

THE EXPERT'S VOICE® IN APP DEVELOPMENT

Beginning App Development with Parse and PhoneGap

*FROM APP DESIGNER TO
APP DEVELOPER*

Wilkins Fernandez and Stephan Alber

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.



Apress®

Contents at a Glance

About the Authors	xix
About the Technical Reviewer	xxi
Acknowledgments	xxiii
Preface	xxv
■ Chapter 1: Introduction	1
■ Chapter 2: Beginning PhoneGap	15
■ Chapter 3: Beginning Parse	35
■ Chapter 4: Tools and Helpers	67
■ Chapter 5: Facebook API	79
■ Chapter 6: The Messenger Application	109
■ Chapter 7: User Registration with Parse	121
■ Chapter 8: Messages	167
■ Chapter 9: Location Services	195
■ Chapter 10: Map Views	223
■ Chapter 11: Accessing and Sharing Photos	245
■ Chapter 12: Network Connection Status	259
Index	267

CHAPTER 1



Introduction

Let's start with the basics! What is PhoneGap? What is Parse? Even if you already know about PhoneGap or Parse, you will still find it worthwhile to read this section, as we'll explain the motivation behind and benefits of combining the two.

After providing a brief description of PhoneGap and Parse, we will talk about previous knowledge and requirements, discuss some code standards, and present a general overview of what you will learn about PhoneGap and Parse.

What Is PhoneGap?

PhoneGap is an open source project used to build mobile applications. It takes your JavaScript, HTML, and CSS code, and packages them into an executable program that can run an array of mobile devices.

PhoneGap does this by exposing access to a device's native features such as file and camera access, using what is known as *Foreign Function Interface* (FFI). This interface lets you invoke platform specific native code using JavaScript. This allows background tasks to send data from JavaScript to native code and vice versa. FFI also allows developers access to native user interface features, such as showing the system's dialog.

PhoneGap makes writing native applications possible by using a plug-in system. This system permits developers to add custom features to their applications. Plug-ins are both maintained by the PhoneGap team and the open source community. We'll be exploring some of them later in this book.

History

The PhoneGap project history and its naming are somewhat bewildering. The project was born at a hackathon in 2008 at the iPhoneDevCamp, which was later renamed iOSDevCamp. In the following years, a company named Nitobi, located in Vancouver, Canada, further developed the software.

In 2011, Nitobi was acquired by Adobe. As a part of the acquisition, the source code of the project was donated to the *Apache Software Foundation* (ASF), allowing the core of the PhoneGap project to remain open sourced. This open source part of the project is named *Apache Cordova* (see Figure 1-1).



Figure 1-1. Apache Cordova homepage: cordova.apache.org

This begs the question: What’s the difference between PhoneGap and Cordova? In short, PhoneGap is Cordova plus Adobe services and extensions that further enhance its capabilities. Cordova can still be used independently.

How PhoneGap Works

The first task is rather easy to understand by imagining your device’s browser without header and footer bars, history, bookmarks, windows, and so forth. What’s left is the WebView, your playground that you can use in full-screen mode to build an application using the web technologies HTML, CSS, and JavaScript.

The second part—exposing the device’s native features—is a more complex subject. Using code samples is the best way to explain this concept. Before we look at code samples, let’s come back to the original issue, which is that while we can do many things in JavaScript, there are some things we still cannot do. For example, let’s say you want to use the Parse iOS *Software Development Kit* (SDK) in your application. How can you use this SDK in JavaScript? You can’t. PhoneGap closes this gap, making native device and operating system capabilities accessible.

Foreign Function Interface

The FFI system enables developers to build a bridge between JavaScript and native code for a very specific task. Collections of these foreign function calls are commonly bundled in plug-ins, while a wide set of basic features comes shipped with the Cordova or PhoneGap base package.

PhoneGap offers a broader feature set, which is required to build your mobile application. For example, in Chapter 2, you will learn about the PhoneGap *command line interface* (CLI). This tool set allows you to build and maintain PhoneGap applications by executing terminal commands.

Using the CLI, you’ll be able to package and compile your application files, as well as test your application using virtual emulators or physical devices. These tools will help you to prepare your application for distribution on marketplaces such as the Apple Store or Google Play.

Supported Platforms

PhoneGap applications are able to run on many platforms using only one code base. This book focuses on developing with the most popular platforms—iOS and Android. Although PhoneGap is primarily used for mobile devices, it is possible to apply the concepts covered here to a desktop platform such as Windows 8.

As you have already learned, PhoneGap projects can be extended with the use of *plug-ins*: concise scripts designed to add extra functionality to an application. As wonderful as they are, plug-ins must be used responsibly due to the possibility of quirks between devices.

For example, the *battery status* application programming interface (API) for Windows Phone 7 and 8 does not yet provide a way to determine the battery level of the device. It *does*, however, provide a way to tell if the device is plugged in or not. You may often run into situations such as this one where it will take some creativity to get the behavior you want.

The examples in this book will demonstrate some of the core plug-ins that come standard with PhoneGap; these features cover all supported platforms, as shown in Figure 1-2.



Figure 1-2. Compatible PhoneGap devices starting from the left: Android, iPhone and Apple devices, Tizen (formally bada), BlackBerry OS, Firefox OS, ubuntu, webOS, Windows Phone

■ **Note** In 2014, PhoneGap stopped actively supporting webOS, Symbian, BlackBerry, and Windows Phone 7 platforms that use Cordova versions below 3.0 (currently v5.0.0) in favor of supporting current and widely used platforms. If you want to use PhoneGap to target these platforms, you can install older versions of Cordova to build your application.

PhoneGap vs. Web Applications

Writing a PhoneGap application isn't the same as building a web site. PhoneGap believes that writing code in the languages that make up web pages should also be used to communicate with hardware devices.

When programming with PhoneGap, we intend to interface with a mobile device. A PhoneGap application shouldn't just be a packaged static web site that doesn't take advantage of its API capabilities. In fact, Apple may refuse your application to the App Store if they feel your application is just a bundled web site that provides no entertainment value. In light of this, there are many examples of successful PhoneGap applications available on the market today.

Building and Testing PhoneGap Applications

Mobile devices typically have a web browser out of the box. The basic technology that renders a web page can also be used to create an installable application with PhoneGap.

The PhoneGap API was designed specifically for interfacing with hardware on mobile devices. This means that PhoneGap applications will not work on a typical web browser, as you would expect on a personal computer.

Web browsers use HTML and CSS for styling, positioning, and animating content elements (such as text, buttons, images, and so forth). JavaScript is used for adding dynamic interactivity, giving the user a unique experience. In this trio of languages, JavaScript arguably plays the most important role.

There are a variety of ways to build and test PhoneGap applications. The good news is that you only need one code base to create an installable package for each compatible device. We will be looking at different ways to test your applications, such as using device emulators and live testing on a compatible mobile device.

For example, if you plan on developing an application for an Apple device such as iPhone, you will need to use a program called *Xcode* installed on a Mac computer. Using the developer tools that Xcode comes with, combined with additional CLI tools, developers can test applications using the *iOS simulator*.

