Comp 322/422 - Software Development for Wireless and Mobile Devices

Fall Semester 2017 - Week 2

Dr Nick Hayward

Image - Apache Cordova

Source - Apache Cordova

Apache Cordova - what can it do?

- designed to offer a simple, powerful set of API calls
  - *calls to JavaScript functions*
  - *functions map native OS code to plugins and code in Cordova*
  - *enables access to core functionality for a device*
- allows us to transfer, manipulate, control
  - *data and resources from the native OS and device*
  - *moves it to the web view in our Cordova app*
- allows us to provide same user experience as native app
  - *minus a few base caveats*
- cross-platform support

Apache Cordova - platform support

support includes following mobile OSs

- Android
- iOS
- Windows 10 Universal platform
- Ubuntu
- LG webOS
- BlackBerry 10
- ...

Apache Cordova - functionality and plugins

- allows us to create native mobile applications using a set of common web technologies
  - *including HTML5, CSS, and JavaScript*
- a set of JavaScript APIs
  - *provides access to natively built core plugins*
- currently offers many core APIs
- includes some of the following native functionality,
  - *access the device's microphone for recording &c.*
  - *capture photos using the device's camera*
  - *photo retrieval from the OSs gallery/photo album*
  - *retrieve device information*
    - locale
    - various sensors such as motion, location, connection information, compass...
  - *retrieve device data, contact information...*
  - *process files from/to storage*
  - ...

Apache Cordova - documentation and APIs

official *Cordova* API documentation is currently available at the following URL,

- Apache Cordova API
- Apache Cordova GitHub
- Android API
- iOS API
- ... & many others

Apache Cordova - why choose it?

- potential to develop once, re-use with ease
- Cordova helps us solve some of the following mobile development issues,

1. different programming languages for different mobile OSs
   - *different programming philosophies, conventions, best practices, guidelines...*
2. unique problems inherent to each given mobile OS
   - *e.g. handling and routing SMS requests, data storage, privacy features...*
3. developing, testing, and maintaining applications across multiple mobile platforms
4. ...

Apache Cordova - overview of APIs

Platform APIs (cordova-plugin)	Android	iOS	Windows Universal Platform
Accelerometer (device-motion)	Yes	Yes	Yes
BatteryStatus (battery-status)	Yes	Yes	Yes
Camera (camera)	Yes	Yes	Yes
Capture (media-capture)	Yes	Yes	Yes
Compass (device-orientation)	Yes	Yes	Yes
Connection (network-information)	Yes	Yes	Yes
Contacts (contacts)	Yes	Yes	Yes
Device (device)	Yes	Yes	Yes
Events	Yes	Yes	Yes
File (file)	Yes	Yes	Yes
File Transfer (file-transfer)	Yes	Yes	Yes (no support for `onprogress` or `abort`)
Geolocation (geolocation)	Yes	Yes	Yes
Globalisation (globalization)	Yes	Yes	Yes
InAppBrowser (inappbrowser)	Yes	Yes	Yes
Media (media)	Yes	Yes	Yes
Notification (dialogs)	Yes	Yes	Yes
Splashscreen (splashscreen)	Yes	Yes	Yes

Platform APIs (cordova-plugin)	Android	iOS	Windows Universal Platform
Storage	Yes	Yes	Yes (localStorage & indexedDB)
Vibration (vibration)	Yes	Yes	Yes

Apache Cordova Documentation - Platform Support

Apache Cordova - API details

- many of these mobile native function APIs self explanatory
- a few examples,
- **capture**
  - *record various media directly from the native device*
- **connection**
  - *provides information about the device's wifi and cellular connections*
- **device**
  - *provides useful information on a device's hardware and software*
  - *native device model, the current platform and its version...*
- **events**
  - *particularly important, and useful, API*
  - *e.g.* `deviceready`, `backbutton`, `batterystatus`, *volume events...*
- **file** api to help process device files
- receive the device's location using GPS or network signals
- many more...
- Apache Cordova Documentation - Platform Support

Cordova App - intro

- working with Cordova - start building basic app from scratch
- helps introduce initial concepts underpinning Cordova app development
- look at some of the initial tools we'll be using
  - *to create, run, and deploy our applications*
- focusing on working with Cordova relative to Android and its native SDK
- setting up and configuring our Android development environment
- installing and configuring Apache Cordova, Node.js...
- getting familiar with the Cordova CLI (command line interface)
- develop and test some code

Cordova App - getting started

- initial focus will fall upon working with the CLI for Cordova
- Cordova CLI helps us create, develop, build, and test our first Cordova application
- Cordova CLI enables us to create our new project/s
- helps us build and target deployment on OSs such as Android, iOS, and Windows...

