**Cordova**

**Introduction**

Cordova sponsored by the Apache Foundation, Cordova allows mobile developers to write for iOS, Android, Windows and other platforms using Web development technologies like HTML, CSS and JavaScript. Many other mobile development frameworks, most notably PhoneGap, are based on the Cordova code base. Applications execute within wrappers targeted at each platform, and rely on standards-compliant API bindings to access each device’s sensors, data and network status.

**Cordova and PhoneGap**

PhoneGap is the name of the original project created by a startup called Nitobi around 2009. In 2011, the company was acquired by Adobe. Adobe didn’t buy the actual codebase, just the name and the people who worked at Nitobi. The open source core was donated to the Apache Software Foundation, which is now known as Cordova.

A common analogy used by developers is that Cordova is to PhoneGap what WebKit is to Safari. Put differently, PhoneGap is a distribution of Cordova that is maintained by Adobe.

In practice, PhoneGap and Cordova are basically the same. The only difference is the CLI (Command Line Interface).PhoneGap’s CLI integrates with Adobe PhoneGap Builds. This service allows developers to compile their apps to the platform of their choice by uploading it to the PhoneGap Build website.

**Architecture**

There are several components to a Cordova application. The following diagram shows a high-level view of the Cordova application architecture.



**Working of Cordova**

Cordova’s user interface is a web view. You can think of the web view as a tab in a browser. When you compile a Cordova application, it doesn’t actually take your HTML, CSS, and JavaScript code and automagically converts it into native code, specific to each platform.

Cordova acts as a container for the app that you write using web technologies. When the app is compiled, your code actually stays intact. The compiler just takes your code and makes it available to the web view for rendering.

If you’ve ever opened an HTML file in a browser, that’s basically the same thing. You have to keep in mind that this web view is implemented differently on every device. This means that there are certain features that are not available on some platforms or they are implemented differently.

**WebView**

The Cordova-enabled WebView may provide the application with its entire user interface. On some platforms, it can also be a component within a larger, hybrid application that mixes the WebView with native application components.

**Web App**

This is the part where your application code resides. The application itself is implemented as a web page, by default a local file named index.html, that references CSS, JavaScript, images, media files, or other resources are necessary for it to run. The app executes in a WebView within the native application wrapper, which you distribute to app stores.

Config.xml (Config.xml is a global configuration file that controls many aspects of a cordova application's behavior) file that provides information about the app and specifies parameters affecting how it works, such as whether it responds to orientation shifts.

## Plugins

Plugins are an integral part of the Cordova ecosystem. They provide an interface for Cordova and native components to communicate with each other and bindings to standard device APIs. This enables you to invoke native code from JavaScript.

Apache Cordova project maintains a set of plugins called the Core plugins. These core plugins provide your application to access device capabilities such as battery, camera, contacts, etc.

In addition to the core plugins, there are several third-party plugins which provide additional bindings to features not necessarily available on all platforms. You can search for Cordova plugins using plugin search or npm. You can also develop your own plugins. Plugins may be necessary, for example, to communicate between Cordova and custom native components.

## Development Paths

Cordova provides you two basic workflows to create a mobile app. While you can often use either workflow to accomplish the same task, they each offer advantages:

* **Cross-platform (CLI) workflow**: Use this workflow if you want your app to run on as many different mobile operating systems as possible, with little need for platform-specific development. This workflow centers around the cordova CLI. The CLI is a high-level tool that allows you to build projects for many platforms at once, abstracting away much of the functionality of lower-level shell scripts. The CLI copies a common set of web assets into subdirectories for each mobile platform, makes any necessary configuration changes for each, runs build scripts to generate application binaries. The CLI also provides a common interface to apply plugins to your app.
* **Platform-centered workflow**: Use this workflow if you want to focus on building an app for a single platform and need to be able to modify it at a lower level. You need to use this approach, for example, if you want your app to mix custom native components with web-based Cordova components. As a rule of thumb, use this workflow if you need to modify the project within the SDK. This workflow relies on a set of lower-level shell scripts that are tailored for each supported platform, and a separate Plugman utility that allows you to apply plugins. While you can use this workflow to build cross-platform apps, it is generally more difficult because the lack of a higher-level tool means separate build cycles and plugin modifications for each platform.

When first starting out, it may be easiest to use the cross-platform workflow to create an app. You then have the option to switch to a platform-centered workflow if you need the greater control the SDK provides.

**Installation of Cordova**

The Cordova command-line tool is distributed as an npm package.

To install the cordova command-line tool, follow these steps:

1. Download and install Node.js. On installation you should be able to invoke node and npm on your command line.
2. (Optional) Download and install a git client, if you don't already have one. Following installation, you should be able to invoke git on your command line. The CLI uses it to download assets when they are referenced using a url to a git repo.
3. Install the cordova module using npm utility of Node.js.
4. The cordova module will automatically be downloaded by the npm utility.

**In command-line on Windows:**

c:\> npm install -g cordova

The -g flag above tells npm to install install globally. Otherwise it will be installed in the node\_modules subdirectory of the current working directory.

**In terminal on Mac OS X/Linux:**

$ sudo npm install -g cordova

On OS X and Linux, prefixing the npm command with sudo may be necessary to install this development utility in otherwise restricted directories such as /usr/local/share. If you are using the optional nvm/nave tool or have write access to the install directory, you may be able to omit the sudo prefix.

## Updating Cordova

After installing the cordova utility, you can always update it to the latest version by running the following command:

**$ sudo npm update –g cordova**

Syntax to install a specific version:

**$ sudo npm install –g cordova@3.1.0-0.2.0**

Run cordova-v to see which version is currently running. To find the latest cordova version:

**$ npm info cordova version**