

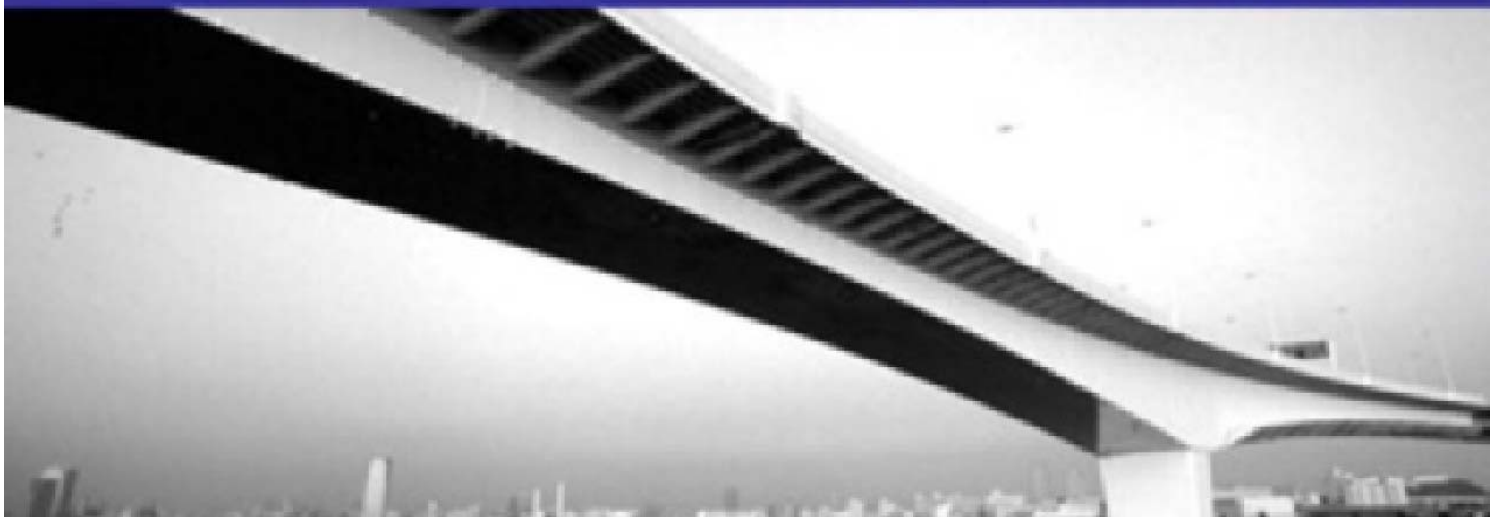
Lee S. Barney



Developing Hybrid Applications for the iPhone

Using HTML, CSS, and JavaScript to
Build Dynamic Apps for the iPhone

Developer's Library



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Barney, Lee.

Developing hybrid applications for the iPhone : using HTML, CSS, and JavaScript to build dynamic apps for the iPhone / Lee S. Barney.

p. cm.

Includes index.

ISBN 978-0-321-60416-3 (pbk. : alk. paper) 1. iPhone (Smartphone)–Programming.
2. Application software–Development. 3. Cross-platform software development. I. Title.

TK6570.M6B37 2009

621.3845'6–dc22

2009019162

Copyright © 2009 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671-3447

ISBN-13: 978-0-321-60416-3

ISBN-10: 0-321-60416-4

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.

First printing June 2009

Editor-in-Chief
Karen Gettman
Acquisitions Editor
Chuck Toporek
Development
Editor
Sheri Cain
Managing Editor
Kristy Hart
Project Editor
Jovana San
Nicolas-Shirley
Copy Editor
Deadline Driven
Publishing
Indexer
Erika Millen
Proofreader
Kathy Ruiz
Technical
Reviewers
August Trometer
Randall Tamura
Publishing
Coordinator
Romny French
Cover Designer
Gary Adair
Compositor
Jake McFarland

Preface

This book shows you how to create a new type of iPhone application: hybrid applications written in HTML, CSS, and JavaScript. Hybrid iPhone applications are standalone applications that run like regular applications on your iPhone, but don't require the files to live on a server on the Internet.