As you may expect, developers face several challenges when writing mobile applications for multiple device platforms. Programming and testing applications natively typically requires its own development tools and setup.

Does this mean you need to have a custom set development environment for *each* platform when using PhoneGap? Yes. For proper testing of your user interface and application behavior, you need to test each platform's version of your application in its proper environment. This applies in particular to plug-ins accessing native device capabilities.

In any case, developing PhoneGap applications for multiple platforms only requires one code base. With that in mind, you may find yourself having to make adjustments to your code in order to provide the best experience for users across operating systems or devices.

Once you are ready to release your application to the world, each marketplace expects you to provide a package type that is unique to its store. The only way to do so is to have PhoneGap compile your application in the environment that it is intended for. For example, if you are developing in an iOS environment and want to release your application to Google Play, you'll need to build your application using Android-based IDE tools Android Studio.

The Adobe PhoneGap Build Service (Optional)

Since teaming up with PhoneGap, Adobe now offers a service that enables developers to build their mobile applications for multiple devices all in one shot. As you know, there are many platforms that PhoneGap can compile to. There is also a unique development environment for each platform; things can get messy quickly.

There are some potential pitfalls with using this service, which we'll examine next. First, let's go over some of the benefits of using Adobe PhoneGap Build.

If you are like most people, you may not have access to the devices and development environments needed for compiling a PhoneGap application. Because of this, Adobe offers the build process in the cloud, allowing developers to package mobile applications simultaneously for all available platforms. This ensures that your application is updated with the latest native SDKs.

With new mobile devices coming out every few months, you can imagine the complexities involved with developers maintaining and supporting new, current, and legacy operating systems. With PhoneGap applications, you can maintain your code with confidence, knowing that you're using open standards that are supported by major platforms.

PhoneGap Build expands your application's audience reach by providing multiple packaged platform builds. This means you no longer need to write software in proprietary vendor code.

Without having to maintain multiple native SDKs, the Adobe PhoneGap build service does it all for you. This optional service allows you to focus more on writing great software, not worrying about how to get it to your users.

Adobe offers a three-tiered plan, each tier offering unique options. Figure 1-3 illustrates each plan in detail, as of this writing. Learn more at build.phonegap.com/plans.



	Free Plan	Paid Plan	Adobe Creative Cloud Membership 
open source apps	∞ unlimited must be pulled from a public Github repo		
private apps 	1	25	
max app size	50 MB	100 MB	1 GB
core cordova plugins <small>* a list of these plugins is here</small>	YES		
third party plugins <small>* includes plugins from npmjs.com, plugins.cordova.io as well as our own repository</small>	YES		
upload plugins <small>* includes plugins only you can use</small>	NO	YES	
collaborators	∞ unlimited invite people to your app as either developers or testers		
	completely free	starting at \$9.99/mo	sign in with your Adobe ID

Figure 1-3. Adobe PhoneGap Build services

When developing a PhoneGap application on your computer, you still need a way to actively test what you're building. Chapter 2 focuses on the tools and software you'll need for the targeted platform.

Caveats

There are some things to consider before using the Adobe PhoneGap Build service. Technically, you don't *need* to install any platform-specific development environment, such as Xcode, to develop PhoneGap applications. Following this logic, because this service builds your application for you, there is no need to install software locally. Although this may be true, we recommend, at least if you're getting started, using the software that is designed to build your application locally. This can give you a better understanding of the platform and its capabilities.

The Adobe PhoneGap Build service compiles your HTML, CSS, and JS code into the appropriate packages for operating systems. It is your responsibility as a developer to submit your compiled application to its respective app store for distribution.

■ **Note** Developing PhoneGap applications for Apple devices requires the use of the proprietary software Xcode to create certificates for an iOS application. These policies and protocols established by Apple must be implemented in order to distribute your application.

Why We Don't Use Adobe PhoneGap Build in This Book

Debugging and testing PhoneGap applications in their native environment is easier using a local development setup. Having a development environment while you are coding increases your chances of avoiding potential bugs in your application. Without a testing environment, you run the risk of compiling and distributing a broken application to your users without knowing it. We feel it is important to use discretion and program safely. Learn more at <http://build.phonegap.com>.

Collaboration and Testing Tools

If you would like to collaborate with other people on an application, we recommend using free services such as Github or Bitbucket for hosting your code.

For distributing test release software, other free solutions are available. For example, with iOS you can use a service called *TestFlight*. Once you're ready to share and test your application, use it to send updates to beta testers for immediate feedback. Android applications can be distributed using the *Google Play Developer Console*.

Using Parse

Parse is the perfect companion for building a highly scalable web or native application. Acting as data storage, Parse is capable of saving things like photos, geo location, and other basic data types.

By providing features that would typically be handled by a variety of technologies, Parse bundles together features such as push notifications and scheduled tasks using its intuitive API. Gone are the days when you need to host your own databases and sync them to your applications. Now that everything can be handled "in the cloud," saving and managing data have never been easier.

Parse was created by a group of developers who got together to create a set of back-end tools and services to help manage all the facets of mobile development. In 2013, Facebook acquired Parse with the intent of further expanding its already amazing feature set.

Developing Applications with Parse

Parse offers its services for use with many different programming languages. For this book, we'll be using the JavaScript SDK, as it's perfect for using with PhoneGap.

Parse offers different tiers of free and paid services based on your needs. When your application reaches the point where your users are making more than 30 requests per second, Parse offers to extend the free service by providing more file storage, as well as a higher data transfer rate.

For our purposes, a free account will suffice. In fact, this would be the case for most new applications. A free account gives you up to 20 gigs of data and file storage respectively for *each* Parse application you create. That's a lot of data! Not only that, it also offers the ability to run what it calls "background jobs." These are programs that you can write to run on the cloud at any given time. This is helpful for things like providing mass updates to your app or introducing new features to your users in real time.

Features

We'll be exploring different aspects of Parse throughout this book. To give you an idea of how awesome Parse is, here are just a few cool things that come with a registered account:

- *Analytics*: Instead of just having to rely on third-party scripts to track your apps usage, Parse offers an analytics tool without your having to configure anything.
- *Parse push*: Using any of the SDKs, you'll be able to send up to one million unique recipients push notifications a month.
- *User management*: In place of creating your own database structure and code for user accounts, Parse offers this functionality right out of the box. This includes things like resetting passwords, linking accounts with Facebook, and many other conveniences.
- *Data browser*: Using the web interface, you can browse all of your application's data with just a few clicks. It also allows you to create new data objects that allow you to save custom data sets, while also providing pre-defined data structures such as the User or Product classes.

Why Parse and PhoneGap?

Of all the frameworks, in all the languages, why did we choose to go with Parse and PhoneGap? The answer is easy: To build a successful application, you need a user interface technology—the *front end*—and a data infrastructure—the *back-end*. While PhoneGap is doing a good job on the front end, the project doesn't cover back-end needs.

With PhoneGap alone, you are forced to learn another programming language and design your back-end system on your own. If you want to save data to the back end, you needed to write a front-end module to prepare the data, connect to a server, and send the data to it. In the back-end tier, the request needs to be authenticated, processed, and then stored to a database. This is typically done using another system with its own language, such as MySQL. The same applies to the task of reading data.

