/programs/ios/>; the cost is USD 99 per year). The first step is to register an app ID with the developer portal and then you have to create the development and distribution certificates.

The screenshot shows the Apple Developer portal interface. The top navigation bar includes links for Technologies, Resources, Programs, Support, Member Center, and a search bar. The user is identified as Giorgio Natili. The main content area is titled "Certificates, Identifiers & Profiles" and has a dropdown menu set to "iOS Apps". On the left, there are three main categories: Certificates, Identifiers, and Devices, each with sub-options like All, Pending, Development, Production, etc. The "Identifiers" section is expanded, and "App IDs" is selected. The right side of the screen displays the "Register iOS App ID" form. The title "ID Registering an App ID" is shown, along with a note explaining the App ID string format (Team ID prefix followed by a Bundle ID suffix). The "App ID Description" section contains a "Name:" field where the user has typed "I". Below it, a note states: "You cannot use special characters such as @, &, *, ',". The "App ID Prefix" section shows the value "D34NY92TC4 (Team ID)".

Then you need to set up a distribution certificate. To do this, you will first need to generate a certificate request from your computer and then upload it to the developer portal. On a Mac, you should do this by opening the **Keychain Access** application available in **Utilities** and then go to **Keychain Access | Certificate Assistant | Request a Certificate from a Certificate Authority**. Enter your e-mail address and name and select **Request is Saved to disk** to save the file `CertificateSigningRequest.certSigningRequest` on your desktop. Go to the developer portal, upload the certificate request, and complete the steps required to generate the distribution certificate.



When the certificate is ready, you can create a new distribution Provisioning Profile by selecting the app you want to submit and the certificate to use. Download the file and, in Xcode select **Window | Organizer**, click on **Devices**, select **Provisioning Profiles** and drag the provisioning profile with the `.mobileprovision` extension to the **Organizer**. Next, open the build settings pane and set the code-signing identity; in this way, the app is then code signed when you create an archive and you can complete the publication procedure using Xcode. When submitting the app, you will also be required to provide a description, several screenshots, icons, and other information. For details, refer to the online documentation at <https://developer.apple.com/library/ios/#documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html>.



The verification process varies depending on the number of submissions currently under review, but it typically takes more than two days. You can find the App Store estimated review time at <http://reviewtimes.shinydevelopment.com/>.

Presenting a simple version of your app as first release will help to speed up the approval process a little bit. It's the initial app approval process that takes the most time; once approved, future updates are far easier to get done. So keep the advanced features for later releases of your app.

Publishing on the Windows Phone Store

The **Windows Phone Store** (previously **Windows Phone Marketplace**) is a digital distribution platform that allows users to browse and install applications that have been developed by third parties. The UI is presented in a very "Metro UI" way using a panoramic view where the user can browse categories and titles, see featured items, and get details with ratings, reviews, screenshots, and pricing information.

To submit and manage apps on Windows Phone Dev Center, you first have to register and become a member using a Microsoft account (formerly known as Windows Live ID). When registering, you will be asked to pay an annual Developer Center subscription fee of USD 99 plus any applicable tax. In exchange, you'll get to submit unlimited paid apps to Windows Phone Store (you can also submit up to 100 free apps). The publishing process is simple and straightforward: you have to provide the app details (name, description, screenshots, and so on) and then submit the XAP file packaged with Visual Studio. You must package and prepare your app before you can upload it to the store; the packaging process starts when you create a Windows Store project or item based on a template (refer to the online documentation at <http://msdn.microsoft.com/en-us/library/windows/apps/br230260.aspx> for a complete overview of the packaging process).



The verification process is pretty fast but you need a couple of days to complete the registration process if you are registering as a company.

In order to reduce the duration of the review, you can screen your app locally using the **Windows Application Certification Kit (WACK)** tool available in the Windows Phone SDK. It reduces the approval cycle by giving you a way to screen your app locally for issues before you even submit it to the Windows Store.

Summary

In this Appendix, you learned how you can publish your app on different app stores and about common issues. It's pretty clear that the Apple one is the one that requires the most complex workflow but it also attracts most developers. You can easily manage publication on all the different markets by yourself or with the support offered by services such as the one by the PhoneGap build.