Creating hybrid iPhone applications reduces creation time and the learning curve required to get your application into the hands of your customers, because you don't have to learn Objective-C or have an intimate knowledge of the Cocoa frameworks.

Hybrid Application Development Tools

This book covers the two most commonly used open-source JavaScript software packages for writing applications for the iPhone and iPod touch devices: QuickConnectiPhone and PhoneGap. These packages enable you to build applications that access native device features directly from JavaScript, such as vibration, GPS location information, the accelerometer, and many other things—all without writing a single line of Objective-C or Cocoa.

QuickConnectiPhone, downloaded from <http://sourceforge.net/projects/quickconnect>, exposes the most native device behavior and provides a highly engineered, full-featured framework for development use. QuickConnectiPhone dramatically reduces your application's time-to-market because part of the framework consists of all of the glue code you have to typically write in Objective-C, Cocoa, and JavaScript. Best of all, it does not require a remote server for hosting JavaScript, HTML, and CSS files.

The second package is PhoneGap, downloaded from <http://phonegap.com>. PhoneGap exposes fewer native behaviors and is a library rather than a full-fledged framework. As a library, PhoneGap enables you to engineer your application any way you want. It does, however, require a remote server for hosting files.

To reduce the learning curve and improve your understanding, good, solid examples are used throughout this book.

If you want to create installable iPhone applications, have the web skills required, and want to create dynamic, compelling solutions that people will use, this book shows you how using these two packages.

Table P.1 compares what each package can do at the time of writing this book.

Table P.1 Comparing the Features of QuickConnectiPhone and PhoneGap

Behavior/Data Available	QuickConnectiPhone	PhoneGap
GPS	Yes	Yes
Accelerometer	Yes	Yes
Vibrate	Yes	Yes
System sounds	Yes	Yes
Ad-hoc (Bonjour) networking	Yes	No
Sync cable networking	Yes	No
Browser-based database access	Yes	No
Shipped database access	Yes	No
Drag-and-drop library	Yes	No
AJAX wrapper	Yes	No
Record/Play audio files	Yes	No
Embedded Google maps	Yes	No
Charts and graphs library	Yes	No

How to Use This Book

Each chapter is organized into two parts. The first part shows you how to use the relevant feature of either QuickConnectiPhone or PhoneGap to accomplish a particular task, such as getting the current geolocation of the device. The second part of the chapter shows how the code behind the JavaScript call is written and how it works. You can decide how deep into the JavaScript and Objective-C you want to delve.

The book is organized as follows:

- Chapter 1, “Developing with Dashcode and Xcode,” teaches you how to use Dashcode and Xcode together with QuickConnectiPhone and PhoneGap to quickly create fun-to-use applications that run on the iPhone. This chapter includes basic Dashcode use and methods for moving your Dashcode application into Xcode for compiling and running on devices.
- Chapter 2, “JavaScript Modularity and iPhone Applications,” teaches you how to dramatically reduce your time to market by taking advantage of the modularity of the QuickConnectiPhone framework. How front controllers, application controllers, and JavaScript reflection are used in code is explained.
- Chapter 3, “Creating iPhone User Interfaces,” helps ensure that Apple App Store distribution approves your applications. It describes best practices for creating highly usable iPhone applications. The different types of applications usually created for iPhones are described as well as pitfalls to watch out for.
- Chapter 4, “GPS, Acceleration, and Other Native Functions with QuickConnectiPhone,” shows you how to get GPS, acceleration, and device

After you run the QuickConnectFamily installer and launch Dashcode, you can find the QuickConnectiPhone template at the bottom of the Dashboard Widget template selection dialog. Double-clicking the QuickConnectiPhone icon takes you directly into the main Dashcode screen. The blank user interface displays on the screen. Figure 1.1 illustrates what the running Dashcode application looks like.