Cordova App - Android - SDK setup

- setup and configure Cordova development environment
- reference point is the Cordova Documentation section on **Platform Guides**
  - *Cordova Platform Guides*
- start work with the Android SDK
  - *download, install, and setup JDK*
  - *Java - JDK*
- Android SDK options include Android Stand-Alone SDK Tools or the recent Android Studio 2
- use **Android Studio** to help build custom Android plugins for Cordova...
- **Android Stand-Alone SDK Tools** sufficient to build and deploy an Android app
  - *command-line and Studio will both provide necessary SDK, build tools...*

Cordova App - Android - setup and configure Java JDK and Android SDK

- downloaded and moved Android SDK - need to configure settings, packages...
- to use the Cordova CLI tools we need to check our system's **PATH** settings
- tell our system where to find the Android SDK tools
- Unix based OSs need to update the **PATH** setting of **bash** profile

```
~/.bash_profile
```

- update these **PATH** settings using the following commands
- OS X

```
export ANDROID_HOME=/<installation location>/android-sdk-macosx
export PATH=${PATH}:$ANDROID_HOME/platform-tools:$ANDROID_HOME/tools
```

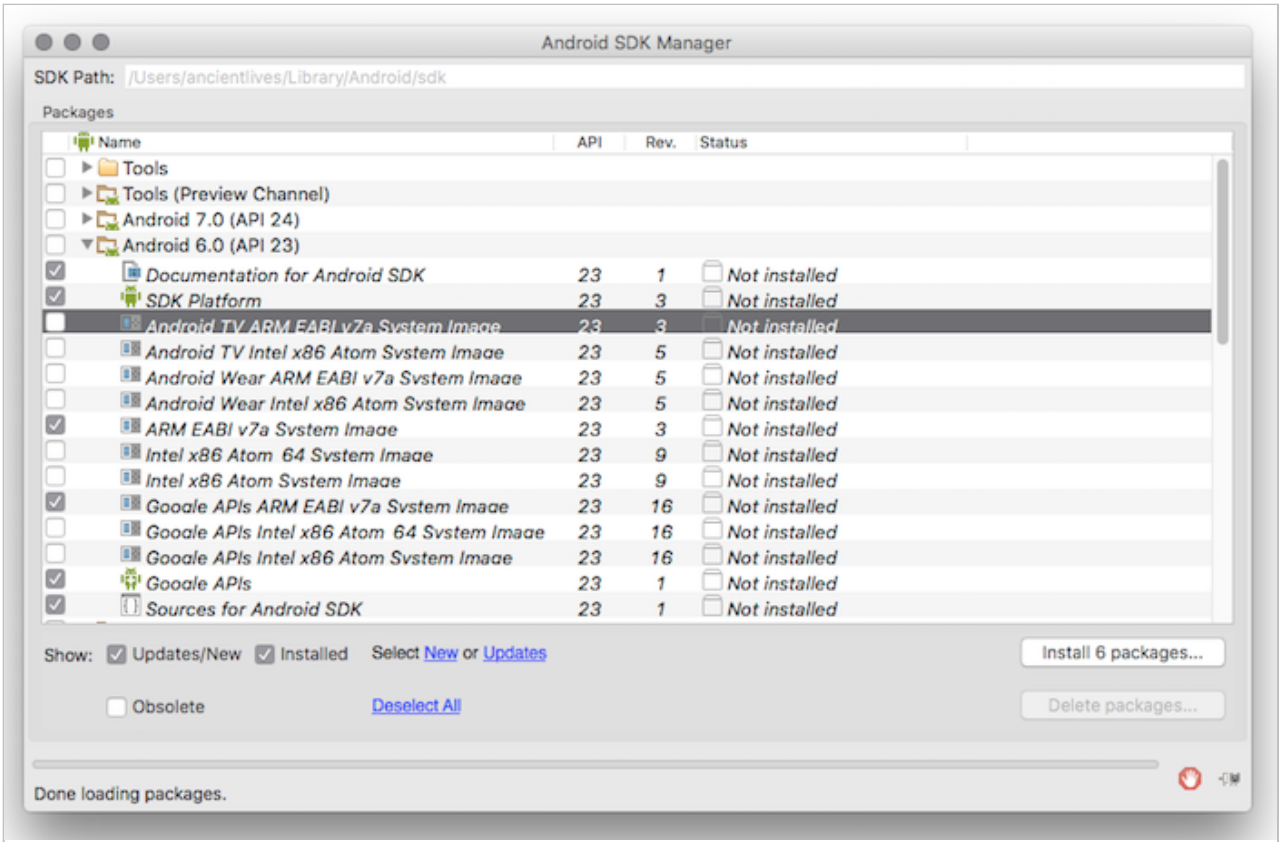
- Windows

```
set ANDROID_HOME=C:\<installation location>\android-sdk-windows
set PATH=%PATH%;%ANDROID_HOME%\tools;%ANDROID_HOME%\platform-tools
```