Pop Quiz Answers

Chapter 2, Building and Debugging on Multiple Platforms

Pop quiz – getting started with mobile apps

Q1	1
Q2	3

Chapter 3, Getting Started with Mobile Applications

Pop quiz – getting started with mobile apps

Q1	3
----	---

Chapter 4, Architecting Your Mobile App

Pop quiz – getting started with mobile apps

Q1	3
Q2	3

Chapter 5, Improving the User Interface and Device Interaction

Pop quiz – getting started with mobile apps

Q1	3
Q2	1

Chapter 6, Using Device Storage and the Contacts API

Pop quiz – getting started with mobile apps

Q1	3
Q2	3

Chapter 7, Accessing Device Sensors

Pop quiz – getting started with mobile apps

Q1	2
Q2	2

Chapter 8, Using Location Data with PhoneGap

Pop quiz – getting started with mobile apps

Q1	2
Q2	2

Chapter 9, Manipulating Files

Pop quiz – working with files

Q1	2
Q2	2

Chapter 10, Capturing and Manipulating Device Media

Pop quiz – getting started with media

Q1	3
Q2	2

Chapter 11, Working with PhoneGap Plugins

Pop quiz – getting started with PhoneGap plugins

Q1	2
Q2	2

Index

A

abort method 213
accelerometer 152
Accelerometer API
 about 152, 156
 device orientation events 165, 166
 device orientation, handling with JavaScript
 167, 168
 shakes, detecting 157-165
 using 156, 157
addEventListener function 59
Adobe Brackets
 about 32
 cloud service, configuring 33, 34
 features 33
Adobe Edge Inspect
 about 51
 features 51
 integrating, with weinre 51, 52
 used, for wireless debugging 51
ADT
 about 17, 35
 installing, into Eclipse 18-22
AJAX request 110
Alice.js
 about 123
 configuration properties 123
 URL 123
anatomy, plugin 241-244
Android
 URL 12

Android development environment
 setting up 13, 14
Android Development Tools. *See* **ADT**
Android Market. *See* **Google Play**
Android online guide
 URL 264
Android platform
 sensors 154
Android Sensor Box 154
Apache Cordova 7, 8
Apache Cordova 2.x
 features 9
Apache Cordova community 71
Apache Cordova Plugman project
 about 240
 plugins, using with 240
 URL 240
Apple App Store
 about 267
 PhoneGap app, publishing on 267, 268
Apple Developer Center
 URL 22
Apple Developer website
 URL 77
ApplicationCache interface
 URL 130
application data storage
 about 129, 130
 ApplicationCache interface 130
 IndexedDB API 130
 LocalStorage API 130
 PhoneGap LocalStorage API, exploring 130

SessionStorage API 130
SQL storage, exploring 136, 137

app views

about 117-119
navigation 120-122
templates, creating 119

Aptana Studio

download link 35
architecture, itinero sample app 89, 90
async RequireJS plugin
downloading 187
Audits panel, Chrome Developer Tools 41
augmented reality 155
autocorrect features 57

B

barometer 152
Battery API 125
BlackBerry 10
URL 12
BlackBerry 10 WebWorks SDK
downloading 266
Blackberry World
about 265
PhoneGap app, publishing on 265, 266
BlackBerry World app store
URL 266

C

Camera API
about 125, 220
used, for accessing camera 220-226
used, for controlling camera popover 227, 228
used, for controlling camera roll 229, 230
camera popover
controlling, Camera API used 227, 228
camera roll
positioning, Camera API used 229
Capture API
about 125, 220, 230, 231
accessing 115, 116
images, manipulating with canvas 234, 235
CaptureError object
pseudo constants 232
Car Finder 155
chunkedMode property 212

cleanPicture function 226
cleanup method 220
clearWatch() method 157, 179
click-to-call format detection 57
cloud service
configuring, in Adobe Brackets 33, 34
CodeIntel plugin 28
code refactoring 31
command-line tool 11
common code base
creating, for multiplatform apps 24
Compass API
about 152, 170, 171
compass, creating 172
using 173, 174
compass.clearWatch function 170
compass.getCurrentHeading function 170
CompassHeading object
properties 170
compass.watchHeading function 170
compression tools
comparing 108
ConfigurationData object
about 231
properties 231
Connection API
exploring 126, 127
Console panel, Chrome Developer Tools 41
ContactAddress object
about 144
properties 144
contact data, Contact object
filtering 147
ContactError object 145
ContactField object
about 143
properties 143
ContactName object
about 143
properties 143
Contact object
about 145
properties 146
ContactOrganization object
properties 144
Contacts API
about 125, 142