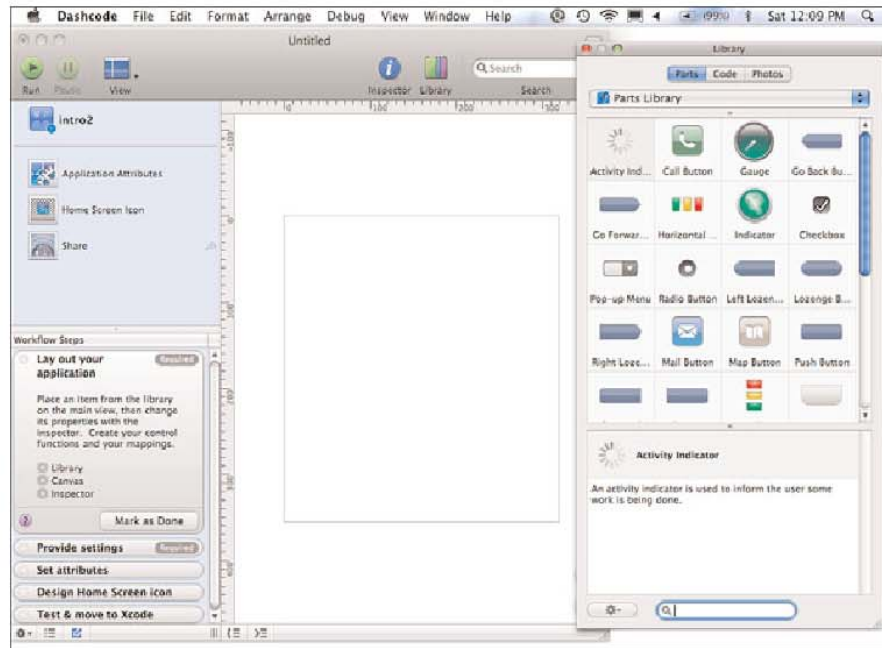


Figure 1.1 The QuickConnectiPhone template is used in Dashcode. The standard Library dialog is displayed.

To understand and easily use the files included in the framework, you must first create a simple user interface using Dashcode and deploy it to your iPhone using Xcode. The user interface created here consists of only a button and a text field. When the button is clicked, the text field displays “You did it!”

Hybrid Applications and the Alert Dialog

People accustomed to writing in JavaScript often use the Alert dialog to debug an application or notify the user of a piece of information. The `alert` function in JavaScript is actually a call into the containing browser's native code rather than something that the JavaScript engine handles.

This is not implemented in QuickConnectiPhone applications because the use of dialogs violates the iPhone user interface standards Apple provided. For debugging, you can use the Dashcode debugger. If you move your application to Xcode you can use the `debug` function to display messages in the Xcode console.

PhoneGap does provide Alert dialog functionality, but it does not provide the Xcode debug function.

To notify users of important pieces of information, insert them into an HTML div or another element regardless of which tool you use.

Remember, be alert, don't alert.

To create this user interface, be sure the Library dialog is open. If it is not, click the Library icon on the top bar of Dashcode. Then, find the Text part at the bottom of the parts library and drag it on to the blank application screen. A new text area displays at the top of your application's interface and contains the word Text. This text, by default, has a width of 100 percent. Dashcode has dynamically inserted an HTML div tag into the underlying index.html file of your application and some JavaScript to fill it with whatever text, background colors, and so on that you choose.

For this example, you change the id of the text div to `display` and empty the text field. This is done using the interface entity inspector. Selecting the Inspector icon in the top bar of Dashcode activates this dialog. Select the red and white tab in the upper-left hand corner of the Inspector, change the ID field to read `display`, and then clear the Label field.

Add a Push button to the interface by dragging and dropping it outside the text field. The inspector now displays the information for the button instead of the text field. Select the blue cube in the upper right-hand corner of the Inspector dialog. This causes the Behaviors tab to appear. This tab enables you to define JavaScript functions as handlers for any of the user interface types of events listed. Notice that many of the standard JavaScript mouse events are not seen. They have been replaced with `ongesturestart`, `ongesturechange`, and `ongestureend`. Enter `changeText` in the handler section of the `onclick` event. This inserts a `changeText` function in the `main.js` file, displays it to you, and enables you to define what should happen when the `onclick` event is fired. In this simple case, place the following code in the `changeText` function:

```
document.getElementById('display').innerHTML = 'You did it!';
```

The sample application is now ready to run in the iPhone simulator. Select the Run icon in the upper left-hand corner of Dashcode. This launches the simulator and runs your application in it. Figure 1.2 shows the simple example application running in the simulator.