- start Android SDK manager using the following command

```
android
```

Image - Android SDK Manager

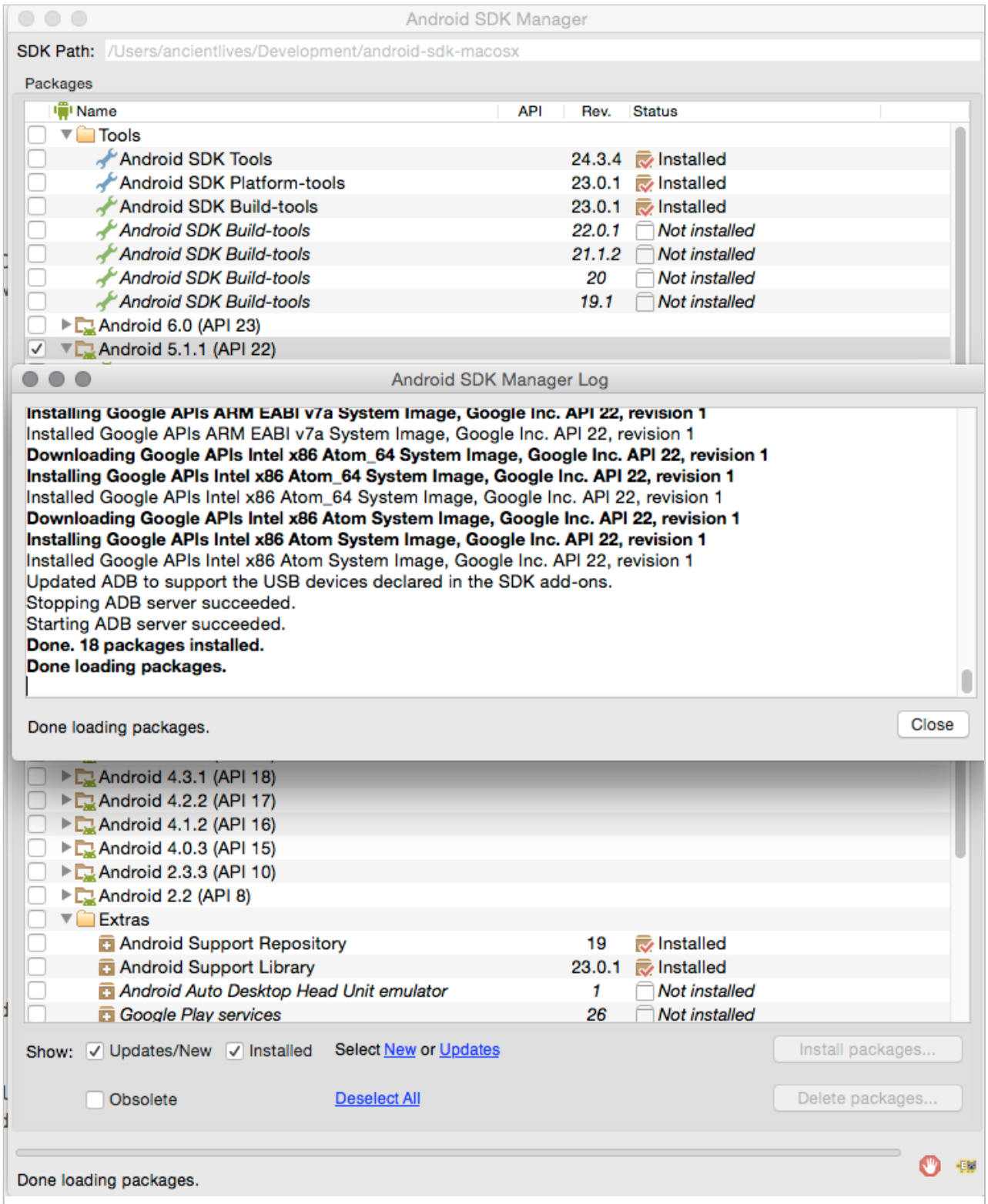


Android SDK Manager

Cordova App - Android - SDK extras

- initially, we need to add the following extra SDK packages for Cordova,
  - *e.g. Android 6.0 (API 23) platform SDK (latest support for API Level 25)*
  - *Android SDK Build-tools (latest version)*
  - *Android Support Repository (listed in the Extras section)*
- downloading and installing many different APIs for Android images
  - *e.g. Android Wear, Android TV, standard Android...*
  - *customise as required for app...*
- SDK manager will ask us to accept an *Android SDK* license
- you'll see a log window for the install process
  - *updates us relative to package installs, any potential issues...*
- may take a few minutes or so to complete