ContactAddress object 144
contact data, filtering 147
ContactField object 143
ContactName object 143
Contact object 145, 146
ContactOrganization object 144
device contacts, filtering 147, 148

contextual help screen
rendering 158

controller 121

Coordinates object
about 185
properties 185

cordova-cli tool
about 24
installing, npm used 71, 72
used, for setting up project 71

cordova.exec function 241

create method 146

CSSLoader 163

CSS media queries 57

CSS mobile properties 57, 58

CSS mobile-specific properties 58

CSS Prefixer plugin 27

CSS Sprites 62

custom plugins 238

D

database limitations 141

database storage
performing, with PhoneGap 137, 138

Data URI 61

dateToString method 255

dependencies
installing 12

desktop browsers
Gecko debug 43
Internet Explorer 10 46
WebKit debug 37
working with 36

Developer Tools, Chrome
Audits panel 41
Console panel 41
Elements panel 38
Network panel 39

Profiles tool 40
Resources panel 39
Sources panel 39
Timeline panel 39

development environment
Android, setting up 13, 14
folder access, speeding up with jump module 82
iOS, setting up 15, 16
LiveReload, enabling 86
LiveReload, used for refreshing pages 86
server alias, creating with serve 83, 84
setting up 13
shell, customizing with iTerm2 84
tuning 81
Windows Phone, setting up 16

development tools
about 26
Adobe Brackets 32
Eclipse 35
IntelliJ IDEA 30
Sublime Text 26

Device API 124

device camera
accessing, Camera API used 223-226

device contacts, Contact object
filtering 147

Device Motion API 125

devicemotion event 60

device orientation
handling, with JavaScript 167, 168

Device Orientation API 125

deviceorientation event 60

DeviceOrientationEvent event 166

device orientation events 165, 166

device position
displaying, Google Maps used 181-183

deviceready event 115

deviceready function 168, 224, 248

device sensors
about 151
accelerometer 152
barometer 152
categories 152
compass 152
gyroscope 152

orientation 152
device sensors, categories
environmental sensors 152
motion sensors 152
position sensors 152
deviceshake event 162
Dialogs API 125
directories
reading 200
DirectoryReader object 201
dotfiles
URL 81
download method 213

E

Eclipse
about 35
ADT, installing 18
URL 17
Edge Inspect. *See* **Adobe Edge Inspect**
Elements panel, Chrome Developer Tools 38
Emmet plugin 28
environmental sensors 152
errorHandler function 249

F

failure handler function 180
File API 125
file compression 102
file data
reading 206-208
writing 206-208
fileKey property 212
fileName property 212
files
downloading 214
reading 200
saving 215, 216
transferring 212, 213
Files API
about 197-199
directories, reading 200, 201
file data, reading 206-208
file data, writing 206-208
file, downloading 214, 216
file, saving 214, 215

files, reading 200, 201
files, transferring 212, 213
folders and files, listing 201-205
image, reading 209

File Transfer API 125
filterContacts function 148
FindPlaceView.js module 190

Firebug
about 43
filter logs, filtering 45
installing 43
Script console 44
toolbar 44

Firebug extension
installing 44

Firefox
about 43
Firebug extension, installing 43

Firefox OS
URL 12
folders and files
listing 201-205
Foundation framework
about 68
downloading 68
features 69

G

g11n. *See* **globalization**
geocode method 192
geolocation 177
Geolocation API
about 125, 179
device position, displaying with Google Maps
181-183
Geolocation data
about 185
places, discovering with Google Places 187-193
geolocation object
about 179
methods 179
geo targeting 177
getCurrencyPattern method 256
getCurrentAcceleration method 156
getCurrentPosition() method 179
getDateNames method 255

getDatePattern method 255
getFile method 211
getFirstDayOfWeek method 256
getLocaleName method 255
getNumberPattern method 256
getPicture method 220, 227
getPlaces event listener 191
getPreferredLanguage method 254
GitHub repository
 URL, for mediator features 91
globalization 253
Globalization API
 about 126, 254
 methods, navigator.globalization object 254
GlobalizationError object
 pseudo constants, defining 254
globalization object 254
globalize project
 URL 254
Global Positioning System (GPS) 178
Gnome 8
GNU project gettext
 URL 254
Google Analytics JavaScript library 62
Google API usage 195
Google Closure Compiler
 about 102
 used, for compressing files 104, 105
 using 102, 103
Google Maps
 used, for displaying device position 181, 183
Google Packaged Apps 43
Google Places
 used, for discovering places 187-194
Google Play
 about 264
 PhoneGap app, publishing on 264
GPU 63
gyroscope 152, 153

