
MOBILE
PROGRAMMING
SERIES
APACHE
CORDOVA 3
PROGRAMMING
COVERS
PHONEGAP
3.X
JOHN M. WARGO
FREE SAMPLE CHAPTER
SHARE WITH OTHERS

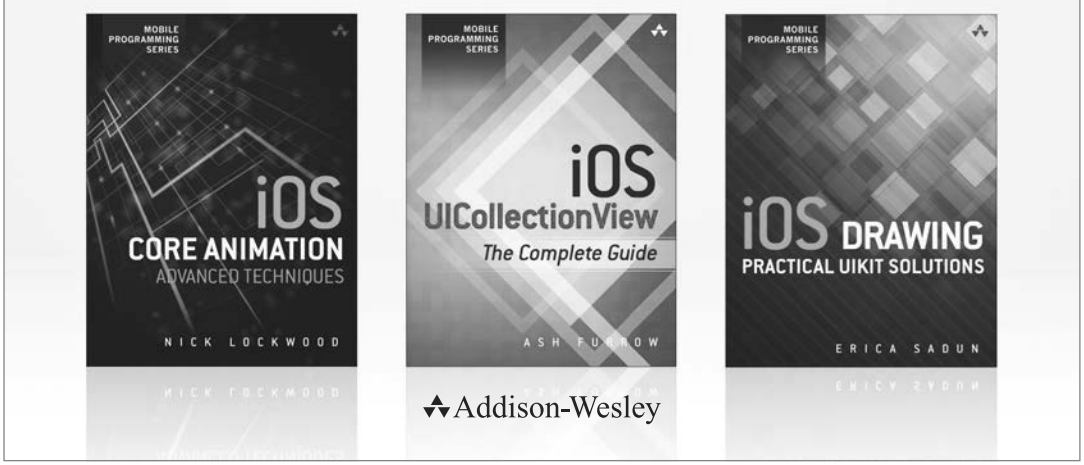
Apache Cordova 3 Programming

John M. Wargo

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Addison-Wesley Mobile Programming Series



Visit **informit.com/mobile** for a complete list of available publications.

The **Addison-Wesley Mobile Programming Series** is a collection of digital-only programming guides that explore key mobile programming features and topics in-depth. The sample code in each title is downloadable and can be used in your own projects. Each topic is covered in as much detail as possible with plenty of visual examples, tips, and step-by-step instructions. When you complete one of these titles, you'll have all the information and code you will need to build that feature into your own mobile application.

Make sure to connect with us!
informit.com/socialconnect

informIT.com
the trusted technology learning source | Addison-Wesley | Safari Books Online

ISBN-13: 978-0-321-95736-8
ISBN-10: 0-321-95736-9

First released, December 2013

To my wife, Anna.

This work exists because of your outstanding support.

To my children,

who were relatively patient as I worked on yet another book.

Contents

Foreword

It has been roughly twenty-five years now since the mobile technology era began. Can it be just a coincidence that it is almost exactly the same amount of time that I have known John Wargo, the author of this book? In 1988, the mobile technology landscape consisted of mobile phones the size of a carry-on suitcase and personal organizers that looked like glorified financial calculators. Wireless networks and widespread access to resources like the Internet were distant dreams. Unless you were a science fiction writer, it would have been hard to imagine the connected world that we take for granted today. If you knew John Wargo as I do, though, it would not have been very far-fetched at all to predict that he would turn out to be the author of four books.

Looking ahead another twenty-five years, I imagine it's safe to say that our ability to accurately predict the advance of technology will continue to fall short. Powerful development tools like Cordova and the combined creativity of millions of application developers around the world virtually guarantee that we won't be able to guess what indispensable capabilities will appear on our mobile devices tomorrow or the next day. Will tomorrow's devices interface directly to our human nervous systems? Will they assemble themselves from organic compounds and heal themselves if they become damaged? Who knows.

Whatever the future of mobile technology holds for us, as long as there is a means to program or control it in some way, I hope that John will put together a collection of words to point others in the right direction.

—David M. Via
AT&T

Preface

This is a book about programming cross-platform mobile applications using Apache Cordova 3 (with some coverage of PhoneGap 3 as well). In Apache Cordova 3, the Cordova development team made some dramatic changes in the framework, and this book is what you need to understand what Cordova 3 is all about. The book can be considered as a sequel to my *PhoneGap Essentials*, updated for Cordova 3.

This is a book about Cordova 3. This book is targeted at mobile developers who want to learn about Cordova 3. If you're brand new to Cordova, then this book is just what you need to get started. If you're experienced with an older version of Cordova, this book will show you in detail how to use all of the new stuff that's in Cordova 3. You will, however, need to have at least some experience with mobile development to benefit from this book. The target audience could be existing web developers who want to get into mobile development, but much of the needed native mobile development background just isn't in here.

What you'll find in the book:

- Lots of detailed information about Apache Cordova: what it does and how it works
- Lots of examples and code

What you won't find in this book:

- Mobile web development and mobile development topics; this book is about Apache Cordova, not mobile development
- Expressions or phrases in languages other than English (I hate it when authors include phrases in Latin or French)
- Obscure references to pop-culture topics (although there is an overt reference to Douglas Adams's *Hitchhiker's Guide to the Galaxy* and one obscure reference to "Monty Python")
- Pictures of my children or my pets

This book is not a book for experienced Cordova 3 developers—if you consider yourself an experienced Cordova 3 developer, then you probably should not buy this book.

Herein I tried to provide complete coverage of Cordova 3; covering enough detail that readers will leave with a complete understanding of what Cordova is, what it does, how it works, and how to use it for their mobile application projects. There's a whole lot more to Cordova; many advance topics and more detailed coverage of the Cordova APIs can be found in the Cordova documentation and maybe in some future book I'll write.

Cordova as a Moving Target

One of the challenges in writing a book about open source projects is that if the project is well staffed and busy, the project gets regular updates. In Cordova's case, it's one of the fastest moving open source projects on the planet, so with their monthly updates and yearly major release, it is definitely a moving target.

I've worked very hard to structure and craft this book so that it can survive the rapid pace of the project, but only time will tell. You may find that something I've written here has changed and the book doesn't align with reality. There's nothing I can do about this except to stay on top of it and post updates to the book's website (described shortly) when I find that something has changed enough that it breaks part of the book.

A Comment on Source Code

One of the things you'll notice as you look at the source code included in the book is that I've paid special attention to the formatting of the code so that it can be easily read and understood. Rather than allowing the source code to wrap wherever necessary to fit the printed page, I've forced line breaks in the code in order to structure it in a way that should benefit the reader. Because of this, as you copy the source code over into your Cordova applications, you will likely find some extra line breaks that affect the functionality of the code. Sorry.

Resources

I've created a website for the book at www.cordovaprogramming.com, shown in Figure P1, where I will post updates, errata, and the answers to questions I've received from readers.

Figure P.1 Apache Cordova 3 Programming Web Site

Additionally, I've posted much of the book's sample code to my GitHub account located at https://github.com/johnwargo. At a minimum, I'll post all of the complete applications up there—potentially adding code snippets if readers ask for them.

Please feel free to use the contact form on the book's website to provide feedback and/or suggestions for the next release.

Acknowledgments

I want to thank the following people for their help with this effort:

- The Cordova development team for answering every one of my silly questions as I prepared the manuscript
- Brian LeRoux from Adobe for his support
- Fil Maj and Hardeep Shoker from Adobe for reviewing and providing feedback on the Cordova CLI and PhoneGap Build chapters
- Brent Thornton, Bryan Higgins, Ken Wallis, and Jeffrey Heifetz from BlackBerry for helping me with the BlackBerry content
- Raman Sethi and the SAP Kapsel development team for teaching me all sorts of stuff I didn't know about Cordova
- My colleague Marcus Pridham for pointing out what was wrong with my Android plugin code and showing me how to create the iOS version of the plugin
- My colleague Istvan Nagy for showing me a better way to tell what's going on with a Cordova application when it fails (described in Chapter 14)
- Colleagues Damien Murphy, Scott Dillon, and Andrew Lunde for their review of the manuscript
- My managers and other colleagues at SAP for supporting me throughout this process
- Greg Doench and the staff at Addison-Wesley/Pearson Education for helping me write yet another book; may this one not be my last.