Having completed and debugged the creation of the application, you can now move the code into Xcode for deployment as an installable application.

To start, you use Dashcode to deploy the current application. If you do not, the code is hidden inside your Dashcode project and has directives in it that only Dashcode can understand. Click the Share icon in the left-hand section of Dashcode to show the deployment screen. This enables you to save the completed HTML, CSS, and JavaScript to disk in a form that is ready to embed in your application. Enter a name for a new directory in the Path field to create a directory on the hard drive of your machine. The files are then

Section 3: Using Xcode and the Custom PhoneGap Template

An Xcode custom PhoneGap application template is available and can be downloaded from <https://sourceforge.net/projects/phonegapinstall>. As with the QuickConnectiPhone installer, a custom template is available in the New → Project dialog; it is called PhoneGap Application. Double-click this icon, name your project, and save it to a location on your disk just as you did with the QuickConnectiPhone Application template.

Differences

One of the major differences between QuickConnectiPhone and PhoneGap is that PhoneGap applications do not include the HTML, CSS, and JavaScript files in the installed application. These files must reside on and be served up by a web server accessible via the Internet.

If you choose to use PhoneGap, your application can be run only when the user has access to the Internet from his device. This is usually not a problem with iPhone owners, but it can be a major impediment to iPod Touch owners because they need to be near an open WiFi access point for Internet access.

If you want your application installation to be complete, use QuickConnectiPhone. If you want the CSS, HTML, and JavaScript to reside remotely, you can use PhoneGap.

Because PhoneGap applications do not include the CSS, HTML, and JavaScript files in the installed application, you need to point it to a web server that contains these files. The simplest, temporary way to do this is on your own development machine.

Open your Sharing System Preference dialog and ensure that web sharing is activated. This turns your machine into a web server. In your <UserHome> → Sites directory, create a pg_hello directory and put an index.html file in it containing the following code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
    <title>pg_example</title>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <meta name="viewport" content="minimum-scale=1.0, width=device-width, maximum-
scale=1.6">
</head>
<body>
Congratulations. You have just loaded this web page from the Internet.
</body>
</html>
```

You must inform your PhoneGap application of the web server's location by modifying the url.txt file found in the Resources group. Because you have placed this file on your own machine, replace the text in the file with `http://localhost/~<user>/`

Hybrid Applications, GPS, Acceleration, and Other Native Functions with PhoneGap

The PhoneGap library provides an alternative to using the QuickConnect framework discussed in Chapter 4, “GPS, Acceleration, and Other Native Functions with QuickConnect.” Although it does not have the full range of features found in QuickConnect, it is, along with QuickConnect, a popular choice for accessing device information and behavior. The first section of this chapter explains how to access these native iPhone functionalities using the PhoneGap JavaScript API. The second section shows the JavaScript and Objective-C code that underlies the PhoneGap JavaScript API.

Section 1: JavaScript Device Activation

Because PhoneGap is undergoing development at the time of this writing, it has a limited number of native behaviors that are available. This section covers only those behaviors that have support on both the JavaScript and Objective-C sides of the application. There are placeholders in the code for behavior code not yet complete or functioning in PhoneGap at this time. These behaviors are not covered.

Table 5.1 lists the methods, functions, and attributes that are part of the working API. Unlike QuickConnectiPhone, these do not belong to a larger framework. As such, you are responsible for calling them directly. Because there is little safety checking in the active code, be sure that parameters passed to these functions are valid and not null.

As discussed in Chapter 4, QuickConnectiPhone includes the HTML, CSS, and JavaScript in the installed application on the device. PhoneGap does not. Instead, at startup, a generic Objective-C application, renamed as your application, temporarily loads these files from a web server to which they had previously been published (see Figure 5.1). To run the application, your device must have network connectivity. Therefore, do not expect your application to run correctly when the user is on an airplane or where connectivity is not available.

PhoneGap Application Startup Process

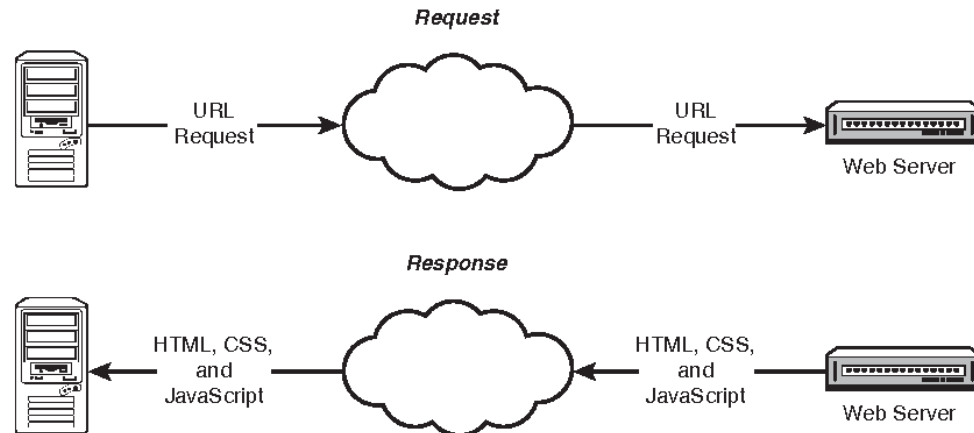


Figure 5.1 The startup request and response for PhoneGap applications used to obtain the HTML, CSS, and JavaScript that define the application

When a PhoneGap application starts loading its web components from the server, a call is made from the generic Objective-C application to JavaScript. This call sets a series of global variables that describe the device the application runs on. The `Device.init` method of the JavaScript API, usually called in a function that is assigned to be the `onload` event listener, exists to gather up the global variables set by the Objective-C portion of the application and stores them in attributes of PhoneGap's `Device` object. The Objective-C code that sets these device variables is discussed in Section 2.