H

Handlebars.js 109
hardware-accelerated transitions 123
Hello World application
 creating 72, 73

cross-platform app, creating 73
Hogan.js 109
HTML5 Mobile Boilerplate
 about 68
 downloading 68
hybrid app 35

I

i18n. See **internationalization**
ICanHaz.js 109
Iceniun
 URL 263
IE 10 Developer Tools 48
image
 reading 209-211
 rendering 209-211
InAppBorwser API 126
Inclinometer sensor 154
IndexedDB API
 URL 130
initAdditionalOptions function 229
init function 189
installation
 PhoneGap 11
integrated development environment (IDE) 26
IntelliJ IDEA
 about 30, 31
 advantages 32
 blog, URL 32
 download page 32
 features 30
internationalization 253
Internet Explorer 10 46
iOS
 URL 12
iOS 6 remote debug 52
iOS debugging
 iWebInspector, configuring for 49
iOS development environment
 setting up 15
iOS platform
 sensors 154
iOS SDK 10
 installing 22, 23
ios-sim tool 13
isDayLightSavingsTime method 255

iTerm2
downloading 84
installing 84
URL 84
used, for customizing shell 84, 85

itinero reference app 185, 186

itinero sample app
about 87
architecture 89, 90
bootstrap loader 95
core, building 93, 94
exploring 87
features 87, 88
mediators 91
modules, communicating between 91
Mustache template, creating 96, 97
navigation flow 88
Require.js Bootstrap, configuring 95, 96
software module 91
splash screen, setting up 98, 99
template, loading 98
template, parsing 98

iWebInspector
about 49
configuring, for iOS debugging 49
URL 49

J

JavaScript best practices, mobile web
about 65
event handling 67, 68
loose coupling 66

JavaScript compression
exploring 102
Google Closure 102
optimizing, with Require.js 107
UglifyJS2 105

JavaScript, for mobile 101 58

JavaScript guidelines
reviewing 86

JavaScript WebWorker 132

jed project
URL 254

jQuery Mobile
about 70
downloading 70

features 70
URL 70

jump module
downloading 82
used, for speeding up folder access 82

L

l10n. *See* **localization**

layers, PhoneGap app
business logic 237
PhoneGap framework 237
user interface 237

LiveReload
about 86
enabling 86
URL 86

load event 115

loadStoredData function 134

local database
populating 138-140

locale 253

LocalFileSystem object 198

localization 253

localized messages
rendering 257-260

LocalStorage API
data, reading 132-135
data, writing 132-135
URL 130

localStorage object
drawbacks 132
methods 131
properties 131

location data 177

loose coupling 66

M

Mac App Store (MAS) 77

magnetometer 153

Media API 125

media capture API 61

MediaFile object
properties 232

mediators, itinero sample app
about 91

add () method 91
broadcast () method 91
get () method 91
has () method 91
remove () method 91
methods, localStorage object
 clear 131
 getItem 131
 key 131
 length 131
 removeItem 131
 .setItem 131
methods, navigator.camera object
 cleanup method 220
 getPicture 220
methods, navigator.globalization object
 dateToString 255
 getCurrencyPattern 256
 getDateNames 255
 getDatePattern 255
 getFirstDayOfWeek 256
 getLocaleName 255
 getNumberPattern 256
 getPreferredLanguage 254
 isDayLightSavingsTime 255
 numberToString 256
 stringToDate 255
 stringToNumber 256
MimeType property 212
Mobile App Analytics SDK
 URL 62
mobile counterparts
 mimicking 52, 54
mobile debugging workflow 48
mobile-specific CSS
 media queries 57
 mobile properties 58
mobile-specific HTML tags
 autocorrect features 57
 click-to-call format detection 57
 viewport meta tag 56
mobile-specific JavaScript
 about 58
 addEventListener function 59
 Data URI 61
 devicemotion event 60
 deviceorientation event 60
 media capture API 61
 orientationchange event 60
 querySelector method 59
 querySelectorAll method 59
mobile web application. *See also PhoneGap app*
 JavaScript, writing 65
 performance best practices 62, 64
 pixel density 64
 screen size 64
 UI images, scaling 64
 web app templates, selecting 68
Module Pattern 92
motion sensors 152
multipage pattern
 about 113
 disadvantages 114