With Parse, this approach is a thing of the past. Using PhoneGap *together* with Parse will enable you to use only *one* programming language—JavaScript—to manage all your data needs: reading, saving, deleting, defining schemas, authentication, querying, and so forth.

To demonstrate how easy it is to save data in Parse, we'll use a simple code example: In Listing 1-1, we initialize the Parse JavaScript SDK, which opens a connection to the Parse backend. This back-end service will store your data securely and efficiently “in the cloud.”

To get an idea of how data is stored using Parse, Listing 1-1 demonstrates creating a class that we save a new record to. We start by defining a new Parse class named Toy, then create a new *instance* of the class, set some sample values, and finally save the object to the database using `myToy.save()`.

Listing 1-1. Saving a Simple Object to the Parse Cloud in JavaScript

```
// Initialize Parse JavaScript SDK
Parse.initialize("APPLICATION ID", "JAVASCRIPT KEY");
// Define a new Parse class named "Toy"
var Toy = Parse.Object.extend("Toy");
// Create a new Instance of the Toy Parse Class
var myToy = new Toy();
```

```
// Set some Example Values
myToy.set("name", "Rocking Horse");
myToy.set("color", "brown");
// Save Object to Parse Cloud Database
myToy.save();
```

That's it! Saving data has never been easier. In a production application, we *could* add more complexity to these tasks, like success and error callbacks, but in a perfect world this code will actually work as is. Figure 1-4 shows how the result from Listing 1-1 would look when it is executed and viewed in Parse's data browser.

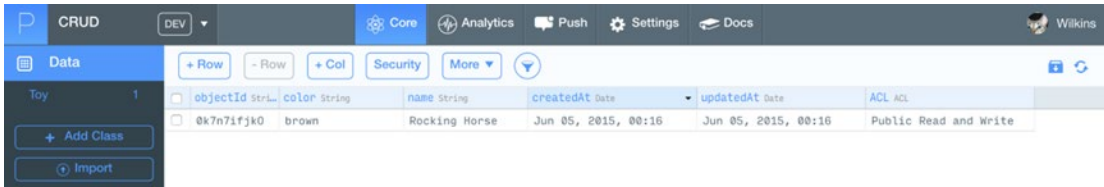


Figure 1-4. The result of Listing 1-1

Yet saving data is just one example how Parse and PhoneGap complement each other. We show a wider span of applications in Table 1-1. These samples will assure you that joining these two forces just seems natural.

Table 1-1. Complementary Functionality between PhoneGap and Parse

PhoneGap	Parse
Access camera and audio functionality of the native operating systems	Has a database field that is specifically used for media such as images, video, and audio files
Get the geo position of a device, pinpointing a user's location anywhere in the world	Has a data field for geo location, allowing you to save the latitude and longitude and query data by location and distance
Access the contact list a user has on his or her phone, providing information that can enhance your application	Stores media, text, and all basic database field types in the cloud. Can store user profiles, content, and any other associated data for your needs
Access the native (push) notification system of Android and iOS devices via PhoneGap plug-ins	Offers a push notification service for both Android and iOS devices
Compiles your application code for multiple platforms	Allows to share data between platforms (iOS, Android, Web)

Make Your Application Social

Humans have a need to communicate with each other. Using Parse and PhoneGap together to create an application allows you to facilitate this need by allowing you to share. This social component can be very important in the success of your application. After all, it's *Words with Friends*, not *Words with Myself*. Building static, nonsocial applications is a thing of the past.

PhoneGap and Parse together stand by your side when building social applications. As an example, you can use the Parse Facebook single sign-on login feature to sign up to an application and connect with friends. You can then use PhoneGap's technology to capture a picture and use Parse again to store it to the cloud and share it with your friends.

Adding a social element to your application does more than expose your work to new users. It can also add a personal touch that can make your application feel special. We'll show you how to build a full social application starting in Chapter 6 that will demonstrate how to keep people connected. We'll be doing all sorts of cool things like accessing media, location sharing, and providing real-time updates.

Previous Knowledge and Requirements

There are a few core concepts in modern front-end web development that need to be understood before continuing. These will provide you with a foundation that you can build upon while developing multiple skills. There are plenty of approaches you can take for creating a PhoneGap application, and you don't have to decide on anything *before* starting a new project. However, knowing the fundamentals will aid you in taking the best approach for each application.

There are many books that cover the delicate intricacies of HTML, JavaScript, and CSS. Because of this, we'll only highlight some of the key concepts that you'll need to get the most out of this book. We encourage you to do further research on any topics you find challenging.

Whatever level of programming you are at right now, consider the following section a refresher on some of the concepts that need to be understood to get the most out of this book.

JavaScript Object Literals

If you're just getting into programming, the phrase *object-oriented programming* can sound frightening. Objects in programming are analogous to real-world tangible objects that have properties and values. If you were to consider yourself as an object, the *value* of your `firstName` property would be *your* first name.

An object literal can be thought of in the same way, with some rules: *the value of a property name must fall within a predefined type of data*. This basically means you need to use valid data. All values in JavaScript are considered objects, so anything can be stored into an object. An object literal is enclosed in a set of curly braces, like this `{ key: value }`. An empty set of braces (`{}`) is considered a valid object; it has zero properties and values. The following code demonstrates JavaScript's primitive types of data by assigning them to an object that is referenced in the variable named `obj`:

```
var obj = {
  name: "value",
  array: [4,2,0],
  boolean: true,
  type: "string",
  number: 0,
  emptyObject: {},
  fn: function(){
    var declaredButNoValueAssigned;
    console.log("fn scoped variable: " + declaredButNoValueAssigned);
    var emptyValue = null;
    // This function returns a null value
    return emptyValue;
  }
};
```

```

console.log(obj.name);           // value
console.log(obj["name"]);       // value
console.log(typeof obj.name);   // string
console.log(typeof obj.type);   // string
console.log(obj.type);          // string
console.log(obj.array);         // [4,2,0]
console.log(typeof obj.boolean);// boolean
console.log(typeof {});        // object
console.log(obj.fn());
// fn scoped variable: undefined
// null

```

Namespaces with Object Literals

Object literals are a great way to organize related functionality. The way programmers write JavaScript applications has developed over the years since its creation in 1995. A common and powerful concept that JavaScript shares with other programming languages is that of *namespaces*. From server-side scripting with Node.js, to client-side and installable applications with PhoneGap, namespaces allow developers to communicate concepts into understandable programmatic interfaces.

All programming languages have unique ways of expressing objects and ideas. Each has its own syntactical rules that need be followed in order to use it. What all programming languages have in common is the use of language itself: strict syntax that is used to communicate concepts that a machine can understand. In JavaScript, *object literals* help developers encapsulate logic in an expressive and organized way that can be understood by others. In Chapter 2, you'll see this put to use in a full example using PhoneGap. Listing 1-2 demonstrates how a fictional JavaScript program may be written using namespaces.

