Developing Cross Platform Mobile Applications with Cordova CLI

Dan Moore

Developing Cross Platform Mobile Applications with Cordova CLI

Use the command line to build your mobile apps with JavaScript, CSS and HTML

Dan Moore

This book is for sale at http://leanpub.com/developingwithcordovacli

This version was published on 2013-10-04

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Dan Moore by spreading the word about this book on Twitter!

The suggested tweet for this book is:

#cordovacli rocks

The suggested hashtag for this book is #cordovacli.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#cordovacli

Contents

Even though this book is small, much gratitude is owed.

Thanks to Pam Moore, my wife, for supporting me in writing this book.

Thanks to the folks at Nitobi and Adobe, and everyone else who has helped create Cordova, a truly fantastic and democratizing mobile development platform.

Thanks to the creators of LeanPub for building a platform that makes writing a book fun.

Thanks to 8z Real Estate for paying me to learn the intricacies of Cordova, and not minding if I shared some of what I learned with The Internet.

Thanks to Ralph Dosser for reviewing this text. Any mistakes are of course mine.

The cover image is courtesy of Marc Lagneau[1].

[1] http://www.flickr.com/photos/marc-lagneau/

Who Is This Book For?

This book is for you, the Cordova or PhoneGap developer who is interested in using the full power of the Cordova CLI, the command line interface for managing Cordova projects.

You know enough about Cordova to have installed at least one device platform toolset. You can build the default application created by running `cordova create sample-cordova-application` and install it on an emulator.

> Almost everything in this book should apply to PhoneGap, because PhoneGap is based on Cordova. In the rare case where it does not, I will note it.

You have installed Cordova CLI using `npm` by following the instructions, either for Cordova 3.0[2] or Cordova 2.9[3].

What Is Covered

The advanced features of Cordova CLI as well as a discussion of how to apply some software techniques, such as deployment environments and version, control to Cordova projects.

Cordova CLI depends on Cordova and has the same version as Cordova. The command line interface was not fully released until Cordova 2.9. I'll be covering both Cordova CLI 2.9, released in June, 2013, which is the last of the monolithic releases and will be supported for a long time[4]) and Cordova CLI 3.0, released in July, 2013, which is the first of the releases with the new plugin based 3.x architecture. Specifically, all the examples were run with Cordova CLI 2.9.7 and Cordova CLI 3.0.9.

> Cordova 3.1 was released just as I was finishing this book, so, while I've taken a look at the docs, I haven't done exhaustive testing on it. That said, I expect 3.1 and 3.0 to be far more alike than 2.9 and 3.0.

Examples will be for the iOS and Android platforms, as those are the ones I am familiar with (as well as being the most common platforms for development). CLI strives to be platform agnostic, but there may be Cordova CLI nuances for other platforms missed.

This book will follow the creation of a sample Cordova application called `sample-cordova-application` for the iOS and Android platforms.

---

[2] http://cordova.apache.org/docs/en/3.0.0/guide_cli_index.md.html
[3] http://cordova.apache.org/docs/en/2.9.0/guide_cli_index.md.html
[4] http://www.infil00p.org/introducing-cordova-2-8-1-on-android/

Sample Project Directory Layout

```
sample-cordova-project/
  config/
    android/
      res/
    ios/
      Resources/
  merges/
    android/
      css/
    ios/
      css/
  platforms/
    ios/
      ...
    android/
      ...
  plugins/
    ios/
      ...
    android/
      ...
  test/
  www/
    js/
    css/
    img/
```

What Is Not Covered

Look elsewhere for help in coding the Cordova application (with JavaScript, HTML and CSS). PhoneGap Essentials: Building Cross-Platform Mobile Apps[5] is a fine book about the nuts and bolts of building an Cordova application.

In addition, this is not a basic tutorial of the Cordova CLI tools. The Cordova CLI guide for 3.0[6] and the guide for 2.9[7] are both useful. Tutorials like this one[8] are worth reading as well.

This book won't cover any areas where there is sufficient documentation available, such as writing your own plugin, but will point to helpful resources in passing.

[5] http://www.barnesandnoble.com/w/phonegap-essentials-john-m-wargo/1110925369
[6] http://cordova.apache.org/docs/en/3.0.0/guide_cli_index.md.html
[7] http://cordova.apache.org/docs/en/3.0.0/guide_cli_index.md.html
[8] http://blog.safaribooksonline.com/2013/07/19/streamline-cross-platform-development-using-apache-cordova-phonegap-cli/

Version Control

When you develop software, you should use version control. Cordova projects are no different. Using version control offers a number of distinct benefits. With version control, you can:

- share code easily with other developers.
- know exactly what version of each file in your codebase you have deployed, which is extremely helpful when debugging issues.
- roll back mistakes.
- move code between machines.
- check out a fresh version of the code and know exactly how to build the project in a replicable fashion.