Image - Android SDK Manager Install



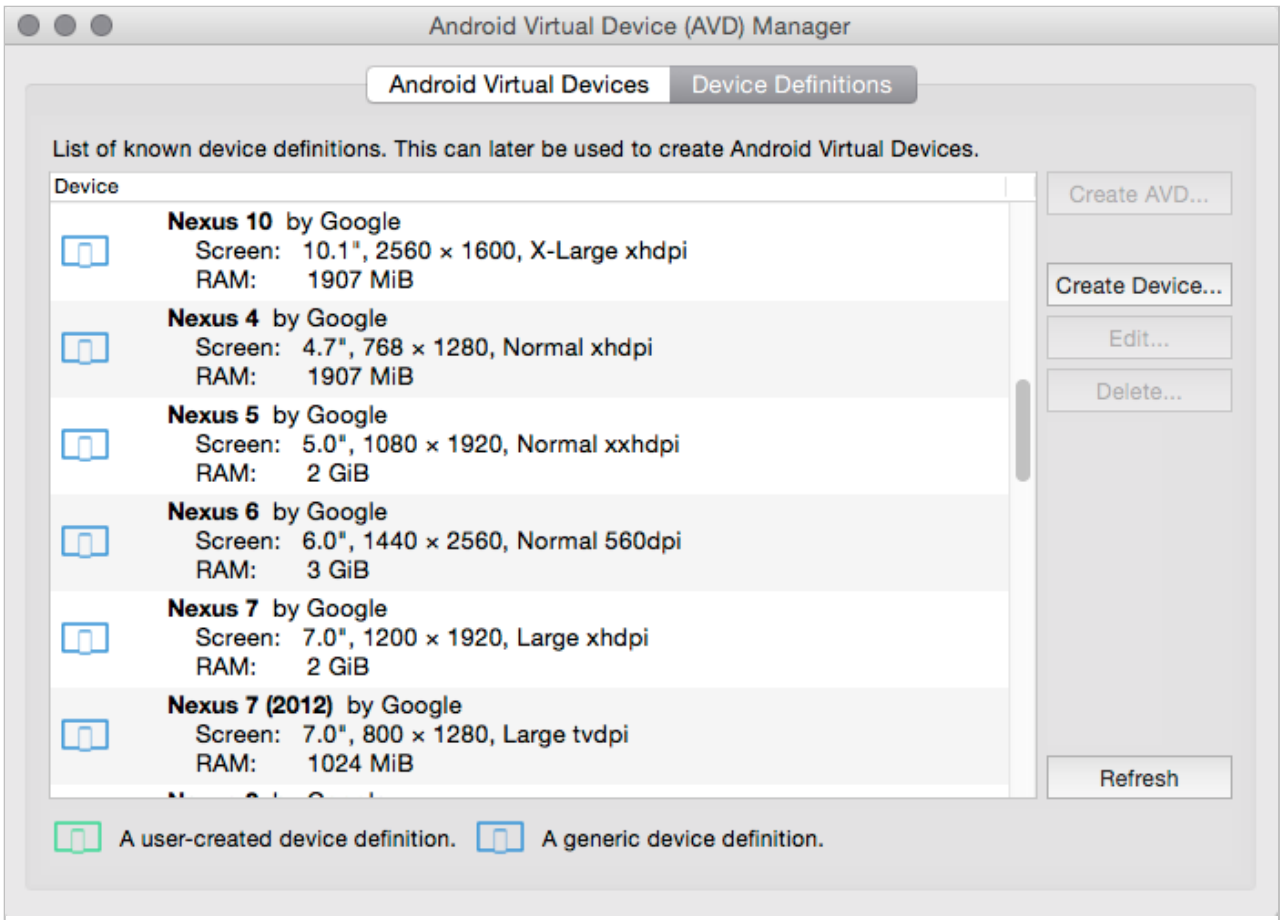
Android SDK Manager Install

Cordova App - Android - Emulator - part I

- close **Android package manager** to ensure that portion of setup is complete
- configure an emulator for testing and development of our applications
- couple of ways to setup and maintain AVD
  - *AVD manager from Stand-alone SDK*
  - *within Android Studio 2.x*
- Android SDK does not provide an existing instance of an emulator
- need to create a new one using the AVD management tool
- open these tools from the command line using the command,

```
android avd
```