Listing 1-2. Creating a Simple JavaScript Using Namespaces

```

var app = {
  settings: {
    version: "0.0.1"
  }
};
app.user = {};
app.user.register = function(){};

```

To get an even deeper understanding of namespaces, we recommend reading a great article by engineer and developer advocate Addy Osmani titled “Essential JavaScript Namespacing Patterns.”¹

Organizing Code for Projects in This Book

The code samples and concepts in this book can be combined with many JavaScript projects, libraries, and frameworks. As a matter of fact, a part of the JavaScript Parse SDK is forked from Backbone.js; existing developers will find converting existing applications intuitive. Some demonstrations in this book use third-party libraries such as jQuery. However, we do also use plain ole’ JavaScript as well. With each chapter, we explore both the PhoneGap and Parse APIs by demonstrating functionality that you could use in your next application.

¹Read the “Essential JavaScript Namespacing Patterns” article at <http://addyosmani.com/blog/essential-js-namespacing/>.

All projects use some form of file structure for organizing different aspects of an application. Chapter 2 explores what a typical PhoneGap application looks like by using the “Hello World” starter application. Parse also has a starter application that we dig into in Chapter 3.

Aside from the default folder structures found in starter Parse and PhoneGap projects, we’ll be using intuitive naming conventions in an attempt to keep things simple, clear, and concise. The following snippet demonstrates a typical folder structure for web-based projects with `index.html` located at the root of the folder:

```
img/
js/
css/
index.html
```

The example structure above contains three directories and one HTML file, the home page. The folders are named in an obvious way as to make it clear what it’s responsible for containing. In Chapter 6, you will start creating a Parse and PhoneGap application that will have a similar structure, while still respecting the required directory structure needed for compiling PhoneGap applications.

Loading Scripts with LABjs

There are many ways to load scripts in JavaScript. From using classic `<script>` tags to building complex modular compiler systems, loading scripts in JavaScript is always an important task. We decided to go with what we find is one of the most straightforward and simple ways of loading scripts: a library named LABjs. LABjs is an open source project written by Open Web Evangelist, Kyle Simpson.

Chapter 6 demonstrates the necessity of using a script loader, in which we’ll begin building an entire application from scratch. There will be *lots* of scripts in this application, and in order to manage them, we will use LABjs to do the hard work.

You can see a comparison of traditional loading vs. loading using LABjs in Listings 1-3 and 1-4.

Listing 1-3. Traditional Script Loading Using the `<script>` Tag

```
<script src="framework.js"></script>
<script src="plugin.framework.js"></script>
<script src="myplugin.framework.js"></script>
<script src="init.js"></script>
```

Listing 1-4. Script Loading using LABjs

```
<script>
  $LAB
    .script("framework.js").wait()
    .script("plugin.framework.js")
    .script("myplugin.framework.js").wait()
    .script("init.js").wait();
</script>
```

Although it may not look like much, using this script loader is an efficient way to load scripts and, more importantly, manage dependencies using the `wait()` method. We’ll be going through this in extensive detail as the application progresses in the chapters following Chapter 6.

Build Process

A *build process* is a series of automated operations that aids developers in outputting production-ready applications. We will not be covering this topic in this book, but thought we'd acknowledge its use and efficiency.

Again, when it concerns JavaScript, everyone has an opinion. What the JavaScript community *can* agree on is trying our hardest to deliver the best experience for our users. In doing so, there have been a few popular tools that aid in this process, including the following:

- Grunt.js
- Gulp.js
- Yeoman.io
- Brunch.io

There are numerous plug-ins for each of these tools designed to make your life as a developer run more smoothly. Processes such as JSHint, SASS/LESS compilation, minification, image compression, and many more are available with build-processing tools.

Debugging

Software that has yet to be written is the only kind without bugs. This means that every time you write code is an opportunity for a bug to appear. We will be introducing some basic debugging techniques that are specific to PhoneGap development. PhoneGap applications aren't like typical web applications and need special attention when it comes to debugging issues. We'll be going through a few ways of testing PhoneGap applications in Chapter 4. Additionally, we'll be covering ways of testing database interactions with Parse by using their Data Browser, among other tools and techniques.

Basic knowledge of web developer tools such as the browser inspector should be understood. It's OK if you're new to it; we'll be covering what you need to know to make the most out of each application you build. The most common and basic way to see what's going on during code execution is to log statements to the console. This technique is often used for building web sites.

Command-Line Interface Tools

A command-line user interface allows programmers to communicate with a computer by executing statements using an interactive console. If you're not familiar with executing code from the command line, we have got you covered. After installing a few command-line based tools for using Parse and PhoneGap, you'll be experienced in no time. All tools will both work in Terminal (Mac) and Command Prompt (Windows).

After installing packages for Parse and PhoneGap development, we'll be experimenting with various aspects of each API, like syncing Parse apps to your local environment for publishing, as well as compiling and previewing PhoneGap applications using an emulator.

Integrated Development Environment (IDE)

An IDE is where all the magic happens. And by *magic*, we mean coding. It's the software you use to *write* software. There are a plethora of options to choose from. From command-line interface editors like *vim*, to basic text editors like Notepad or textEdit, practically anything can be used to create a web site.

While some are more powerful and fully featured than others, the IDE you use is up to you. Some of our recommendations include the following:

- Atom.io
- Sublime Text 2
- NotePad++

Any of these IDEs will get the job done. If you already use an editor you know and love, great! But if you haven't found one, here are some things to consider:

- Is it free? Is paying for one worth it?
- Are there plug-ins that can extend its functionality?
- Does it contain any built-in features like code collapsing, autofill, or syntax highlighting?

What You Will Learn about PhoneGap and Parse

Chapter 2 will provide you with everything you need to begin application development with PhoneGap. From installing and setting up, to previewing and editing an example application, to implementing real-world scenarios that demonstrate some of PhoneGap's core functionality, we'll attempt to cover as much of their API as possible.

We'll also be covering some cool plug-ins that extend the functionality of your applications, as well as some debugging and development tools that you can start using immediately.

Using Parse as our main data storage, we'll be covering everything from creating an account to setting up your first Parse application. As the book progresses, we'll introduce some core features of Parse. We'll be covering the Parse JavaScript SDK, which we'll tie into PhoneGap applications. This will enable you to do things like take a picture using the native camera on a mobile device and saving it to a database.

Going Further

The code and concepts used for examples in this book can be applied to any PhoneGap or Parse application you create. Because we are using the language of the web (HTML, JS, CSS), there is opportunity to build a variety of different experiences for your users per application. For example:

- Storing media with Parse: audio, image, and video storage
- Accessing media content with PhoneGap
- Creating user accounts with Parse
- Sync Facebook accounts with your Parse app
- Using PhoneGap plug-ins to extend device features
- Structuring your application data using the Parse Data Browser

We are convinced that this book contains enough information to get you building applications that you have only imagined. Aside from the hands-on projects that we'll be walking you through, we encourage you to take everything you learn and apply it to your next project.

CHAPTER 2



Beginning PhoneGap

Configuring your Development Environment

In this chapter, you'll learn the essentials of PhoneGap. First, we will walk you through the installation process of PhoneGap and its dependencies. Then you'll be creating, debugging, and testing your first "Hello World" PhoneGap application.