Version controlling Cordova projects is fairly similar to version controlling other web development projects. However, the Cordova CLI generates some artifacts that should be excluded.

What to Version Control

After you run `cordova create sample-cordova-project`, a Cordova project has these directories in the project:

- `www`: your application's HTML, CSS and JavaScript files
- `merges`: platform specific HTML, CSS and JavaScript files
- `.cordova`: 'under the hood' cordova files and directories, as well as lifecycle hook scripts
- `platforms`: platform specific build directories
- `plugins`: plugins added via `cordova plugin add`

The `www`, `merges`, and `.cordova` directories should be versioned, and the `platforms` and `plugins` directories should not[9]. Add the `platforms` and `plugins` directories to your version control system's ignore file.

There is also a `.cordova` directory in your home directory that you should not version control. It does have an impact on your project–it's where the global Cordova libraries are kept–but it should be entirely managed by Cordova CLI and you can safely ignore it.

You are not limited to the above five directories. You can add additional directories in your project should you require them. A common additional top level directory would be `test`, to hold your unit tests. Such directories will be ignored by the Cordova CLI.

You can use all the features of your version control system, just as you would with any web development projects. Tags, branching, merges, etc, are all fair game. Version control meta directories like `CVS` and `.svn` are copied from the `www` directory to the platform `www` directories when you build. Therefore, someone decompiling

[9] http://stackoverflow.com/questions/16989933/what-parts-of-cordova-cli-generated-projects-can-be-safely-versioned-in-source-c

your application may have access to your internal directory structure. (You can use a hook to clean up the meta directories, though.)

The `plugins` directory is typically empty until you add a plugin. Some version control systems don't create empty directories. If you add a plugin but the `plugins` directory is not present, you'll receive an error.

Missing Plugins Directory Error Message

```
$ cordova platform add android
{ [Error: ENOENT, no such file or directory '/path/to/sample-cordova-application/plugins/a\
ndroid.json']
  errno: 34,
  code: 'ENOENT',
  path: '/path/to/sample-cordova-application/plugins/android.json',
  syscall: 'open' }
```

Limits of Version Control

Not everything in a production Cordova application can be version controlled. Larger environment components like your platform specific SDK can't be easily version controlled, though if you are using virtual machines, you can use vagrant[10] and setup scripts to attempt this.

In addition, there are some IDE settings, especially when releasing, that can't be version controlled. Building for iOS in Xcode, in particular, has some arcane build settings and processes that require use of the IDE to release. There are also Cordova bugs that are worked around through editing build settings in the IDE[11]. When I can't version control key project configuration like IDE settings, I do the next best thing–document them well.

[10] http://www.vagrantup.com/

[11] http://stackoverflow.com/questions/17351446/building-an-archive-for-xcode-4-6-release-with-phonegap-v-2-9-fails/17372031#17372031

Hooks

Hooks are pieces of code that are executed at certain points in the application build lifecycle. They let you extend the Cordova CLI framework in a number of ways.

There are two types of hooks–project specific and module level. Module level hooks are used if you leverage the Cordova CLI in a larger Node.js application. Module level hooks provide events that your larger project can attach to "using the standard EventEmitter methods"[12]. I have no experience with this and on researching for an example, this seems to be more of a "Cordova platform developer" type of feature, so I won't discuss this further.

Project level hooks, on the other hand, are indispensable parts of a Cordova CLI application. They are executable scripts run before and after each stage of the Cordova CLI lifecycle[13]. They live in a `sample-cordova-project/.cordova/hooks/before_xxx` or `sample-cordova-project/.cordova/hooks/after_xxx` directory, where `xxx` is the project lifecycle stage (prepare, build, etc). The current lifecycle stages are listed here[14], and here's an example[15] you can "borrow" from.

Hooks can be written in any programming language, but they are executed on every platform you are building your project on. So I'd stay away from C#, unless you are only deploying to Windows Phone. I have written shell scripts and Node.js scripts. Node.js is server side JavaScript, and is what Cordova CLI is written in. Node.js is a safe choice because you can be sure that script will run anywhere you are using Cordova CLI.

The first argument of hook scripts is the project base directory.