Image - Android Device Definitions

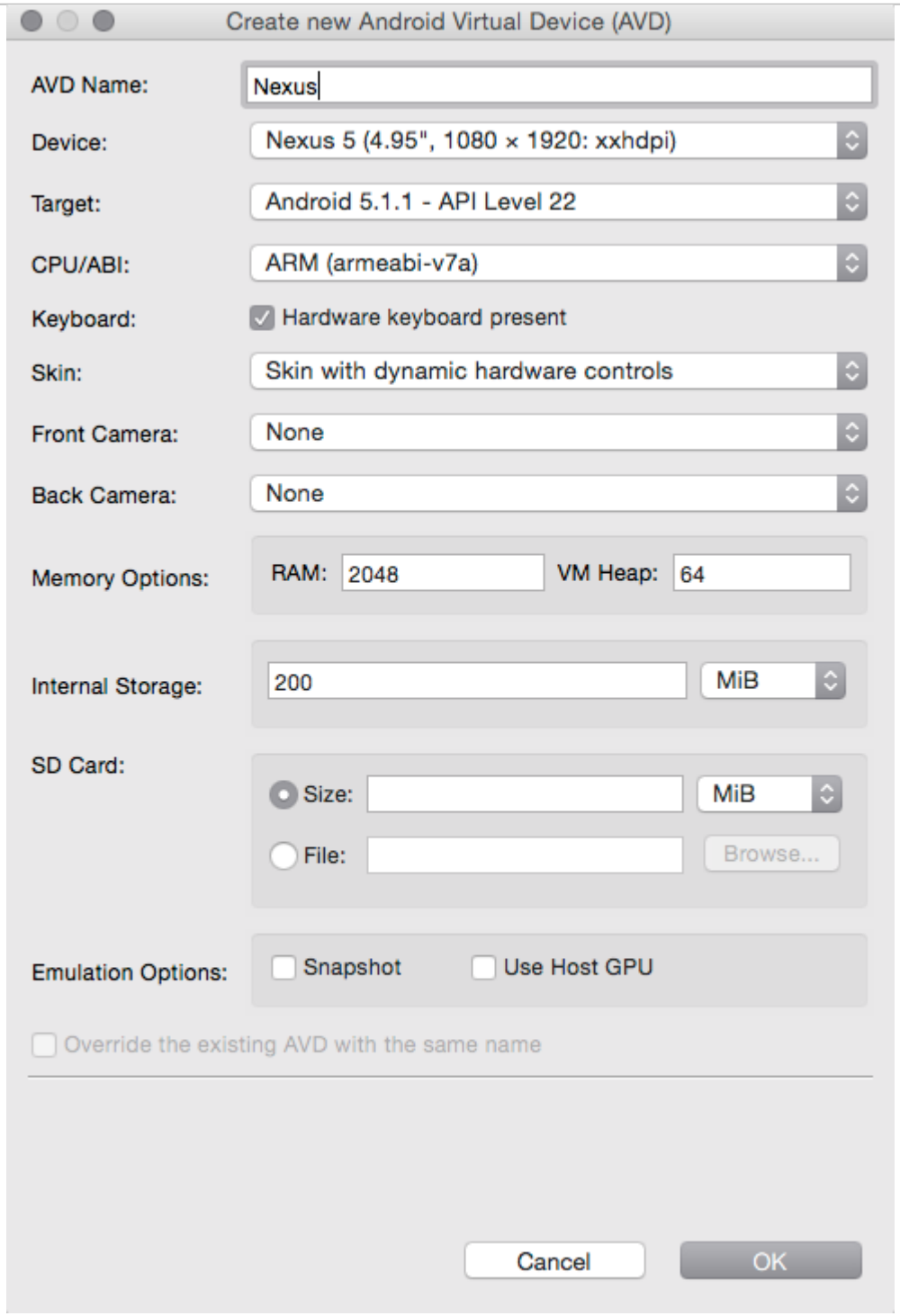


Android Device Definitions

Cordova App - Android - Emulator - part 2

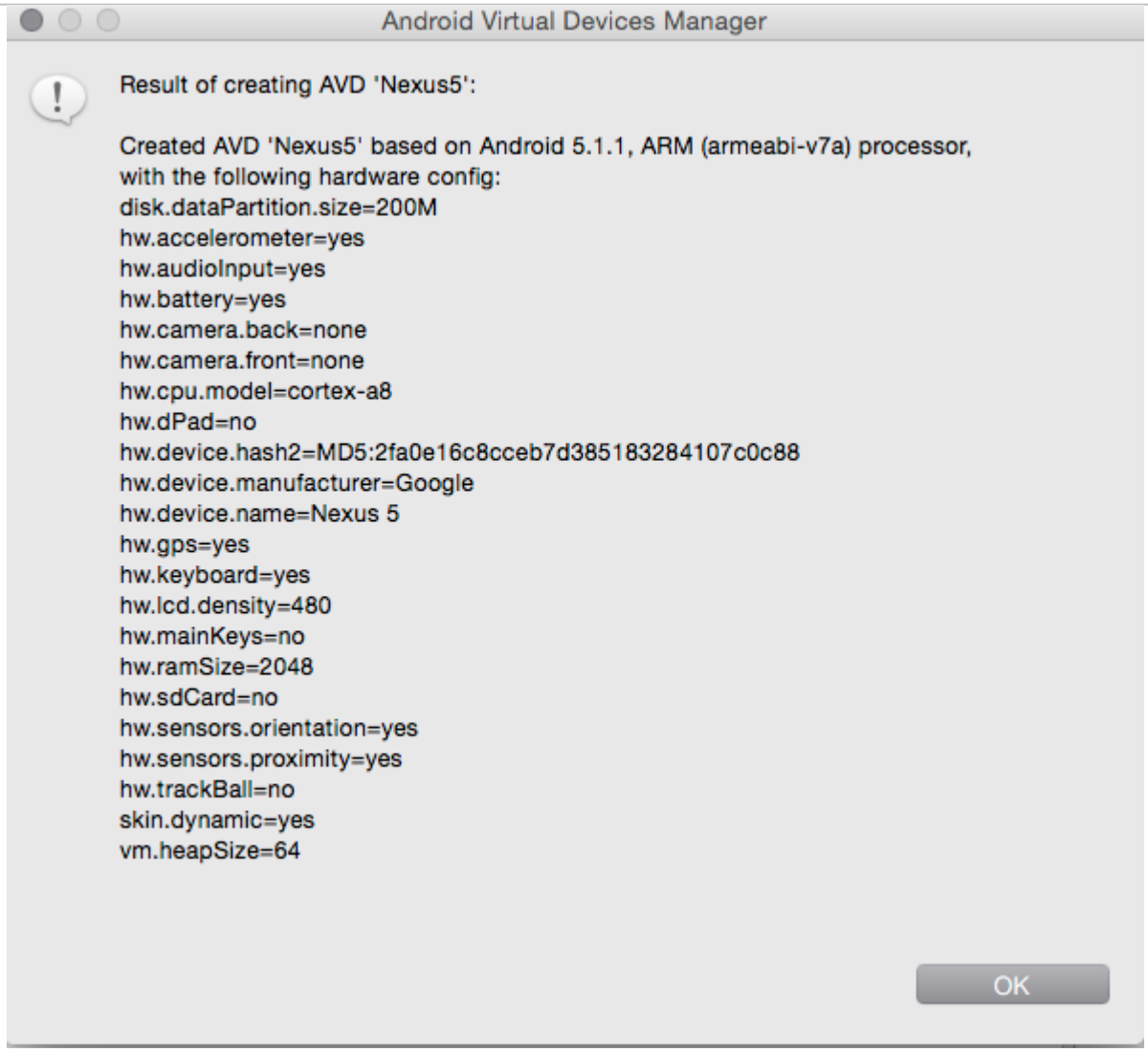
- many different device definitions
  - *e.g. Nexus devices (5, 6, 5x, 6P, Pixel...), generic definitions...*
- select your preferred *Device Definition* from the available list
  - *e.g. Nexus 5X...*
- then **Create AVD**
- modify a few settings to help device creation, e.g. for Nexus 5x
  - *optionally customise the* ***AVD Name***
  - *set the target, e.g.* ***Android 6.0 - API Level 23***
  - *set the CPU/ABI to* ***ARM (armeabi-v7a)***
  - *set skin to* ***Skin with dynamic hardware controls***