6

The Mechanics of Cordova Development

Each of the mobile platforms supported by Cordova has a process and tools you can use to test and, in the unlikely event your code has bugs, debug Cordova applications. In general, you can load a Cordova application into a device simulator or emulator, provided as part of the mobile platform's SDK, or you can load an application onto a physical device. There are also third-party solutions you can use to test your Cordova applications within a desktop browser interface.

Some processes and capabilities apply across all supported mobile device platforms. In this chapter, I address the mechanics of Apache Cordova development. I begin the chapter by addressing some of the issues a Cordova developer must deal with, then cover the development process and some of the tools you can use to test and debug your Cordova applications.

Cordova Development Issues

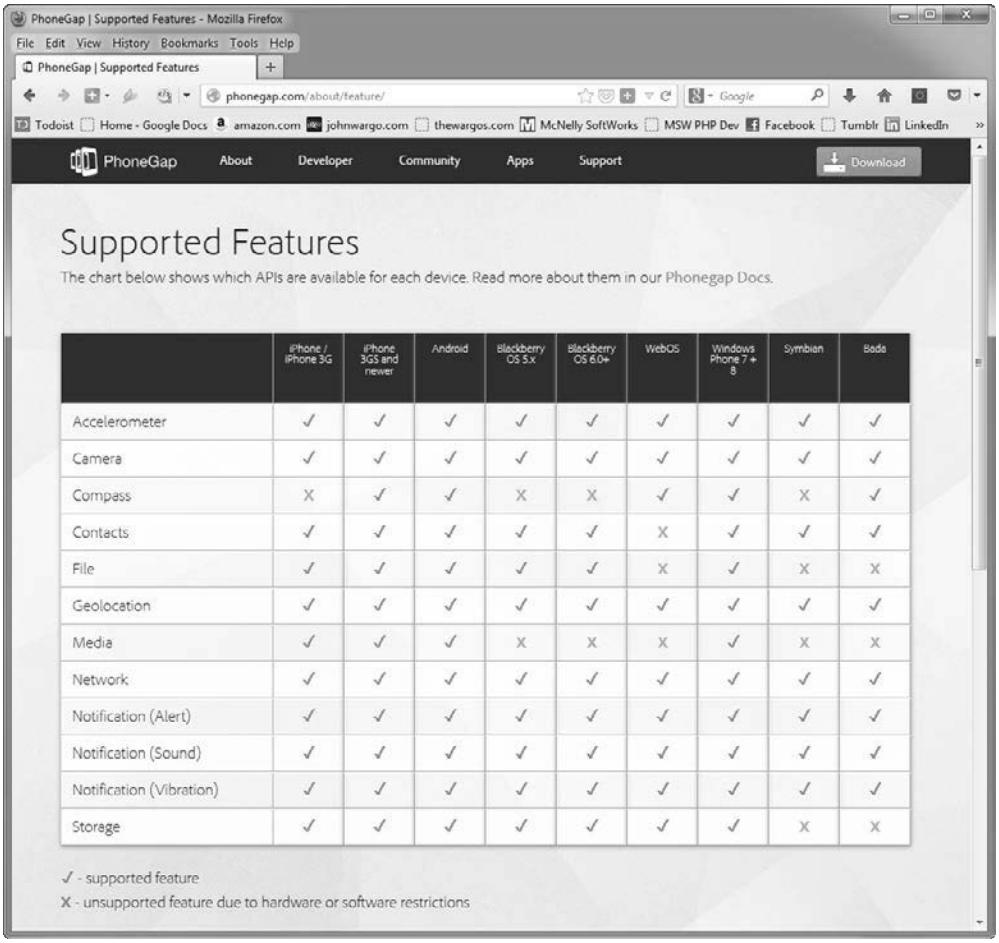
Before we start discussing how to develop Cordova applications, let's address some of the issues that you will face as you work with the framework. The Cordova project is supported by developers from all over the world, developers who may have experience with only one or a small number of mobile platforms, developers who have a strong opinion about how something should be done. The problem is that when you take development projects written by different people and try to collect them into a single framework, you will likely bump up against some inconsistencies. Add the fact that every mobile platform supported by Cordova is different and has different ways of doing things, and you have a difficult task to make everything work cleanly and seamlessly.

In the predecessor to this book, I used this section of the chapter to highlight all of the issues I'd encountered while learning (at the time) PhoneGap and later writing the book. Back then, there were a bunch of issues, and they created some interesting problems for developers. The good news is that over time, the Cordova development team has done an amazing job in eliminating most of them. All that's left are two, and they're not that complicated.

Dealing with API Inconsistency

Figure 6.1 shows the supported feature matrix from the PhoneGap website (the Cordova team doesn't seem to publish a matrix); you can find the page at http://phonegap.com/about/feature/. As you can see, the table is pretty complete; there are some gaps, but it's more full than empty. If a particular feature you want to use in your application is supported only on some mobile platforms, then you'll

have to make special accommodation within your application for platforms that do not support the particular API.



	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 5.x	Blackberry OS 6.0+	WebOS	Windows Phone 7 + 8	Symbian	Bada
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓
Compass	X	✓	✓	X	X	✓	✓	X	✓
Contacts	✓	✓	✓	✓	✓	X	✓	✓	✓
File	✓	✓	✓	✓	✓	X	✓	X	X
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	X	X	X	✓	X	X
Network	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	X	X

Figure 6.1 Cordova-Supported Feature Matrix

Caution

Keep in mind that the table is not updated as often as the API is, so you may want to validate through the API documentation or through actual testing of an API whether or not it works on a platform where there's an *X* in Figure 6.1.

If your application uses an API that isn't supported on all of the mobile devices that your application will target, then your application's code can use the Device API discussed in Chapter 5. Your application should use `device.platform` and, as necessary, `device.version` to determine which platform and OS the application is running on and disable any unsupported feature if the application is running on a device that doesn't support the API. Another option is to simply wrap the call to a particular API with a JavaScript try/catch block and deal directly with any failures that occur.

Application Graphics

Each mobile platform and, often, different versions of a particular device OS have different requirements for application icons and splash screens. Developers building Cordova applications for multiple device platforms must be prepared to create a suite of graphics for their application that addresses the specific requirements for each target device platform and/or device OS. For application icons, the PhoneGap project maintains a wiki page listing the icon requirements for the different supported operating systems here: https://github.com/phonegap/phonegap/wiki/App-Icon-Sizes.

Additionally, for some devices on some carriers (older BlackBerry devices, for example), mobile carriers apply a specific theme to the OS to help distinguish themselves in the market. Any application icon designed for one of these devices will need to accommodate, as best as possible, rendering pleasantly within different themes. Fortunately, with the merges capabilities described later in this chapter, you have the ability to easily merge the appropriate graphics files (and other content as needed) into your project depending on which mobile platform you are building for.

Developing Cordova Applications

Now it's time to start working through the process of how to create Cordova applications. In this section I describe the process for coding a Cordova application. In later sections, I show you how to test and debug applications.

Working with a Single Mobile Device Platform

It's possible that some developers will work with only a single mobile platform. If you are such a developer, all you have to do is open up a terminal window and issue the following commands (which are described in Chapter 4, "Using the Cordova Command-Line Interface"):

```
cordova create app_name
cd app_name
cordova platform add platform_name
cordova prepare platform_name
```

> **Warning**
>
> This isn't necessarily the right way to do single-platform development, as I'll describe later—I'm just trying to describe a potential process here.