N

native apps 35
Native Inspector 45
native iOS-like UI
 creating 77
navigation flow, itinero sample app 88
navigator.camera object
 about 220
 methods 220
 properties 220
navigator.contacts object 147
Nearby Search 193
Network API 125
Network panel, Chrome Developer Tools 39
Node.js
 URL 50
NodeJS project 8
Node package manager (npm) 9
numberToString method 256

O

occasionally connected computing 130
onabort event 207
onContactFindSuccess function 148
onerror event 207
onGetFile function 211
onloadend function 208
onload function 208
onloadstart function 208

onNotificationAPN function 249
onNotificationGCM function 249
onResult function 193
onSuccess function 199
onSuccess handler 199
onTripmateChange function 148
onwrite event 207
onwritestart event 207
openDatabase method 137
operating systems 9
orientation 152
orientationchange event 60
orientationchange event handler 166

P

Parallels 10
params property 212
parseDirectory function 202
parseEntries function 203
parseQueuedDirs function 202, 204
pause event 115
perceptual computing 155
performance best practices, mobile web 62, 63
performance.memory property 42
PhoneGap
 about 7, 8
 Camera API 220
 Capture API 220
 desktop browsers, working with 36, 37
 development tools 26
 evolution 8
 Files API 197
 hybrid app 35
 installing 11
 mobile counterparts, mimicking 52
 mobile debugging workflow 48
 native apps 35
 remote debugging 48
 Web apps 35
PhoneGap Accelerometer API. *See* **Accelerometer API**

PhoneGap APIs
 about 124
 Battery API 125
 Camera API 125
 Capture API 125

Contacts API 125
Device API 124
Device Motion API 125
Device Orientation API 125
Dialogs API 125
File API 125
File Transfer API 125
Geolocation API 125
Globalization API 126
InAppBorwser API 126
Media API 125
Network API 125
Splashscreen API 126
Vibration API 125
PhoneGap app. *See also* **mobile web application**
 cross-platform app, creating 73
 interactivity, adding 74
 layers 237
 modal window, opening
 programmatically 74-76
 multiple views, creating 113
 native-like CSS, setting up 77
 publishing, on Apple App Store 267, 268
 publishing, on Blackberry World 265, 266
 publishing, on Google Play 264
 publishing, on Windows Phone Store 269

PhoneGap Build Services

 URL 263

PhoneGap Compass API. *See* **Compass API**

PhoneGap Geolocation API. *See* **Geolocation API**

PhoneGap lifecycle events

 about 114
 deviceready event 115
 load event 115
 pause event 115
 resume event 115
 unload event 115

PhoneGap LocalStorage API

 exploring 130

PhoneGap platform

 about 10
 Android apps, developing 10
 BlackBerry apps, developing 10
 Symbian Web Runtime apps, developing 10

PhoneGap plugins

 about 237
 anatomy 241-244

push notifications plugin 245, 246
 usig, with Plugman 240, 241
 working with 245

PhoneGap SQL storage

- exploring 136, 137

pistachio

- about 109
- features 110

pixel density, mobile web application **64**

places

- discovering, Google Places used 187-194

plugins. *See PhoneGap plugins*

plugins, Sublime Text

- CodeIntel 28
- CSS Prefixer 27
- Emmet 28
- SideBarEnhancements 27
- SublimeLinter 27

plugman **9**

Plugman project. *See Apache Cordova Plugman project* **124**

PO (portable object) files **253**

PopoverArrowDirection object **227**

position sensors **152**

printDir function **205**

Profiles tool, Chrome Developer Tools **40**

project

- setting up, cordova-cli used 71

properties, CompassHeading object

- headingAccuracy 170
- magneticHeading 170
- timestamp 170
- trueHeading 170

properties, ConfigurationData object

- height 231
- type 231
- width 231

properties, ContactAddress object

- country 144
- formatted 144
- locality 144
- postalCode 144
- pref 144
- region 144
- streetAddress 144
- type 144

properties, ContactField object

- pref 143
- type 143
- value 143

properties, ContactName object

- familyName 143
- formatted 143
- givenName 143
- honorablePrefix 143
- honorableSuffix 143
- middleName 143