```

init: function(model, version) {
    .
    .
    .

    Device.available = __gap;
    Device.model = __gap_device_model;
    Device.version = __gap_device_version;
    Device.gapVersion = __gap_version;
    Device.uuid = __gap_device_uniqueid;
    .
    .
    .
}

```

The previous example shows code relevant to the iPhone and iPod Touch devices. Each of the global value names begins with `__` (double underscore) and has a unique value. The `__gap` variable is used as a flag to indicate whether or not the JavaScript is run from within a PhoneGap application. This is required because at the time of this book's writing, the application's HTML, CSS, and JavaScript files are not included in any compiled, installed PhoneGap application but downloaded by the wrapper from the Internet. If your application's HTML, CSS, and JavaScript files are accessed by a web browser instead of a PhoneGap application, the `__gap` variable would be null and can be used as a flag later in your code.

The `__gap_device_model` variable contains a string that describes the device your application is running on, such as iPhone, iPod Touch, or iPhone Simulator. This variable also includes the device's OS version, which is in the `__gap_device_version` global variable. The device's globally unique id is in the `__gap_device_uniqueid` variable, and the version of PhoneGap is found in `__gap_version`. Each of these values is available for use either from the public device attributes seen in the snippet or from the global variables anywhere in your code. Table 5.1 describes each of the JavaScript functions that allow you to access device behaviors.

Table 5.1 PhoneGap JavaScript API

Element	Parameters	Behavior
<code>gotAcceleration()</code>	x—The acceleration value along the x-axis. y—The acceleration value along the y-axis. z—The acceleration value along the z-axis.	This function is not predefined. To receive accelerometer information, create this function somewhere in your application. When created, the Objective-C library can then call it and pass it the parameter values.
<code>Device.init()</code>	None	Gathers up the device information from global variables that were set when the application started. This information is placed in the Device object and includes: - Device type (iPhone/iPodTouch) - Device OS version - Device UniqueID - PhoneGap version
<code>Device.vibrate()</code>	None	Causes the phone to vibrate for the standard amount of time.

Table 5.1 PhoneGap JavaScript API

Element	Parameters	Behavior
<code>Device.sound()</code>	clip—A string that is the name and type of the sound file to be played. Example: "tweet.wav"	A sound file with this name must be in the Resources of your application or an execution error will happen.
<code>Device.Location.init()</code>	None	Causes location information already on the Objective-C side of the application to be sent to the JavaScript side for processing and/or display.
<code>Device.Location.callback</code>		This attribute of the Location object when set to a custom callback function causes a callback function to be executed when the GPS information is available.
<code>Device.Location.wait()</code>	func—A callback function is executed when the GPS information is available.	This is an alternative to the <code>Device.Location.init()</code> function.
<code>Device.exec()</code>	command—A command string that is passed to the Objective-C side of the application for processing. Examples: "vibrate," "sound," and "getLoc"	This is the function that handles all communication to the Objective-C side of the application. Each command sent triggers different native device behavior. This is the method you call from JavaScript if you created custom Objective-C code that you want executed.

The PGDeviceCatalog sample application includes a `vibrate` button that when clicked causes the phone to shake. The button's `onclick` event handler function is called `vibrateDevice`. This function calls the `Device.vibrate` method that causes the device to vibrate.