In this example, `app_name` refers to the name of the application you are creating and `platform_name` refers to the mobile device platform you will be working with. So, if you were creating a BlackBerry application called lunch_menu, you would issue the following commands:

```
cordova create lunch_menu
cd lunch_menu
cordova platform add blackberry
cordova prepare blackberry
```

You can also specify more information about your application by using the following:

```
cordova create lunch_menu com.cordovaprogramming.lunchmenu "Lunch Menu"
cd lunch_menu
```

```
cordova platform add blackberry
cordova prepare blackberry
```

At this point, the command-line interface (CLI) would create the Cordova project folder shown in Figure 6.2, and all you need to do at this point is open your code editor of choice and start coding and testing your new Cordova application.

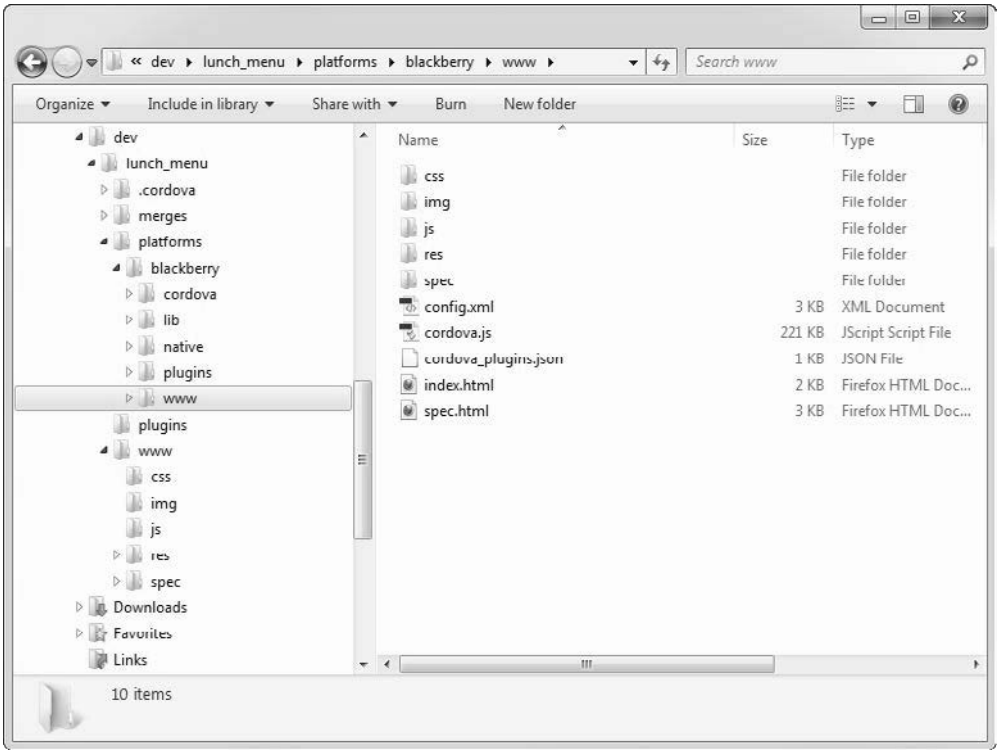


Figure 6.2 Cordova Application Project Folder Structure: BlackBerry Application

The BlackBerry platform project folder contains a copy of the web application files you need to work with.

> **Note**
>
> If you later decide to add additional mobile device platforms to your project, you need to manually copy the application's web content files from the BlackBerry project's www folder, highlighted in Figure 6.2, over to the www folder within the overall Cordova (not BlackBerry) project shown in the figure. The content that follows describes how multiplatform Cordova projects differ and why this is important.

Working with Multiple Mobile Device Platforms

Because Cordova is all about cross-platform mobile development, you're probably going to want to target multiple mobile device platforms. In that case, if you were building an app for Android and iOS, for example, you would open a terminal window and do something like the following:

```
cordova create lunch_menu
cd lunch_menu
cordova platform add android ios
```

At this point, you'd have a new Cordova project structure with projects for both Android and iOS, as shown in Figure 6.3. As discussed in Chapter 4, there's a separate folder called www contains the application's core web content files, the content files that are shared across both the Android and iOS projects.

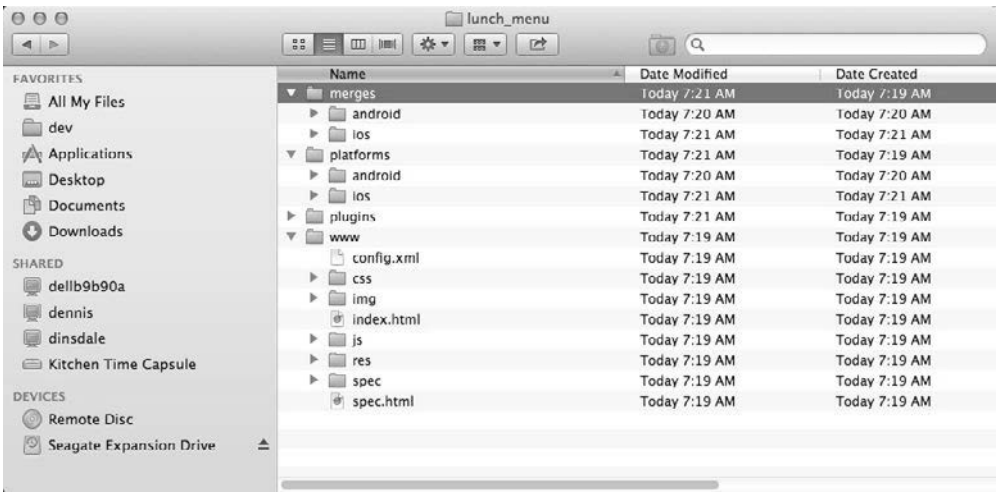


Figure 6.3 Cordova Application Project Folder Structure

In this scenario, you will work with the web content stored in the www folder, shown at the bottom of the folder structure in Figure 6.3. When you have the web application content in that folder ready for testing, you use the CLI to copy the code into the platforms sub-folders (android and ios), shown in the figure.

What I do while working on a Cordova project is keep my web content files open in an HTML editor such as Adobe Brackets (www.brackets.io) or Aptana Studio (www.aptana.com), then use the CLI to manage my mobile device platform projects for me. As I edit the files, I add the web content to the .html file and my application's code to the application's .js files. When I'm ready to test (and debug) the applications, I switch over to a terminal window that I keep open and pointed to the Cordova project's root folder (the lunch_menu folder I created a while back) and issue some commands. If I want to switch to the Android IDE and test the Android application, I issue the following command:

```
cordova prepare android
```

Or, if I will be testing and debugging both the Android and iOS versions of the application, I issue the following command:

```
cordova prepare android ios
```

What this command does is copy all of the project files from the www folder into the appropriate folder for each specified mobile platform project, as shown in Figure 6.4. In this example, it copies the content files to the Android project's assets/www folder and the iOS project's www folder. The contents of the config.xml file should be applied to the platform-specific config.xml file located in the target directory.