Image - Android Virtual Device



Android Virtual Device

Image - Android Virtual Device - Success



Android Virtual Device - Success

Cordova App - Android - Emulator - Android Studio 2.x AVD

- create AVD using Android Studio 2
  - *select `Tools > Android > AVD Manager`*
- select option `Create Virtual Device`
- select preferred device from **phone** category
  - *e.g. Nexus 5X, 6P...*
- select target **System Image**
  - *e.g. API Level 23 (Android 6.0)*
- then verify config for AVD
- check devices available for Cordova (when installed)

```
cordova run --list
```

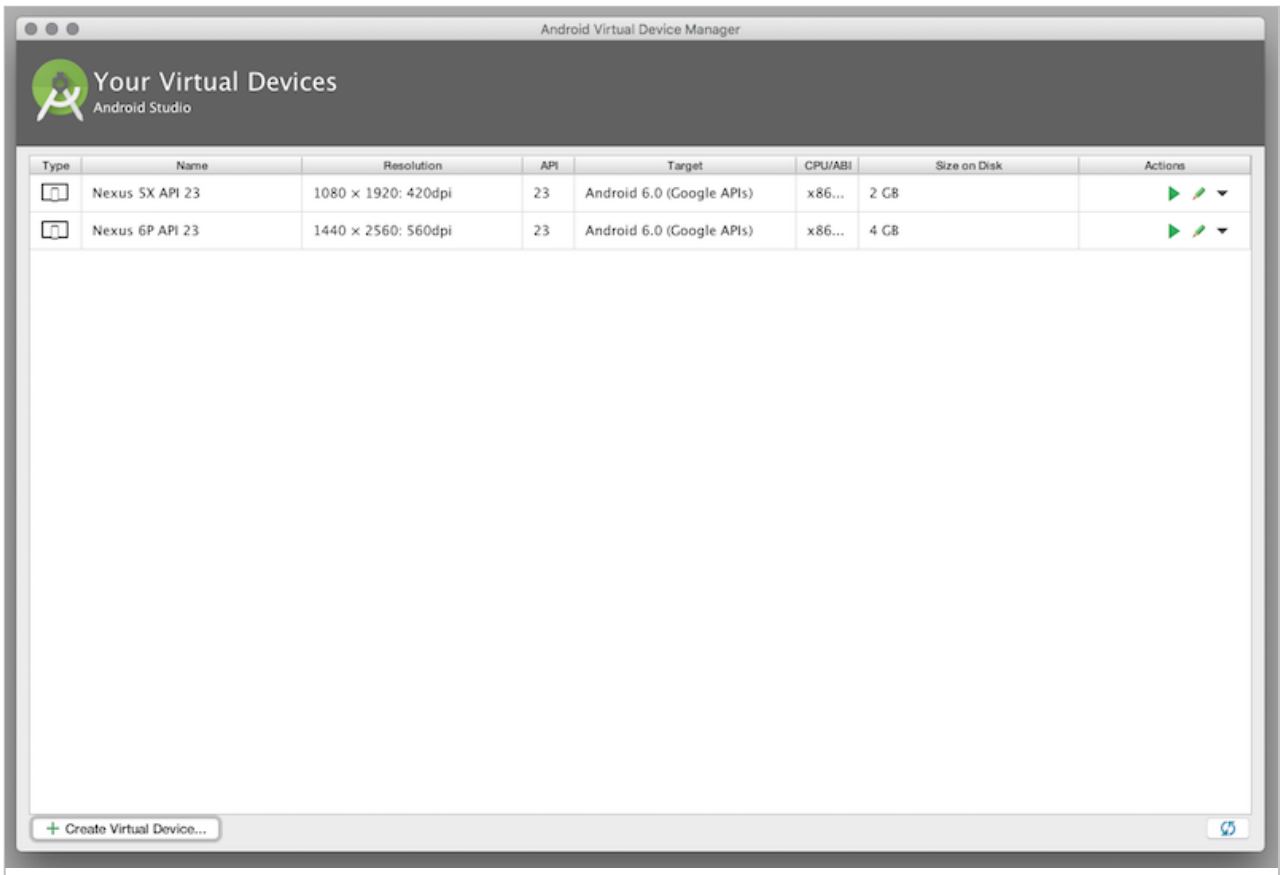
For example, this will output the following

```
Available android devices:
Available android virtual devices:
Nexus_5X_API_23
Nexus_6P_API_23
```