We get underway with setting up your development environment for the two most popular mobile platforms, iOS and Android. You'll install the package manager for Node.js, which will lead to the PhoneGap installation process.

Platform Setup and Restrictions

While PhoneGap is designed to use one codebase to handle multiple platforms, you still need to prepare your system for *each* platform you wish to support. For example, with *Android* you'll need to install the Android SDK, with the *Windows Phone* you will need to install the Windows Phone SDK, and so forth.

Each platform may have further dependencies or operating system restrictions. For instance, you need a Windows system to build Windows Phone applications and a Mac OS X system to build iOS applications. While the Android SDK can run on Windows and Mac, it requires the installation of the Java Development Kit (JDK). Requirements and installation procedures will be covered in the following sections.

Installing Node.js and Node Package Manager

Before installing PhoneGap and its dependencies, you need to install Node.js. Node.js is a server side JavaScript programming environment used to build fast, scalable network applications. When Node.js is installed on your computer, it includes a JavaScript package manager called node package manager (npm), which we'll be using to install PhoneGap and other tools.

Even though npm was intended for use with Node.js and JavaScript, it may be used for source files using other programming languages as well. You can install JavaScript packages, called *node modules*, by using the `npm install <package-name>` terminal command. Download Node.js from <http://nodejs.org/download/> and follow the installation instructions.

Node.js is available for both Windows and Mac. To ensure you have installed Node.js properly, in your terminal, run `node -v`. This should return the currently installed version of Node.js on your system.

iOS Environment Setup

If you want to develop iOS applications, you need a Mac OS X computer. If you are a Windows user, this doesn't necessarily mean you need run to the nearest Apple store and buy a Mac. It is possible, but *not* recommended, to simulate an installation of Mac OS X on Windows using virtualization software like *VMWare*, or the open source alternative *VirtualBox*.

Once you either have a Mac or virtual machine configured and running, you need to install Apple's IDE software called *Xcode*. With *Xcode*, you can build applications for OS X and iOS. You'll only need the iOS portion for this book.

Figure 2-1 shows the *Xcode* IDE for Mac. It also shows how the iOS simulator looks when running the starter project that comes with all PhoneGap applications.

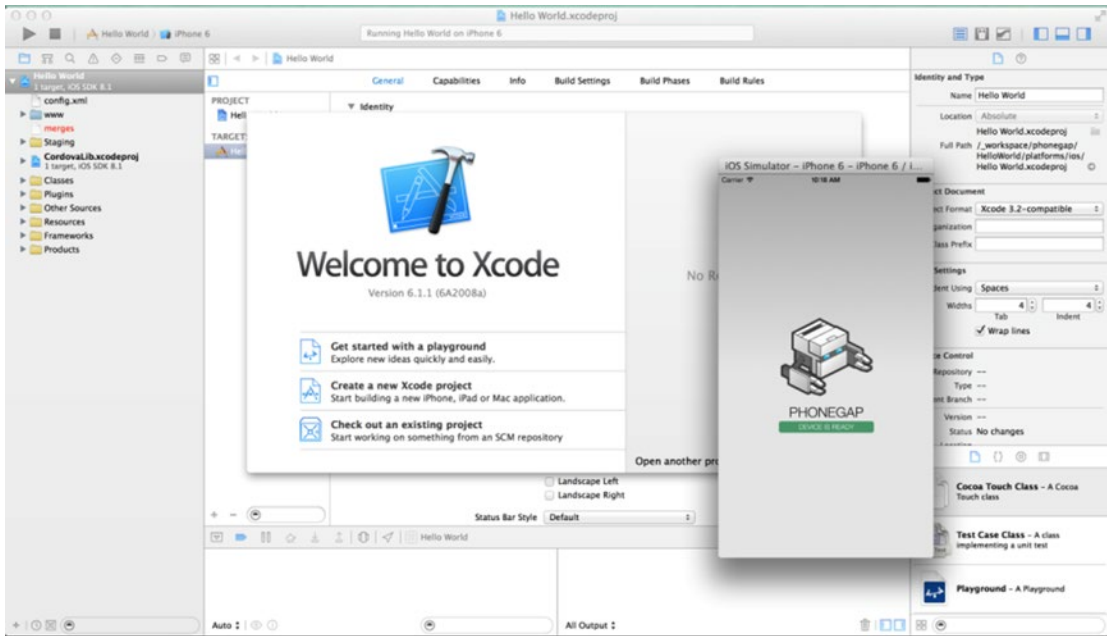


Figure 2-1. *Xcode*, the IDE for iOS, and the iOS simulator

The fastest way to download and install *Xcode* is by searching for “*Xcode*” in the App Store desktop application. After it's installed, it will either be listed in the Mac's dock or in the Launcher. You can also find it using the Spotlight Search.

Alternatively, you may download *Xcode* (and more) in the *Apple Developer Member Center*. The Member Center requires you have an *Apple ID* and register as an Apple Developer. This can be done for free at <https://developer.apple.com/register/>. After you have registered as a developer, visit <https://developer.apple.com/downloads/> to download *Xcode*.

If you intend on releasing your applications to the Apple App Store, you will need to enroll in the *iOS Developer Program* (\$99/year). With that said, you won't need it right now.

■ **Note** Setting up things like Certificates, Identifiers, and Provisioning Profiles (things you'll need to publish your applications) are outside of the scope of this book. For a complete guide on setting this up, visit developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/MaintainingCertificates/MaintainingCertificates.html.

Command Line Tools for Xcode

The Command Line Tools package allows you to do command-line development in OS X. PhoneGap uses this package to build the iOS version of your application. For OS X 10.9 and greater, Xcode comes bundled with all command-line tools.

If you're using an older version of OS X, you can install the Command Line Tools package from the main menu in Xcode. Select *Preferences* and then click the Downloads tab. From the Components panel, click the Install button next to the Command Line Tools listing. If Xcode does not show an option to install the package or if it's missing for any other reason, you can download the package from the Apple Developers downloads page at <https://developer.apple.com/downloads>. Note that in order to view the downloads page, you must be logged in with your Apple Developer credentials.

iOS Launcher Packages

To be able to install and run PhoneGap applications using the iOS simulator or an iOS device from the command line, you need to install two more JavaScript packages using npm. The command `ios-deploy` launches iOS apps to a physical iOS device, and `ios-sim` executes the application using the iOS simulator provided by Xcode. Install the packages via the following commands, adding the `sudo` prefix if needed. The following code lines demonstrate two separate commands for installing the packages used for testing PhoneGap applications.

```
npm install -g ios-sim
npm install -g ios-deploy
```

At this point, you should have everything you need for iOS development with PhoneGap. If you do *not* wish to develop for Android, skip the next section and continue to *PhoneGap Installation*.

■ **Note** You don't have to use Xcode as your code editor. You also don't need to use it to build PhoneGap applications; the latter happens using the command line. However, Xcode is a good tool for testing your application on different types of iOS devices. You can download more iOS simulators using Xcode via Preferences from the main menu (in the Downloads tab).

Android Environment Setup

Setting up your environment for Android development differs greatly from that of iOS. Figure 2-2 illustrates where it all starts—the Android Developer Portal.