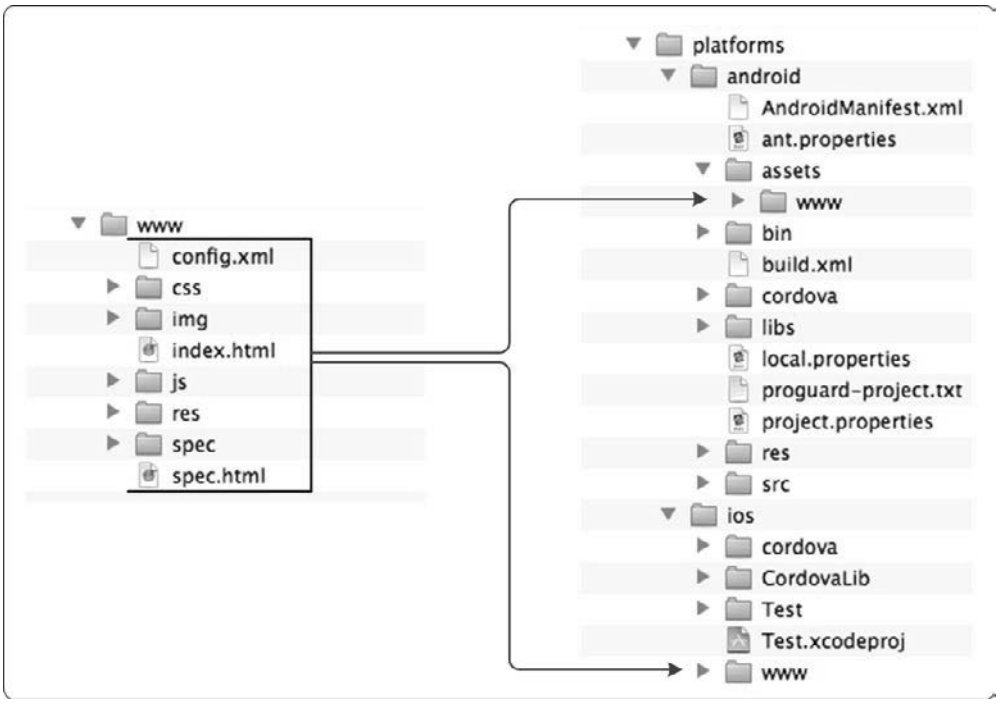


Figure 6.4 Copying Web Content to the Platform Projects Folders

Now, any self-respecting mobile web project may have some icons, screen graphics, CSS, and/or JavaScript files that are unique to each target platform. Since each mobile device has its own theme and icon requirements, it's likely that at a minimum of those will be required. In older versions of Cordova, you had to manage all of that manually; with the CLI, that's all taken care of for you.

Notice the merges folder shown in Figure 6.3; Cordova uses that folder structure to provide you with a place to store the web application resources that are unique to each target platform. When you issue the Cordova `prepare` commands shown earlier, the CLI copies the custom content for each of the platforms into the appropriate web content folder for each platform's project folder, as shown in Figure 6.5.

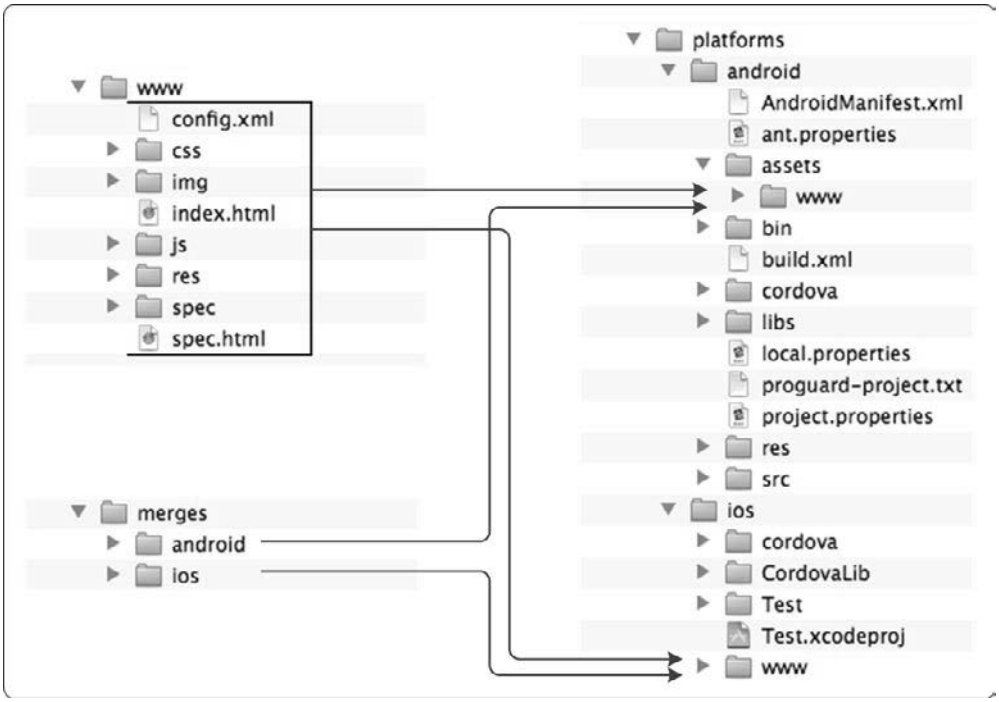


Figure 6.5 Copying Web Content and Platform-Specific Content to the Platform Projects Folders

As shown in the figure, custom content for the Android platform stored in the merges\android folder is copied into the Android platform project's assets\www folder. Custom content for iOS applications is copied from merges\ios to the iOS project's www folder.

With all of the application's content copied into the appropriate project folders, you open the appropriate IDE (Eclipse for Android and Xcode for iOS) and begin the testing process. For information on how to import the Cordova projects to each IDE and use the platform's debugging tools, refer to Chapters 7 through 10.

Testing Cordova Applications

You can also skip the IDEs entirely and test the applications directly from the command line; I show you how in the following sections.

Run a Cordova Application on a Device Simulator

Most mobile device manufacturers provide a software program that emulates or simulates a mobile device. This allows developers to easily test their mobile applications when they don't have a physical device. Performance isn't usually the same, but it looks and acts like a real device much as it can. In some cases, what's provided is generic and simply mimics the capabilities of the specific OS version, while for other mobile platforms it might mimic specific devices. Either way, there's a software-only solution available that developers can use to test Cordova applications in an almost

real-world scenario (I'll explain "almost real-world" in the following section). Google, for example, provides Android emulators, and Apple and BlackBerry provide simulators of their devices.

> **Simulator vs. Emulator**
>
> There is a technical difference between the two that I'm not going to get into here. In order to make things simpler for me (and you), I'm going to dispense with calling out whether I'm referring to an emulator or simulator for the remainder of the book and simply refer to either as simulators. If you see that word going forward, know that I mean either *emulator* or *simulator*.

To run a Cordova application using a device simulator, you would use the following command:

```
cordova emulate device_platform
```

Replace the value for `device_platform` with the name of the mobile device platform you wish to emulate (android, blackberry10, ios, wp8, and so on). For example, to run the application on a BlackBerry 10 simulator, you would issue the following command:

```
cordova emulate blackberry10
```

In this example, the CLI will prepare the files, build the application using the platform's command-line tools, then launch the appropriate simulator and run the application. You saw examples of the device simulators and emulators in the screenshots found in Chapter 5, "Anatomy of a Cordova Application."

Run a Cordova Application on a Physical Device

Before you deploy your application to mobile users, you should perform final testing on a physical device. As good as these options are, there is always something that doesn't work quite right on a simulator. To test an application on a physical device, connect the device to your development system using a USB cable, then issue the following command:

```
cordova run device_platform
```

For example, to run the application on an Android device, issue the following command:

```
cordova run android
```

Behind the scenes, the CLI will execute the `prepare` command described earlier, then call the particular platform's command-line tools to package the application and deploy it to the device that is connected to the system. Within seconds (or as much as a few minutes in the case of some platforms), the application will appear on the device's screen.

> **Warning**
>
> For many mobile device platforms, applications will not run on physical devices without first being registered with the manufacturer (Windows Phone 8) or signed by an appropriate signing authority (BlackBerry 10, iOS). I'm deliberately omitting the details of this process from this chapter, as it differs across the different supported mobile device platforms and would add some bulk to this manuscript. I cover this topic a little bit in the chapters that deal with each mobile device platform separately (Chapters 7 through 10).
>
> Before testing Cordova applications on a physical device, make sure you have followed the manufacturer's instructions for configuring the appropriate environment to do so.

Leveraging Cordova Debugging Capabilities

As you test your Cordova applications, you're likely to run into issues that you must resolve. The purpose of this section is to highlight some of the debugging capabilities that are available to you outside of an IDE.