Cordova App - Device List Output

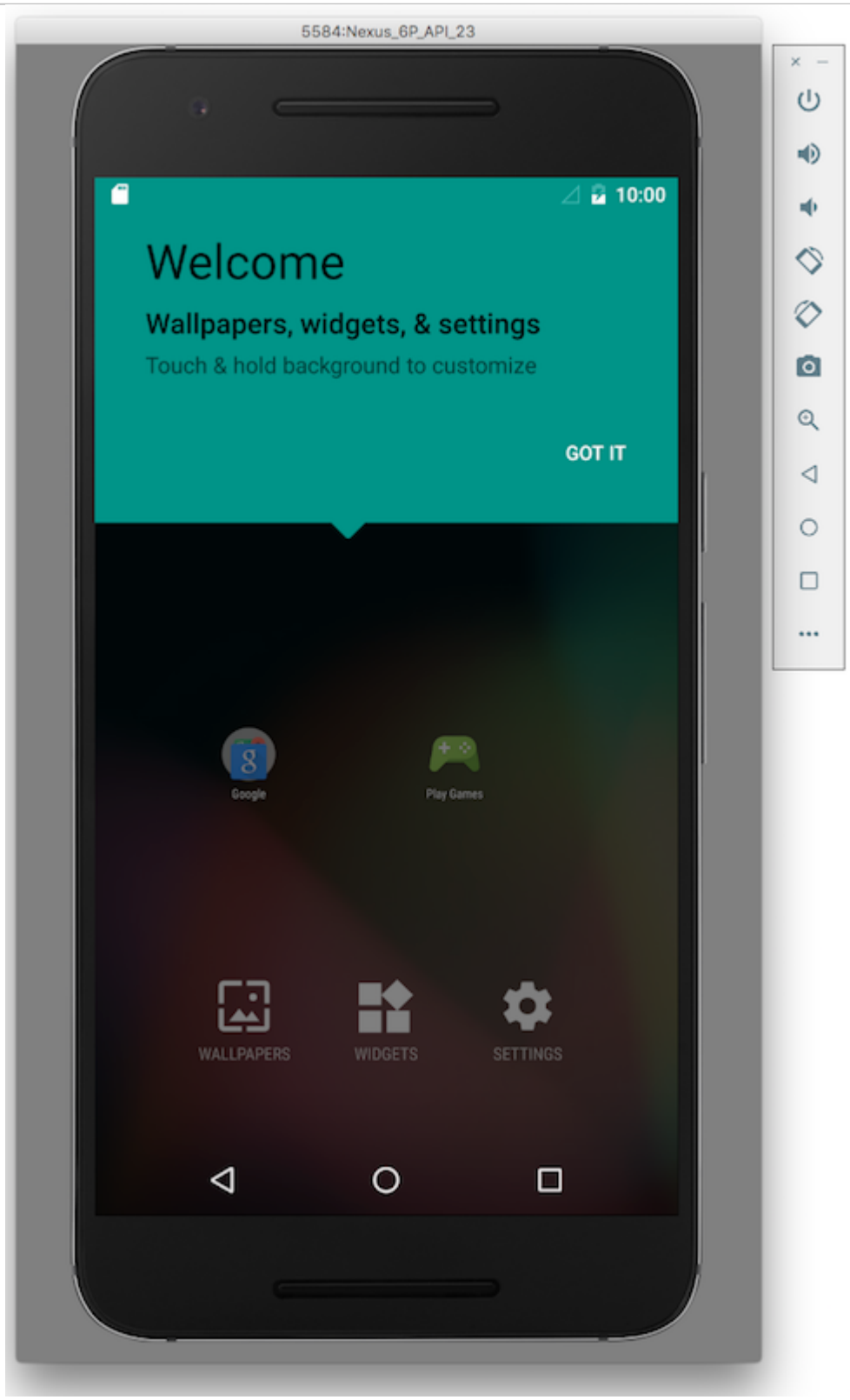
```
cordova run --list
Available android devices:
Available android virtual devices:
Nexus_5X_API_24_x86
Nexus_5X_API_25
Nexus_5X_API_26_x86
Available ios devices:
Available ios virtual devices:
Apple-TV-1080p, tvOS 10.2
iPhone-5, 10.3
iPhone-5s, 10.3
iPhone-6, 10.3
iPhone-6-Plus, 10.3
iPhone-6s, 10.3
iPhone-6s-Plus, 10.3
iPhone-7, 10.3
iPhone-7-Plus, 10.3
iPhone-SE, 10.3
iPad-Air, 10.3
iPad-Air-2, 10.3
iPad-Pro--9-7-inch-, 10.3
iPad-Pro, 10.3
iPad--5th-generation-, 10.3
iPad-Pro--12-9-inch---2nd-generation-, 10.3
iPad-Pro--10-5-inch-, 10.3
Apple-Watch-38mm, watchOS 3.2
Apple-Watch-42mm, watchOS 3.2
Apple-Watch-Series-2-38mm, watchOS 3.2
Apple-Watch-Series-2-42mm, watchOS 3.2
```

Image - Android Studio 2.x AVD Manager



Android Studio 2.x - AVD Manager

Image - Android Virtual Device - Nexus 6P



Android Virtual Device - Nexus 6P

Cordova App - Node.js Setup

- installed, setup, and configured Android SDK, and associated virtual device
- we also need to install and setup the **Cordova CLI**
- a few additional tools required including
  - *Node.js*
  - *Git*
- Node.js provides access to Node's JavaScript library and NPM
  - *NPM - Node Package Manager*
- Git is used by Cordova CLI to install various assets for new projects

Cordova App - CLI setup

- setup and configured Android SDK, initial Android Virtual Devices, Node.js, Git client...
- now ready to install and setup the **Cordova CLI** itself using **NPM**
- *OS X*

```
npm install -g cordova
```

- *Windows*

```
C:\>npm install -g cordova
```

- –g flag tells NPM to set package, Cordova as global
- default NPM behaviour is to install the package in working directory
- if necessary, update system's **PATH**
- *OS X (Unix)*

```
/usr/local/share/npm
```

- *Windows*

```
C:\Users\username\AppData\Roaming\npm
```

- test Cordova install, e.g. version 7.0.1

```
cordova -v
```

Cordova App - iOS Cross-platform

- start by installing the latest version of Xcode
  - *available from Mac App Store*
  - *follow install prompts*
  - *install additional components*
- check Xcode command line tools are installed

```
xcode-select --install
```

- then update required packages for Cordova and iOS development
  - *update `ios-deploy` if needed for the local system*

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

- update `ios-sim` if needed for the local system

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

- then follow Cordova patterns for creating a new app

Cordova App - development paths - intro

Since the advent of version 3.0 of Apache Cordova, it's been possible to develop apps using two distinct workflows and paths. These workflows include,