properties, Contact object

- addresses 146
- birthday 146
- categories 146
- displayName 146
- emails 146
- id 146
- ims 146
- name 146
- nickname 146
- note 146
- organizations 146
- phoneNumbers 146
- photos 146
- urls 146

properties, ContactOrganization object

- department 145
- name 145
- pref 144
- title 145
- type 144

properties, Coordinates object

- accuracy 185
- altitude 185
- altitudeAccuracy 185
- heading 185
- speed 185

properties, MediaFile object

- fullPath 232
- lastModifiedDate 232
- name 232
- size 232
- type 232

properties, navigator.camera object

- allowEdit 221

cameraDirection 221
 correctOrientation 221
 destinationType 220
 encodingType 221
 mediaType 221
 popoverOptions 221
 quality 220
 saveToPhotoAlbum 221
 sourceType 221
 targetHeight 221
 targetWidth 221
pseudo constants
 FileError.ABORT_ERR 200
 FileError.ENCODING_ERR 200
 FileError.INVALID_MODIFICATION_ERR 200
 FileError.INVALID_STATE_ERR 200
 FileError.NO_MODIFICATION_ALLOWED_ERR 200
 FileError.NOT_FOUND_ERR 199
 FileError.NOT_READABLE_ERR 200
 FileError.PATH_EXISTS_ERR 200
 FileError.QUOTA_EXCEEDED_ERR 200
 FileError.SECURITY_ERR 199
 FileError.SYNTAX_ERR 200
 FileError.TYPE_MISMATCH_ERR 200
push notifications plugin
 about 245
 using 245-250

Q

Quaternion sensor 154
querySelectorAll method 59
querySelector method 59

R

readAsArrayBuffer 209
readAsBinaryString 209
readAsDataURL 208
readAsDataURL method 211
readAsText 208
readEntries method 205
readTransaction() method 138
remote debugging
 about 48
 iOS 6 remote debug 52

iWebInspector, used 49
 weinre, used 50
 wireless debugging, with Wireless debugging 51
render function 190
requestFileSystem method 198-210
Require.js
 about 92, 93
 used, for optimizing JavaScript 107
resolveLocalFileSystemURI method 199
Resources panel, Chrome Developer Tools 39
Responsive Design View tool 46
resume event 115
retina display user interface
 about 111
 elements, preparing 112, 113
 handling 111
Ripple 10
Ripple extension, Chrome
 URL 53

S

saveUserToStorage function 135
screen orientation 60
Semantic Versioning specification 8
sensors
 interacting, with human-computer 155
 using 155, 156
sensors, Android platform
 ambient humidity 154
 ambient pressure 154
 ambient temperature 154
 luminance 154
sensors, iOS platform
 accelerometer 154
 gyroscope 154
 magnetometer 154
 proximity 154
sensors, Windows Phone 7.5/8 platform
 Inclinometer sensor 154
 Quaternion sensor 154
serve module
 installing 83
SessionStorage API
 URL 130
setPosition method 227

shake detection module
integrating, in app 158
shakes
detecting, Accelerometer used 159-165
SideBarEnhancements plugin 27
single page pattern
about 114
disadvantage 114
Skype or Photoshop Lightroom 136
Sources panel, Chrome Developer Tools 39
splash screen
about 98
setting up 98, 99
Splashscreen API 126
SQLcipher
URL 141
SQLite 136
SQLite Plugin
URL 141
using 141
State.js module 120
StorageEvent properties
about 131
key 131
newValue 131
oldValue 131
url 131
stringToDate method 255
stringToNumber method 256
SublimeLinter plugin 27
Sublime Text
about 26
blog, URL 29
download link 29
plugins 27
shortcuts 28
using 27
success handler function 180, 248

T

takePicture function 224
template
compiling, UglifyJS2 used 110, 111
template engine compression
using 109, 110