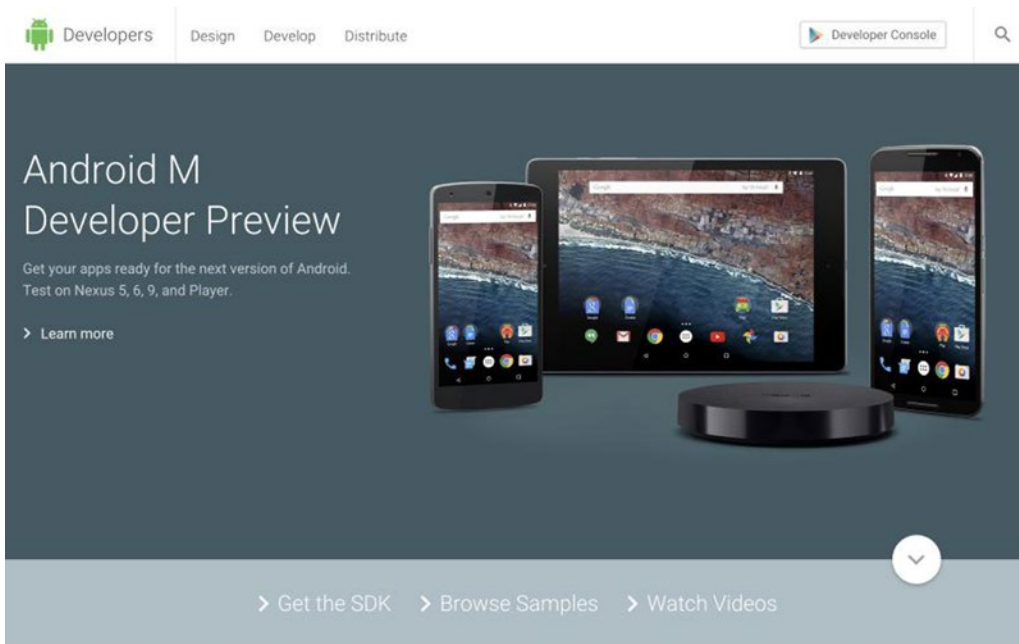


Figure 2-2. The Android Developer Portal at <http://developer.android.com>

To build applications for the Android platform, you need to install the *Android SDK*. As previously mentioned, there are no OS restrictions—you can use either Windows or Mac to build Android applications. The Android environment setup is slightly more involved than the iOS setup. But fear not, you will be guided through the setup process step-by-step.

There are theoretically two ways to install the Android SDK: via the Android Studio software or with the Android standalone SDK. Because Android Studio is not needed to build PhoneGap applications, we don't cover it in this book, but stick to the stand-alone SDK.

Java Development Kit (JDK)

Android is closely tied to the Java programming language. It's no surprise that you'll need to install the JDK to be able to use the Android SDK. This applies in particular to Windows systems; you have to install the JDK *prior* to running Android installations. For older Mac systems (< 10.7), you may skip this step.

Identify the JDK package fitting your system configuration on the Oracle Java Downloads page: www.oracle.com/technetwork/java/javase/downloads/.

Android SDK Installation

As previously stated, we will use the Android SDK Tools only to build the Android version of our PhoneGap applications. You should also be able to navigate to the SDK package download from developer.android.com/sdk/index.html.

In case something has changed since this writing, head over to the Android Developer site (<http://developer.android.com>) and choose Get the SDK link (as shown in Figure 2-2) to navigate to the latest Android SDK download instructions. Next, choose the Installing the Android SDK link from the main navigation. This will take you to a new page where you will click the Stand-alone SDK Tools button.

Pick either Windows or Mac and accept the terms and conditions on the following page to initiate the download. For Windows, we recommend using the executable installer package.

Windows

Double-click the executable (.exe) file to start the installation process. Write down the location where you saved the SDK on your system—you will need it to refer to the SDK directory later when using the SDK tools from the command line. Once the installation completes, the installer starts the *Android SDK Manager*. Skip the following Mac paragraph and continue at Android SDK Manager.

Mac

Unpack the ZIP file you've downloaded. By default, it's unpacked into a directory named `android-sdk-mac_x86`. Move it to an appropriate location on your machine, for example, to the directory `/Users/{YOUR_USERNAME}/Library/Android/sdk/`.

Android SDK Manager

As you may know, there are many different versions of Android. The latest version 5.0 is named *Lollipop*, yet more common are the older releases *KitKat* (4.4) and *Jelly Bean* (4.3, 4.2, 4.1). To install the SDKs for these different platform versions, use the Android SDK Manager. There are some other tools, like the Android SDK Build Tools, that you need to install. These will be installed via the Android SDK Manager as well.

To start the SDK Manager on Windows, double-click the `SDK Manager.exe` file at the root of the Android SDK directory.

On Mac, open a terminal and navigate to the `/tools` directory in the Android SDK directory (for example, `/Users/{YOUR_USERNAME}/Library/Android/sdk/tools`). Then execute the command `android sdk`. This will open a new window, the Android SDK Manager.

Once the Android SDK Manager starts, it will look for packages available for download. This will include both the latest releases as well as older or common packages. See Figure 2-3 for reference. By checking the boxes right next to the packages and hitting the Install button, you can download and install the selected packages.