```
function vibrateDevice(event)
{
    Device.vibrate();
}
```

The following `Device.vibrate` method is a façade for the `Device` object's `exec` method. It calls `Device.exec` and passes it the `vibrate` command. All the PhoneGap

device behavior methods are actually façades of `Device.exec`, which is similar to `Device.vibrate`.

```
vibrate: function() {
    return Device.exec("vibrate")
}
```

The `Device.exec` function used in the previous code exists completely in JavaScript and is shown in the following code. Like `QuickConnectiPhone`, `PhoneGap` sets the URL and creates a message consisting of the URL and the command, such as `vibrate`, to be sent to an Objective-C component that is part of the underlying `PhoneGap` framework. This component is programmed to terminate the loading of a new page and evaluate which command has been sent. For more information on this process, see Section 2.

```
exec: function(command) {
    if (Device.available) {
        try {
            document.location = "gap:" + command;
        } catch(e) {
            console.log("Command '" + command +
                "' has not been executed, because of exception: "
                + e);
            alert("Error executing command '" +
                command + "'.");
        }
    }
}
```

In the catch portion of the previous code, there are two ways of notifying the user that something has gone wrong. The first, a `console.log` call, writes a message to Dashcode's console and works only when running in Dashcode, not on the device.

The second message uses an alert dialog. Because `PhoneGap` has implemented alert behavior in Objective-C, this is active on the device but according to Apple should never be used in iPhone applications. For more information on iPhone application user interface design, see Chapter 3, “Creating iPhone User Interfaces.”

The `playSound` and `vibrate` methods are both unidirectional, which means there is communication from JavaScript to Objective-C with no data expected back. The `Device.Location.init` method is bidirectional and therefore expects to receive data from the Objective-C `PhoneGap` library. As shown in the following code, this `init` method is also a façade for the `Device.exec` method. In this case, the command passed is `getLocation` as opposed to the `vibrate` command shown earlier.

Figure 5.2 shows a running `PhoneGap` application that has requested GPS location information. GPS information is also accessed by a façade that calls `Device.exec`.

```
init: function() {
    .
    .
}
```

Appendix C

The PhoneGap Development Roadmap

Because PhoneGap is early in its development cycle, it is rapidly undergoing change. As such, it is important to closely watch the development for updates. Table C.1 is from the PhoneGap wiki. It shows what developers claim about the availability of functions and what has been confirmed as of the time of writing this book.

Confirmed means JavaScript functions are found in PhoneGap as of February 23, 2009. The author is unable to comment on the availability of the Blackberry platform. *In Work* means that the PhoneGap team is working to make this feature available. *Claimed* means that the PhoneGap team states that the feature is available, but the author cannot find evidence of this in the downloaded code. Dashed cells indicate that work is not being done and does not currently exist. *Not Possible* indicates that the developers believe that the functionality indicated is not accessible on the device.

Note

Development for Symbian and Windows Mobile is planned, but no work has been done; therefore, they are not in Table C.1.

Table C.1 PhoneGap Development Roadmap on February 23, 2009

	iPhone	Android	Blackberry (OS 4.5)
Geolocation	Confirmed	Confirmed	Claimed
Accelerometer	Confirmed	Confirmed	Available in OS 4.7
Camera	In Work	In Work	In Work
Vibration	Confirmed	Confirmed	Claimed
Offline (local files)	In Work	Claimed	In Work
Contacts API	Claimed	—	Claimed
SQLite wrapper	Claimed	—	Not Possible