Using `Alert()`

One of the simplest, and most annoying, ways to debug a Cordova application is to use the JavaScript `alert()` function to let you know what part of the code you're running or to quickly display the contents of a variable. I've always called this approach the "poor man's debugger," but it works quite well for certain types of application debugging tasks. If you see an event that's not firing within your application or some variable that's not being set or read correctly, you can simply insert an `alert()` that displays a relevant message and use that to see what's going on.

As I started working with PhoneGap and PhoneGap Build, I noticed that there were many times when the `deviceready` event wasn't firing in my applications. I would write my application and start testing it only to find that none of the PhoneGap APIs were working. In some cases, it was because the PhoneGap Build service wasn't packaging the phonegap.js file with the application (that's what happens when you use a beta product). In other cases, it was simply because I had some stupid typo in the application that I couldn't see.

> **Warning**
>
> Cordova fails silently when it encounters a JavaScript error, so if you have a typo in your code, the code will simply not run.

What I started doing in my then PhoneGap, now Cordova, applications was to add a call to `alert()` in the `onBodyLoad` and `onDeviceReady` functions of all of my applications during development. In Chapter 5, I provided a listing for the HelloWorld3 application, but the real HelloWorld3 application code is shown in Listing 6.1. In this version of the application, you can see the calls to alert in the `onBodyLoad` and `onDeviceReady`. Once I was certain that the application worked correctly, I would remove the alerts.

Listing 6.1 The "Real" HelloWorld3 application

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-type" content="text/html;
      charset=utf-8">
    <meta name="viewport" content="user-scalable=no,
      initial-scale=1, maximum-scale=1, minimum-scale=1,
      width=device-width;" />
    <script type="text/javascript" charset="utf-8"
      src="Cordova.js"></script>
    <script type="text/javascript" charset="utf-8">
      function onBodyLoad() {
        alert("onBodyLoad!");
        document.addEventListener("deviceready", onDeviceReady,
```

```
        false);
      }
      function onDeviceReady() {
        alert("onDeviceReady!");
        br = "<br />";
        //Get the appInfo DOM element
        var element = document.getElementById("appInfo");
        //Replace it with specific information about the device
        //running the application
        element.innerHTML = 'Cordova Version: ' +
          device.cordova + br +
          'Platform: ' + device.platform + br +
         'Model: ' + device.model + br +
         'OS Version: ' + device.version;
       }
    </script>

  </head>
  <body onload="onBodyLoad()">
    <h1>HelloWorld3</h1>
    <p>This is a Cordova application that makes calls to the
    Cordova APIs.</p>
    <p id="appInfo">Waiting for Cordova Initialization to
    complete</p>
  </body>
</html>
```

When I was writing all of the sample applications for *PhoneGap Essentials*, I even go as far as to put an alert at the beginning of every function in the application. As I learned how and when each event fired, I used the alerts to help me tell what was going on. Now, there are easier ways to do that, which I show you in the next section, but this was just a simple approach to help me as I got started with each API.

> **Warning**
>
> Those of you who know a little bit about the Cordova APIs might be asking, Why did he use `alert()` rather than the Cordova `navigator.notification.alert()` function?
>
> Well, in the `onBodyLoad()` function, it is highly likely that cordova.js hasn't loaded yet, so I can't be sure that the Cordova `navigator.notification.alert()` will even be available. I could have used `navigator.notification.alert()` in the `onDeviceReady()` function because the only time that function runs is when the Cordova `deviceready` event has fired, but for some reason I just kept the two alerts consistent.

Writing to the Console

The problem with using the approach described in the previous section is that when you fill your buggy code with alerts, you're constantly interrupting the application flow to dismiss the alerts as they come up. For a simple problem, this approach works pretty well, but when debugging more troublesome errors, you need an approach that allows you to let the application run then analyze what

is happening in real time or after the application or a process within the application has completed, without interrupting the application. Cordova applications can do this through the JavaScript `console` object implemented by the WebKit browser-rendering engine.

Using the console object, developers can write messages to the browser's console that can be viewed outside of the running program through capabilities provided by the native SDKs or device simulators. The `console` object has scope at the window level, so it's essentially a global object accessible by any JavaScript code within the application. WebKit supports several options; the most common ones used are listed here:

- `console.log("message");`
- `console.warn("message");`
- `console.error("message");`

Beginning with Cordova 3.0, the console has been removed from the core Cordova APIs and is instead available as a plugin. To add console capabilities to your Cordova project, you must open a terminal window, navigate to the project folder, and issue the following command:

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

Now, let's take a look at a sample application that illustrates the use of this feature, as shown in Listing 6.2

Listing 6.2 Example Application That Writes to the Console

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width,
      height=device-height, initial-scale=1.0,
      maximum-scale=1.0, user-scalable=no;" />
    <meta http-equiv="Content-type" content="text/html;
      charset=utf-8">
    <script type="text/javascript" charset="utf-8"
      src="cordova.js"></script>

    <script type="text/javascript" charset="utf-8">

      function onBodyLoad() {
        document.addEventListener("deviceready", onDeviceReady,
          false);
      }
      function onDeviceReady() {
        //Just writing some console messages
        console.warn("This is a warning message!");
        console.log("This is a log message!");
        console.error("And this is an error message!");
      }

    </script>
  </head>
  <body onload="onBodyLoad()">
```

```
    <h1>Debug Example</h1>
    <p>Look at the console to see the messages the application
    has outputted</p>
  </body>
</html>
```

As you can see from the code, all the application has to do is call the appropriate method and pass in the text of the message that is supposed to be written to the console.

Figure 6.6 shows the messages highlighted in the Xcode console window. This window is accessible while the program is running on an iOS simulator, so you can debug applications in real time.

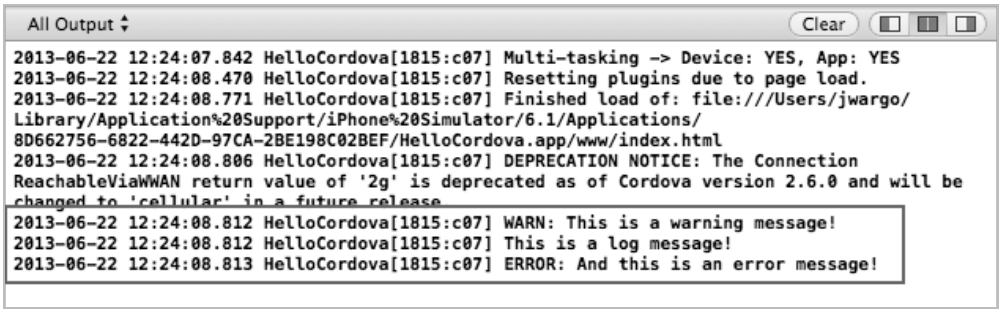


Figure 6.6 Cordova iOS Application Output Log in Xcode

On some platforms, the console will display `log`, `warning`, or `error` messages differently, making it easier for developers to identify a warning versus an error message. To illustrate this, Figure 6.7 shows the contents of the Android LogCat (described in Chapter 7, "Android Development with Cordova"). Notice that the different console message types are color coded, making it easier for you to spot a particular type of message.