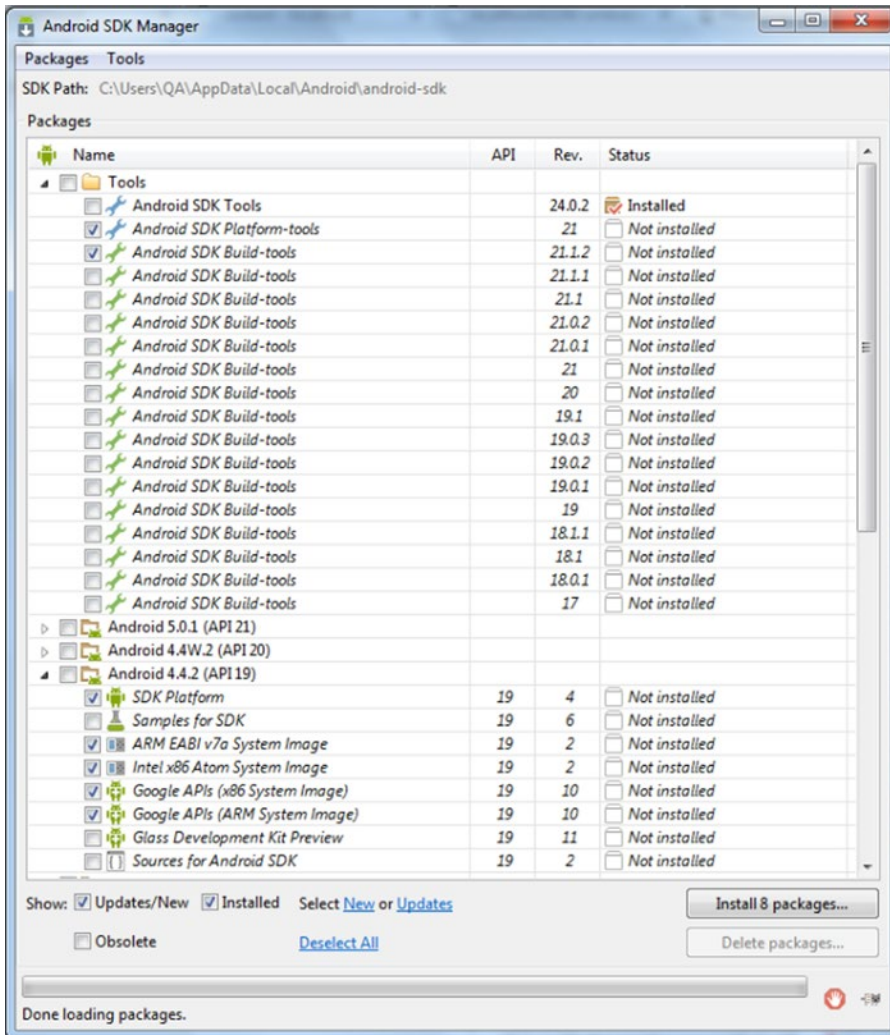


Figure 2-3. Android SDK Manager

At minimum, you should download the latest tools and Android platform:

- Android SDK Tools
- Android SDK Platform-tools
- Android SDK Build-tools (*highest version*)

Next, you need to add each platform version you want to test your application on. We recommend starting on the most common version (as of this writing), *KitKat (4.4)*, first. You can later verify that your application is working on other platforms as well.

Open the folder *Android 4.4.22 (API 19)* and select the following:

- SDK Platform
- A system image for the emulator, such as ARM EABI v7a System Image

Choosing an Android Emulator Image

Emulation on Android is a tricky thing. And by tricky, we mean it's slow. To speed up your emulator, install the package, use the `android sdk` manager, and select "Intel x86 Emulator Accelerator" from the Extras menu. Also select a corresponding emulator image (for example, Intel x86 Atom System Image).

You can find more information about this topic in the article "Speeding Up the Android* Emulator on Intel Architecture" at <https://software.intel.com/en-us/android/articles/speeding-up-the-android-emulator-on-intel-architecture>.

As an alternative, you can use a physical Android device connected via USB cable as a test system. Even though it's an external device, it will speed up the process of installing and testing your application significantly.

There are also third-party services, such as Genymotion (www.genymotion.com), that can assist you in testing your Android applications.

Managing Virtual Devices

Once you have downloaded and installed all Android packages, it's time to add a *virtual device*. A virtual device is an emulator configuration defined by hardware and software options. This means you can combine any emulator image you downloaded in the previous step with a range of devices, operating system versions, and so forth.

Virtual devices are managed via the Android Virtual Device (AVD) Manager. To start the manager, run the command `android avd` in the `tools` directory. Add a new virtual device via the Create button. You will need to configure the following:

- *A hardware profile*: The hardware profile defines the hardware features of the virtual device, including how much memory the device holds, whether it has a camera, and so on.
- *A mapping to a system image*: In the previous step, you downloaded at least one image using the Android SDK Manager. Assign one of the images you downloaded.
- *Other options*: You can specify the screen dimensions, appearance, and other options of the device, including the size of the device's internal storage in which it saves the user's data (installed applications, settings, and so on) as well as a virtual SD card memory.

Once you have finished your configuration, confirm via the OK button, as shown in Figure 2-4.

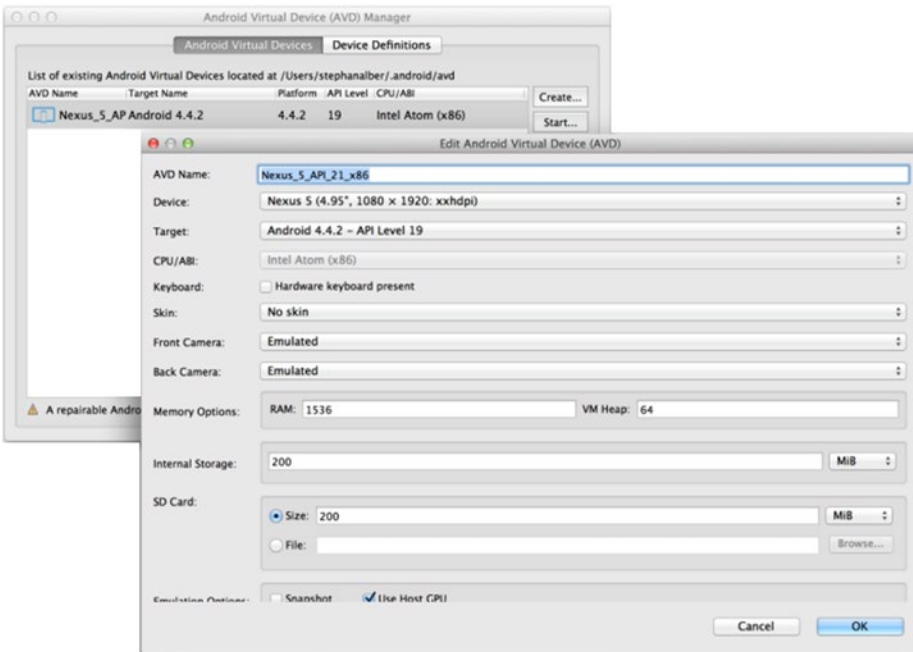


Figure 2-4. The Android Virtual Device Manager (AVD Manager)

You can test your virtual device by selecting it and clicking the Start... button. It may take some time to launch the emulator. If you have major speed issues, please refer back to the preceding section, “Choosing an Android Emulator Image.” Figure 2-5 shows an example of the emulator.



Figure 2-5. Android emulator

Adding Android SDK Paths

To make the Android SDK accessible for PhoneGap, you need to add SDK paths to your system's settings. Among other things, adding these paths will allow PhoneGap to compile your application.

On Windows systems, execute the commands shown in Listing 2-1 in the command prompt. On Mac systems, execute Listing 2-2 in the terminal. Replace the placeholder {INSTALLATION-LOCATION} with the path you picked at the beginning of the Android SDK installation process.

Listing 2-1. Adding Android SDK Paths on Windows

```
set ANDROID_HOME=C:\{INSTALLATION-LOCATION}\android-sdk
set PATH=%PATH%;%ANDROID_HOME%\tools;%ANDROID_HOME%\platform-tools
```

Listing 2-2. Adding Android SDK Paths on Mac

```
export ANDROID_HOME=/\{INSTALLATION-LOCATION}/android-sdk-macosx
export PATH=${PATH}:%ANDROID_HOME%/tools:%ANDROID_HOME%/platform-tools
```

Installing Apache Ant

On most systems, you need to install Apache Ant (<http://ant.apache.org>) as a last step. If you own a Mac and use Homebrew, you can run `brew install ant`.

■ **Note** Homebrew is a package manager for Mac OS X. If you don't yet use it, you should! You can find download and install instructions on <http://brew.sh/>.