Timeline panel, Chrome Developer Tools 39

tokenHandler function 249

transaction() method 138

trilateration 179

Twitter Bootstrap

about 70

downloading 70

U

UglifyJS2 project

used, for compiling template 110

UglifyJS project

about 105

using, with Closure Compiler 106

UIPopoverArrowDirection class 227

UIWebView class 136

underscore.js templates 109

unload event 115

updateContent function 121

updatePreferences function 225

UQL

about 137

URL 137

V

Vibration API 125

view 121

viewport meta tag 56

Visual Studio

working with 23

VMWare Fusion 10

W

watchAcceleration method 157

watchHeading function 171

watchPosition() method 179

web apps 35

web app templates

about 68

Foundation 68

HTML5 Mobile Boilerplate 68

jQuery Mobile 70

selecting 71

Twitter Bootstrap 70

Web Inspector Remote. *See* `weinre`

WebKit debug

- about 37
- Chrome 37
- Safari 37

Web SQL

- about 136
- implementation 141

WebStorage API 130

WebView 72, 102

weinre

- about 9, 50
- Adobe Edge Inspect, integrating with 51, 52
- Node.js, configuring for 50

Windows Application Certification Kit (WACK) tool 269

Windows Phone

- URL 12

Windows Phone 7 SDK 10

Windows Phone development environment

- setting up 16

Windows Phone Store

- about 269
- PhoneGap app, publishing on 269
- URL, for online documentation 269

wireless debugging

- Adobe Edge Inspect, used 51

X

Xcode

- installing, from App Store 22

Xcode command-line tools

- installing 22



Thank you for buying PhoneGap 3 Beginner's Guide

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

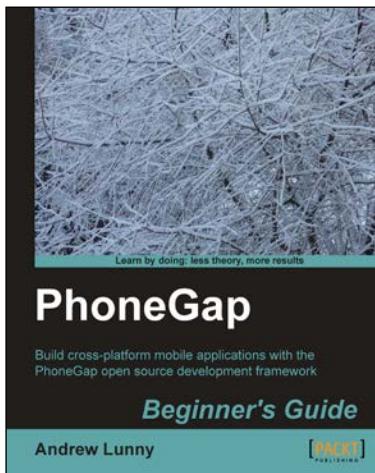
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

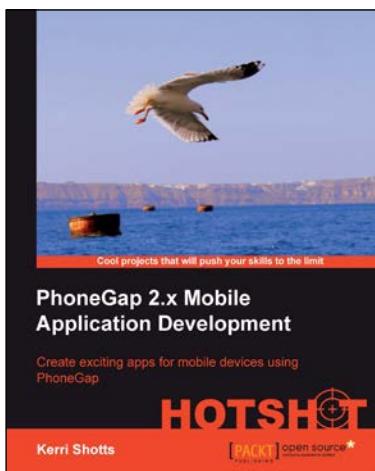


PhoneGap Beginner's Guide

ISBN: 978-1-84951-536-8 Paperback: 328 pages

Build cross-platform mobile applications with the PhoneGap open source development framework

1. Learn how to use the PhoneGap mobile application framework
2. Develop cross-platform code for iOS, Android, BlackBerry, and more
3. Write robust and extensible JavaScript code
4. Master new HTML5 and CSS3 APIs
5. Full of practical tutorials to get you writing code right away



PhoneGap 2.x Mobile Application Development Hotshot

ISBN: 978-1-84951-940-3 Paperback: 388 pages

Create exciting apps for mobile devices using PhoneGap

1. Ten apps included to help you get started on your very own exciting mobile app
2. These apps include working with localization, social networks, geolocation, as well as the camera, audio, video, plugins, and more
3. Apps cover the spectrum from productivity apps, educational apps, all the way to entertainment and games
4. Explore design patterns common in apps designed for mobile devices

Please check www.PacktPub.com for information on our titles



PhoneGap Mobile Application Development Cookbook

ISBN: 978-1-84951-858-1 Paperback: 320 pages

Over 40 recipes to create mobile applications using the PhoneGap API with examples and clear instructions

1. Use the PhoneGap API to create native mobile applications that work on a wide range of mobile devices
2. Discover the native device features and functions you can access and include within your applications
3. Packed with clear and concise examples to show you how to easily build native mobile applications



WordPress Mobile Applications with PhoneGap

ISBN: 978-1-84951-986-1 Paperback: 96 pages

A straightforward, example-based guide to leveraging your web development skills to build mobile applications using WordPress, jQuery, jQuery Mobile, and PhoneGap

1. Discover how we can leverage on Wordpress as a content management system and serve content to mobile apps by exposing its API
2. Learn how to build geolocation mobile applications using Wordpress and PhoneGap
3. Step-by-step instructions on how you can make use of jQuery and jQuery mobile to provide an interface between Wordpress and your PhoneGap app

Please check www.PacktPub.com for information on our titles