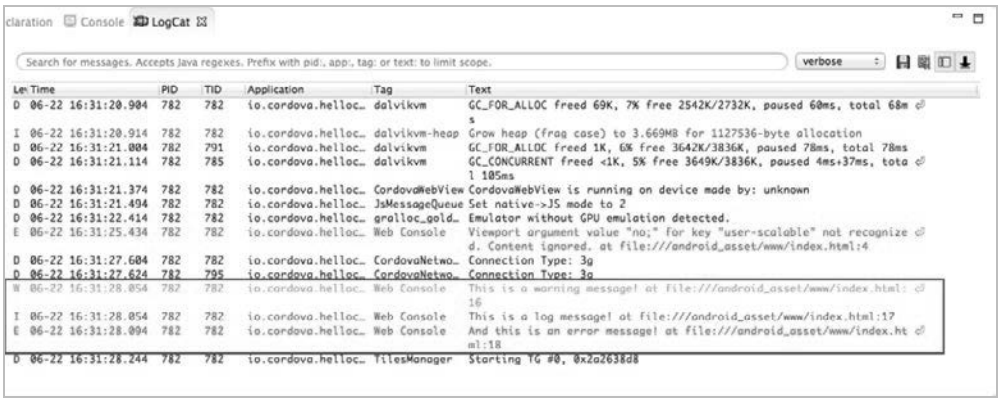


Figure 6.7 Cordova Android Application LogCat Output in Eclipse

Remember I mentioned in the previous section that the JavaScript code in a Cordova application fails silently? Well, you can also wrap the code in a try/catch block so your application will at least have the chance to write its error to the console, as shown in the following example:

```
try {
  console.log("Validating the meaning of life");
  someBogusFunction("42");
} catch (e) {
  console.error("Hmmm, not sure why this happened here: " +
    e.message);
}
```

Notice that in Figure 6.7, the Android LogCat shows you the line number where the console message was generated. This helps you identify information about where the application is failing. You could also use an alert here, but that's slightly less elegant.

Debugging and Testing Using External Tools

There's a very active partner community supporting Cordova with additional tools for Cordova developers. In this section, I introduce a couple of the more popular tools that help developers test and debug Cordova applications. This is by no means a complete list of options; refer to the PhoneGap Tools page (http://phonegap.com/tool) for information on additional tools that might be available. Some built-in debugging tools are also available with several of the mobile SDKs. These tools are covered in the individual chapters for each mobile OS (Chapters 7 through 10).

Debugging Applications with Weinre

Web Inspector Remote (weinre) is a community-built remote debugger for web pages. It was donated to the PhoneGap project and is currently implemented as part of the PhoneGap Build service. You can find the download files and instructions at http://people.apache.org/~pmuellr/weinre/docs/latest.

For Cordova development, it allows you to remotely debug a web application running in a Cordova container on a physical device or a device simulator. Weinre consists of a debug server, debug client, and debug target. The debug server runs on Macintosh or Windows, and the debug client runs in any compatible desktop browser.

To configure weinre, you need to perform a series of steps. The process begins with the server installation. Weinre is Node.js-based, and since we already have Node installed for the Cordova CLI, you can install the server using the following command:

```
npm install -g weinre
```

Unfortunately, on Macintosh weinre may not like your security configuration, so even though it's not recommended, you may have to install weinre using `sudo` using the following command:

```
sudo npm install -g weinre
```

After the installation completes, you should see a message similar to the following:

```
weinre@2.0.0-pre-HH0SN197 /usr/local/lib/node_modules/weinre
├── underscore@1.3.3
├── coffee-script@1.3.3
├── nopt@1.0.10 (abbrev@1.0.4)
└── express@2.5.11 (qs@0.4.2, mime@1.2.4, mkdirp@0.3.0, connect@1.9.2)
```

With the installation completed, you can start weinre by issuing the following command in the terminal window:

```
weinre
```

When the server starts, it will indicate that it is running by displaying a message in the terminal window similar to the following:

```
2013-06-22T17:00:50.564Z weinre: starting server at http://localhost:8080
```

> **Note**
> There are some command-line options you can pass to the weinre server at startup. I chose not to cover them here, but you can find detailed information on the weinre website at http://people.apache.org/~pmuellr/weinre/docs/latest/Running.html.

With the weinre server started, you use a browser-based client application to interact with the server and Cordova client application. Open your browser of choice (I recommend using Safari or Chrome) and point it to the URL shown on the server console when the weinre server started. For my development environment, I simply use:

```
http://localhost:8080
```

The browser will connect to the weinre server and open the weinre debug client, which will display a page similar to the one shown in Figure 6.8.

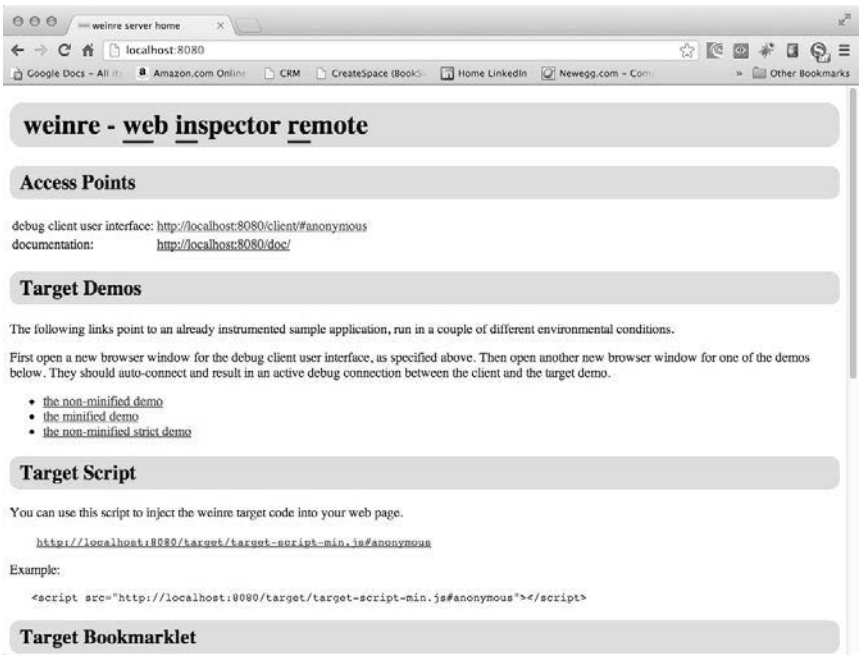


Figure 6.8 Weinre Debug Client Startup Page

With the server and client running, you can now connect a Cordova application to the debug server by adding the following script tag to `body` section of the Cordova application's index.html file:

```
<script src="http://debug_server:8080/target/target-script-min.js"></script>
```

You need to replace the `debug_server` portion of the URL with the correct host name or IP address for the debug server (the system running the weinre server). This makes the application into a weinre debug target and provides the Cordova application with the code needed to upload information to the weinre server as the application runs.

When using weinre with a device simulator, you can usually point the Cordova application to the local weinre server instance using

```
<script src="http://localhost:8080/target/target-script-min.js"></script>
```

The Android emulator, however, does not have the ability to connect to host-side resources using localhost, so for the Android emulator you must use the host address http://10.0.2.2, as shown in the following example:

```
<script src="http://10.0.2.2:8080/target/target-script-min.js"></script>
```

When using weinre to debug a Cordova application running on a physical device, the device must be able to connect to your debug server. That means that the device must be able to "see" the server on the local network (most likely over a Wi-Fi connection), or the system running the weinre server must have a public facing IP address. Using a server host name of localhost will not work on a physical device; you must use an actual host name or IP address that is visible to the device.

> **Warning**
>
> Be sure to remove the weinre script tag from your Cordova application before releasing it into production. The application will likely hang if attempting to connect to debug server that isn't available.

After you have added the script tag to the Cordova application's index.html file, run the application in the simulator or on a device. Nothing special will appear on the device screen—you can't tell that the weinre debug client is running. However, if you switch to the browser running the weinre debug client and click the first link, shown in Figure 6.8 (the one labeled "debug client user interface"), you will initially see a page similar to the one shown in Figure 6.9.