You can also pick and download a fitting Ant binary from www.apache.org/dist/ant/binaries/. As Ant is built with Java, the files for Mac and Windows are the same. Extract the ZIP file (for example, `apache-ant-1.9.4-bin.zip`) somewhere on your computer. Add the full path to the contained `bin/` folder to the end of your PATH environment variable as shown in Listings 2-3 and 2-4. You can find more instructions and help how to install Apache Ant on ant.apache.org/manual/install.html.

Listing 2-3. Adding the Apache Ant Path on Windows

```
set PATH=%PATH%;C:\{YOUR-PATH}\apache-ant-1.9.4\bin
```

Listing 2-4. Adding the Apache Ant Path on Mac

```
export PATH=${PATH}:{YOUR-PATH}\apache-ant-1.9.4\bin
```

Installing PhoneGap

Next, you'll be installing PhoneGap via the node package manager. The PhoneGap command-line tool will allow you to install plug-ins, build your application, or install and run your application using a simulator.

Windows

To install PhoneGap on Windows, open the command prompt. You can find the command prompt via the Windows search using “cmd” as your search term or via the Start button in *All Programs* ► *Accessories* ► *Command Prompt*. Execute `npm install -g phonegap`. It does not matter in which directory you execute this command.

Mac

Open the Terminal application. You can find it using Spotlight Search. As on Windows, execute `npm install -g phonegap`. If you run into a permission error, you may need to run this as a *superuser* (`sudo`), since the installation requires authorization. In this case, execute `sudo npm install -g phonegap` instead.

■ **Tip** The `-g` flag installs npm packages globally, making the command available to run in any directory on your computer. This is why the location of where you execute this command is irrelevant. Without using `-g`, the package will only install in the directory that `npm install` is executed.

During the installation process, you will see several requests to npm logged in the terminal; this retrieves all the packages used for PhoneGap. To ensure it's properly installed, run `phonegap -v` once. This will return the version of PhoneGap you have installed.

Updating PhoneGap

When installing PhoneGap the first time, you'll always have the latest version. You can stay up-to-date with npm packages by using the `update` command before the name of the package, for example, `npm update -g phonegap` or respectively `sudo npm update -g phonegap`. Follow the PhoneGap blog to stay up-to-date with new releases and information at <http://phonegap.com/blog/>.

Using the PhoneGap CLI Tools

You'll be using the CLI throughout this book to run an array of commands such as compiling PhoneGap applications and running your application using emulation software. To get an idea of what is in store when using `phonegap` commands, in your terminal execute `phonegap help`. This will show all the commands available in the PhoneGap CLI, as shown in Figure 2-6.

```

Usage: phonegap [options] [commands]

Description:

  PhoneGap command-line tool.

Commands:

  help [command]      output usage information
  create <path>       create a phonegap project
  build <platforms>   build the project for a specific platform
  install <platforms> install the project on for a specific platform
  run <platforms>    build and install the project for a specific platform
  platform [command] update a platform version
  plugin [command]   add, remove, and list plugins
  info               display information about the project
  serve              serve a phonegap project
  version            output version number

Additional Commands:

  local [command]     development on local system
  remote [command]   development in cloud with phonegap/build
  prepare <platforms> copies www/ into platform project before compiling
  compile <platforms> compiles platform project without preparing it
  emulate <platforms> runs the project with the flag --emulator
  cordova             execute of any cordova command

Experimental Commands:

  Requires the --experimental flag to use the command

  save               save installed platforms and plugins
  restore            restores saved platforms and plugins

Options:

  -d, --verbose     allow verbose output
  -v, --version     output version number
  -h, --help        output usage information

Examples:

  $ phonegap help create
  $ phonegap create path/to/my-app
  $ cd my-app/
  $ phonegap run ios

```

Figure 2-6. PhoneGap help menu used in the command-line interface

For even more information from the command line, run `phonegap help <command>`. For example, `phonegap help create` provides a full description of how to use the create command.

Creating a New PhoneGap Application

It's time to create your first PhoneGap application. All new PhoneGap projects start by using the create keyword. Before you get started, select a location on your computer where you intend on saving projects associated with this book. You can develop from any directory just as long as you keep it consistent and easy find. For example:

Windows

```
C:\Users\%USERNAME%\Appres\Chapter-2\
```

Mac

```
/Users/<username>/Documents/Apress/Chapter-2/
```

When you have a location that you're happy with, navigate to that folder in your terminal and execute `phonegap create <project name>` to create a new PhoneGap application. You can see an example for creating an application named `HelloWorld` in Listing 2-5.

Listing 2-5. Terminal Command for Creating a New PhoneGap Application Named “HelloWorld”

```
phonegap create HelloWorld
```

Executing the code in Listing 2-5 will create a new folder of the same name, for example, `C:\Users\%USERNAME%\Apress\Chapter-2\HelloWorld`. The contents inside are files and folders PhoneGap uses to build and compile the application. You'll need to navigate into this folder through the terminal to be able to run `phonegap` commands. You can do so by executing the change directory command followed by the folder name: `cd HelloWorld` (Windows and Mac).

PhoneGap “Hello World” Application

The `create` command will add a small sample application to your project folder. You can find the application files in the directory `HelloWorld/www`. Further details about files and folders inside the “Hello World” application will be described later. For the moment, the goal is to run this application on an Android and/or iOS simulator.

After creating a PhoneGap application, it's time to add a testing platform. To do so, run the command `phonegap platform add <platform>`. To add iOS and Android to your project, run the commands shown in Listing 2-6.

Listing 2-6. Adding Platforms to Existing PhoneGap Projects (iOS, Android)

```
phonegap platform add ios
phonegap platform add android
```

The command will create a directory for each platform in `./HelloWorld/platforms`. These directories will contain platform specific code and libraries as well as your JavaScript application, HTML, CSS, and image files.

Important rule for the `platforms` directory in advance: You should *never* change the contained files, or bad things may happen.

You can get a list of all installed and available platforms using `phonegap platform list`. Other `phonegap platform` commands include `update` and `remove` respectively. For more information, run `phonegap platform help`.

Building Applications

As explained in the introductory chapter, PhoneGap will compile your web application into a program that can run natively on a device. Executing `phonegap build <platform>` performs this operation. You will need to use the `build` command every time you change your web application code or when you add or remove a PhoneGap plug-in.

The first time you add a platform to a PhoneGap project, the application is ready to run. However, for testing purposes, use the `build` command for all platforms you added in the previous section, as shown in Listing 2-7.

Listing 2-7. Build the Application for the Targeted Operating System

```
phonegap build ios
phonegap build android
```

Running Applications

There are generally two different ways to run a PhoneGap application: using a simulator or a physical device. Because running your application on a device will require some additional work, let's focus on using a simulator for now. To install and run your application using a simulator, use the `emulate` command, as shown in Listing 2-8.

Listing 2-8. Start the Emulator from the Command Line Using the Targeted Device Name

```
phonegap emulate ios
phonegap emulate android
```

After a few moments, your system will do a few things:

1. Launch the simulator for the targeted platform
2. Compile and install the application on the simulator
3. Start the application

The application will show the PhoneGap icon followed by the message *Connecting to Device*. Once the app is fully loaded, the message will switch to *Device is Ready*. Figure 2-7 shows that the device is ready.