Accessing the base directory in a Node.js hook

```
#!/usr/bin/env node
var fs = require('fs');

var rootdir = process.argv[2];

// ... using rootdir
```

[12] https://npmjs.org/package/cordova
[13] https://github.com/apache/cordova-cli#project_commands
[14] https://github.com/apache/cordova-cli#hooks
[15] https://gist.github.com/dpogue/4100866

Accessing the base directory in a shell script

```
#!/bin/sh

ROOTDIR=$1

# ... using $ROOTDIR
```

Because hooks have the base directory, they can manipulate files in all project directories. In the future, hooks will get access to more data[16], including which platform the build is targeting, but for now all the scripts receive is the base directory.

You can use hooks for a wide variety of tasks, including:

- adding needed plugins to a project, in `after_platform_add`
- copying resources like icons and splash screens to appropriate locations within a platform directory, in `after_prepare`
- run unit tests written before building a binary, in `before_build`
- downloading remote API content and making it available to your application for quicker start up, in `before_prepare`.
- moving platform specific configuration files (like `AndroidManifest.xml`), in `after_platform_add`. Make sure this happens before you install any plugins, because they can also modify platform configuration files.
- injecting deployment specific values, such as which server the mobile application should be communicating with, in `after_prepare`.
- changing native code to support multiple versions of Cordova, if you need to support multiple versions with breaking changes. For example, the package of `CordovaPlugin`, which changed between 2.9 and 3.0 could be modified.

If a hook is going to be manipulating a file that you change a lot during development, like a JavaScript file containing business logic, you want to have the hook execute during a common lifecycle stage, like `after_prepare`; otherwise you can have the hook execute after a less common event, which will speed up your build process.

Execution

You can see which hook scripts are executing by running the CLI with the verbose switch: `cordova -d [command]`. Each hook must be executable, otherwise you will get an error message.

[16] https://issues.apache.org/jira/browse/CB-4591

Not Executable Hook Script Error Message

```
[Error: Script "/path/to/sample-cordova-application/.cordova/hooks/after_prepare/hookscrip\
t.sh" exited with non-zero status code. Aborting. Output: /bin/sh: /path/to/sample-cordova\
-application/.cordova/hooks/after_prepare/hookscript.sh: Permission denied
```

Each hook is executed in OS specific order within the directory containing all the scripts for a lifecycle stage (`before_xxx` or `after_xxx`). On Windows, `ae.js` runs before `aG.js`, but on linux, `aG.js` runs before `ae.js`. Rather than rely on this implicit order, you should prefix each hook with a three or four digit number, to make the execution order explicit. For example:

- `001_script.js`
- `010_script.js`
- `020_script.sh`
- `120_script.js`

executes these scripts in the order you would expect, where:

- `copy_resources.js`
- `modify_config_files.sh`
- `delete_modified_files.js`
- `copy_files.js`

will execute in this order:

- `copy_files.js`
- `copy_resources.js`
- `delete_modified_files.js`
- `modify_config_files.sh`

which could have unfortunate side effects.

Stopping builds

Hooks can also stop execution of a build. If, for example, you are running your unit tests on every `before_prepare` event, and the unit tests fail, the build should stop–no sense in deploying to an emulator or building for a device if the unit tests fail. Stopping the build might be useful if a remote API is not available and you require it to build the application because you are preloading some data.

In order to stop execution, you can exit your hook script with a non zero exit code. You can also throw an error in a Node.js script, but exiting is cleaner.

Stopping Build Execution in a Node.js Hook

```
#!/usr/bin/env node

console.log("unable to complete the hook");
process.exit(1);
```

You will see an error in the build process like

Output of Stopping Execution

```
$ cordova build android
[Error: Script "/path/to/sample-cordova-application/.cordova/hooks/after_prepare/hookscrip\
t.js" exited with non-zero status code. Aborting. Output: unable to complete the hook
```

Hooks in Node.js

If you choose to write Node.js scripts, use the synchronous API for your filesystem operations. Node.js defaults to asynchronous filesystem access. While this is non-blocking and faster than the synchronous access, asynchronous access can lead to confusing results when moving or processing the same files with more than one hook. For example, operations you think will be completed in order are reversed, but only sometimes. Yes, faster builds are always better, but I have found the majority of my development time was spent waiting on emulators or thinking, not on hook scripts.

One benefit of using Node.js scripts, beyond being cross platform, is that you can abstract common functionality out into a local `npm` package and share it between different Cordova projects. You can even leverage `npm`'s versioning system so that bugfixes can migrate between your hook scripts.

If you are using Node.js scripts on windows, make sure that Node.js is the default handler for files with the `.js` suffix. If another program handles your JavaScript hook scripts, you may see bizarre errors. To correct it, follow these instructions[17].

[17] http://superuser.com/questions/475915/use-node-js-as-the-default-application-for-opening-js-files