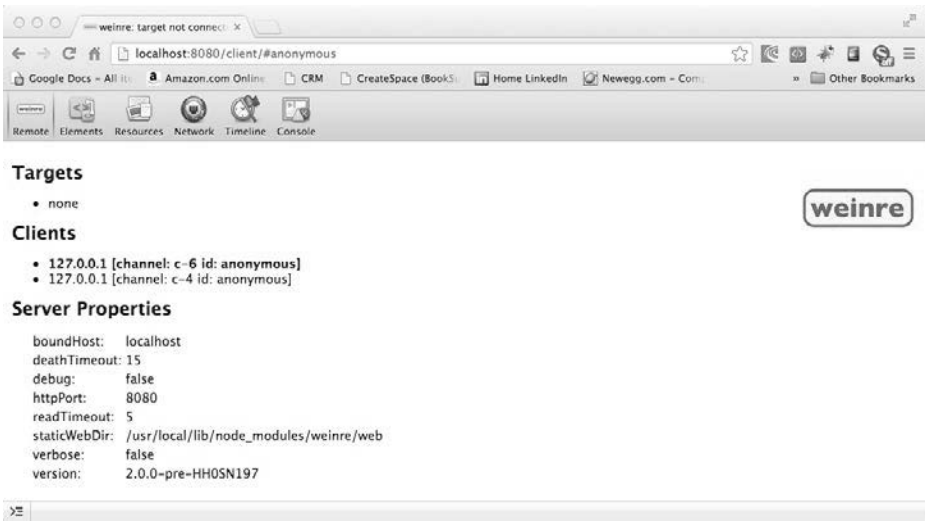


Figure 6.9 Weinre Debug Client

In this example, the figure is indicating that no targets have connected yet, but as soon as I start my Cordova application, as long as it can connect to the weinre server, the debug client page will update and display the content shown in Figure 6.10.

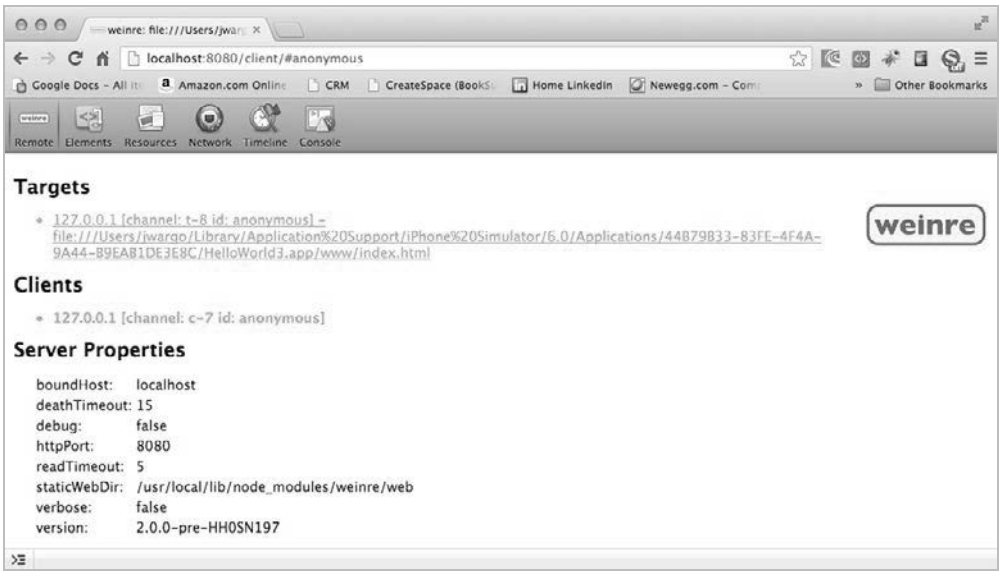


Figure 6.10 Weinre Debug Client with an Application Connected

The debug client provides the means to view and optionally manipulate many of the page elements and other aspects of your application's web content.

At this point, the different buttons across the top of the debug client are available to provide you with information about the debug target. For example, in Figure 6.11 you see the contents of the Elements page; it shows you the current HTML5 content running within the debug target.

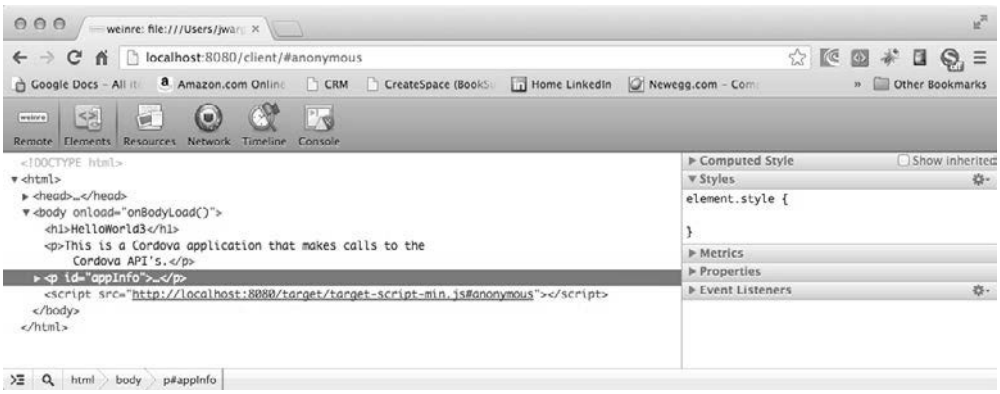


Figure 6.11 Weinre Debug Client Resources Area

One of the cool features of weinre is that as you highlight the different code sections shown in Figure 6.11, weinre will highlight the corresponding content within the web application. So, for the HelloWorld3 application shown in Figure 6.11, highlighting the paragraph tag `<p id= "appInfo">_</p>` reveals, in the debug target, the section of the page shown in Figure 6.12. In this example, I kept the content of the paragraph tag collapsed in the debug client. You can click the black triangle to the left of the `<p>` element to see the complete HTML content.

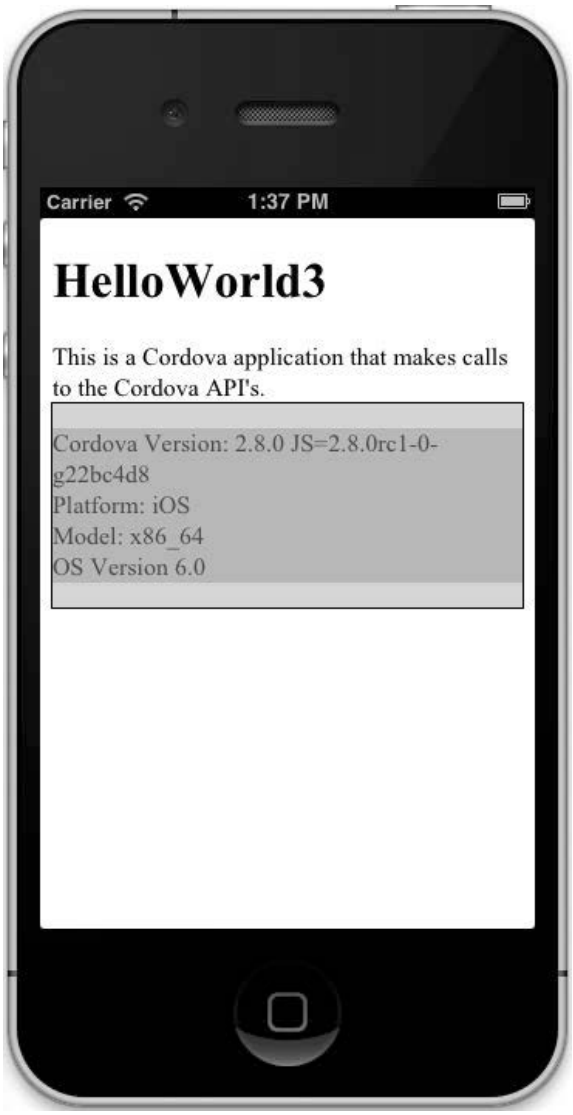


Figure 6.12 Weinre Target Highlighting HTML Content

Using the debug client, you can access the following content areas:

- Elements: The HTML, CSS, and JavaScript code for the application
- Resources: Local resources used by the application, such as databases, local storage, and session storage
- Network: Information about requests made using the XMLHTTPRequests (XHR)
- Timeline: Events that occur within the target application
- Console: Information written to the console using the `console` object described earlier in the chapter

The available documentation for Weinre is pretty light, but since the project's capabilities are based on the Google Chrome Developer Tools, you can find additional information on the Google Code website at http://code.google.com/chrome/devtools/docs/overview.html.

Testing Applications Using the Ripple Emulator

The Ripple Emulator is a tool you can use to help with the initial testing of your Cordova application. Ripple is a browser-based emulator that can be used to emulate several different systems. Originally created by Tiny Hippos, which was then acquired by Research In Motion (now called BlackBerry), Ripple is now an incubator project at Apache. The problem with Ripple is that it's been in beta for a very long time (almost two years by my counting), and the emulator is way behind on its Cordova support (supporting Cordova 2.0 when Cordova 2.8 was just released). Because of those limitations, I don't go into too much detail about how Ripple works.

Ripple emulates the execution of the Cordova APIs within the browser container. You can use Ripple for quick testing of Cordova application features and UI during development, then switch to packaging/building Cordova applications and testing them on actual devices or device simulators for more thorough testing. Ripple is not designed to replace testing on real devices or simulators.

Since Ripple was a BlackBerry project for a while, it has a lot of features that help BlackBerry developers. You can, for example, test your BlackBerry WebWorks applications using Ripple, then package them into WebWorks applications directly from the browser. You can learn more about Ripple's capabilities at https://developer.blackberry.com/html5/documentation/getting_started_with_ripple_1866966_11.html.

The emulator installs as a Google Chrome plugin, so you will need to install Chrome from www.google.com/chrome before you begin. Because Ripple is an incubator project and may become a full Apache project at any time, any URL I give you now may be invalid by the time you read this. So, to install Ripple, you should point your browser to http://emulate.phonagep.com. Follow the links on that page to download and install Ripple in your instance of the Chrome browser.

Once you have Ripple installed, you must enable file access for Ripple. In Chrome, open the settings page, then select the Extensions section. In the list of plugins that appears, enable the "Allow access to file URLs" option, shown in Figure 6.13.

Figure 6.13 Enabling Ripple File Access in the Chrome

Once the browser is configured, open your application's index.html file in the browser. You can press Ctrl-O on Windows or Command-O on Macintosh to open the File Open dialog. Once the page has loaded, you need to enable Ripple for the selected page. To do this, click the Ripple icon to the right of the browser's address bar to open a window allowing you to enable Ripple for the loaded page. You can also append `?enableripple=true` to the end of any URL to enable Ripple emulation for that page.

With Ripple enabled, the browser will display a page that prompts you to identify which type of emulation you wish to enable, as shown in Figure 6.14. As you can see, Ripple can emulate Apache Cordova plus several other platforms and frameworks. Click the Cordova 2.0 button to continue.

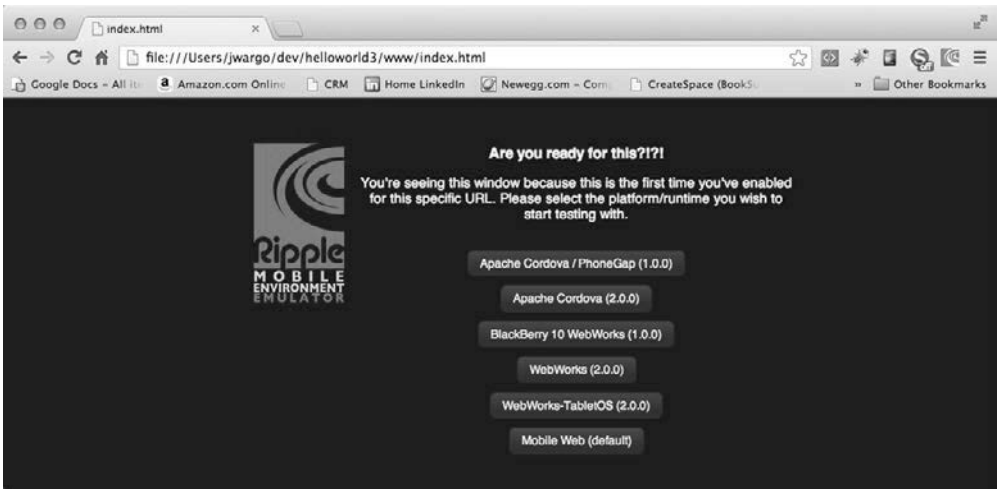


Figure 6.14 Ripple Emulation Platform Selection Page

At this point, Ripple will display a page with the content from the index.html file rendered within the boundaries of a simulated smartphone screen, as shown in Figure 6.15. Wrapped around the simulated smartphone are properties panes that can be used to configure options and status for the simulated smartphone, such as simulated device screen resolution, accelerometer, network, geolocation, and more.

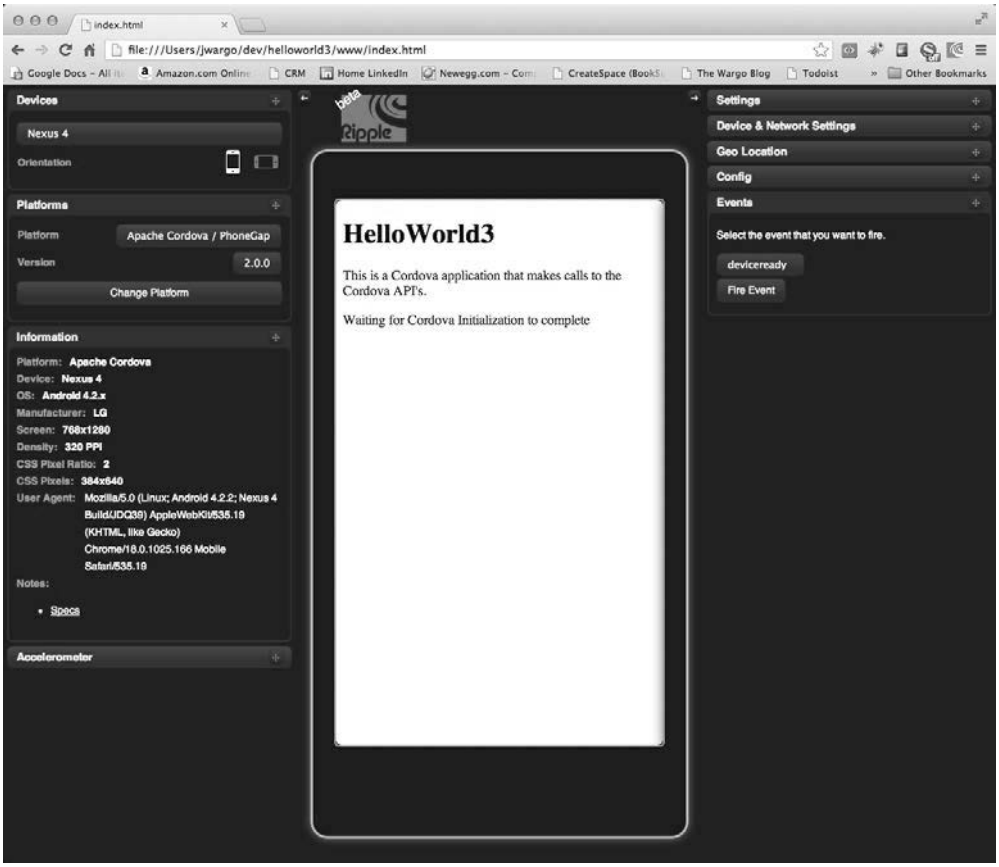


Figure 6.15 Ripple Emulator Running a Cordova Application

You can click on each of the tabs to expand the options for the tab and make changes to the simulated device's configuration. At this point, you would simply click around within the simulated smartphone screen and interact with the options presented within your application. When you find a problem or a change you want to make within the Cordova application, simply return to your HTML editor, make the necessary changes, write the changes to disk, then reload the page in the Chrome browser to continue with testing.

Wrap-Up

Hopefully by now, you have an inkling of how to build and debug Cordova applications. In the four chapters that follow, I show you how to use the platform-specific development tools and debugging capabilities.