- Cross-platform CLI
- Platform-centric

Cordova App - development paths - cross-platform CLI workflow

- CLI allows us to develop cross-platform apps
- offers a broad, wide-ranging collection of mobile OSs
  - *support for many native devices*
- workflow focused upon the `cordova` utility
  - *become known as the Cordova CLI*
- CLI is considered a high-level tool
  - *designed to abstract cross-platform development considerations*
  - *designed to abstract functionality of lower-level shell scripts*

e.g.

```
cordova platform add android --save
```

Cordova App - development paths - platform-centric workflow

- *platform-centric* focuses development on one native platform
  - *choose required native SDK - Android, iOS...*
- augment Cordova app with native components developed in the SDK
- workflow has been tailored for each platform
- relies on a set of lower level shell scripts
- features a custom plugin utility designed to help us apply plugins

Cordova App - development paths - careful choice

- start Cordova development using the *cross-platform* workflow
  - *using CLI*
- then option to migrate to the platform-centric workflow
  - *useful for lower-level, specific OS targeted apps*
- one way from CLI to *platform-centric*
  - *CLI keeps a common set of cross-platform source code*
  - *uses this code to ensure broad support and compliance per build*
  - *overrides any platform-specific modifications and code*
- lower-level specific apps should use *platform-centric* shell tools

Cordova App - creating a new project

- basic outline for creating a template for our project is as follows

```
cordova create basic com.example.basic 422Basic
cd basic
cordova platform add android --save
cordova build
```

- then launch this new Cordova project application, e.g. in the emulator

```
cordova emulate android
```

Cordova App - anatomy of a template - part I

```
cordova create basic com.example.basic 422Basic
```

- first parameter of this represents the path of our project
- creating a new directory in the current working directory called **basic**
- second and third parameters are initially optional
- helps to define at least the third parameter
  - *the visible name of the project,* `422Basic`
- edit either of the last two parameters in the projects `config.xml` file
- at the root level of our newly created project

Cordova App - anatomy of a template - part 2

- new project includes the following default structure
- default parts for our development...

```
config.xml
hooks
  - README.md
platforms
  - android
  - platforms.json
plugins
  - android.json
  - cordova-plugin-whitelist
  - fetch.json
res
  - icon
  - screen
www
  - css
  - img
  - index.html
  - js
```

- initially, our main focus will be the www directory

Cordova App - anatomy of a template - part 3

- the `www` directory will be the initial primary focus
- three primary child directories
  - *css*
  - *img*
  - *js*
- important `index.html` file at the root level
- three primary files, which include
  - *`index.css`*
  - *`index.js`*
  - *`logo.png`*
- `config.xml` file stores configuration settings for the application

Cordova App - anatomy of a template - part 4

- three important directories to help manipulate and configure our Cordova application
  - *platforms*
  - *plugins*
  - *hooks*
- `platforms` includes an application's currently supported native platforms
  - *e.g. Android*
- `plugins` includes all of the application's used plugins
- `hooks` contains a set of scripts used to customise commands in Cordova
  - *customise scripts that execute before or after a Cordova command runs*

Cordova App - anatomy of a template - part 5

WWW directory

- three primary files initially help us develop Cordova application
  - *index.html*
  - *index.js*
  - *index.css*

Cordova App - anatomy of a template - part 6

- `index.html` - default template for new project

```
<body>
    <div class="app">
        <h1>Apache Cordova</h1>
        <div id="deviceready" class="blink">
            <p class="event listening">Connecting to Device</p>
            <p class="event received">Device is Ready</p>
        </div>
    </div>
    <script type="text/javascript" src="cordova.js"></script>
    <script type="text/javascript" src="js/index.js"></script>
</body>
```

Cordova App - anatomy of a template - part 7

- default `index.html` page very straightforward
- `<div class="app">` is the parent section, acts as the app's container
- contains a child `div`
  - *unique ID `deviceready`*
  - *two key paragraphs triggered relative to state changes in the app*
- app simply updates state relative to event being actioned and listened
- events are monitored and controlled using the app's initial JavaScript
- `initialize()` method calls `bindEvents()` method
  - *adds an event listener to this `deviceready` div*
- means when device is ready `event listening` paragraph will be hidden
- `event received paragraph` is now shown

Cordova App - anatomy of a template - part 8

- js/index.js

```
var app = {
    // Application Constructor
    initialize: function() {
        this.bindEvents();
    },
    // Bind Event Listeners
    //
    // Bind any events that are required on startup. Common events are:
    // 'load', 'deviceready', 'offline', and 'online'.
    bindEvents: function() {
        document.addEventListener('deviceready', this.onDeviceReady, fal
    },
    // deviceready Event Handler
    //
    // The scope of 'this' is the event. In order to call the 'receivedE
    // function, we must explicitly call 'app.receivedEvent(...);'
    onDeviceReady: function() {
        app.receivedEvent('deviceready');
    },
    // Update DOM on a Received Event
    receivedEvent: function(id) {
        var parentElement = document.getElementById(id);
        var listeningElement = parentElement.querySelector('.listening')
        var receivedElement = parentElement.querySelector('.received');

        listeningElement.setAttribute('style', 'display:none;');
        receivedElement.setAttribute('style', 'display:block;');

        console.log('Received Event: ' + id);
    }
};
```

Image - Cordova Splash Screen

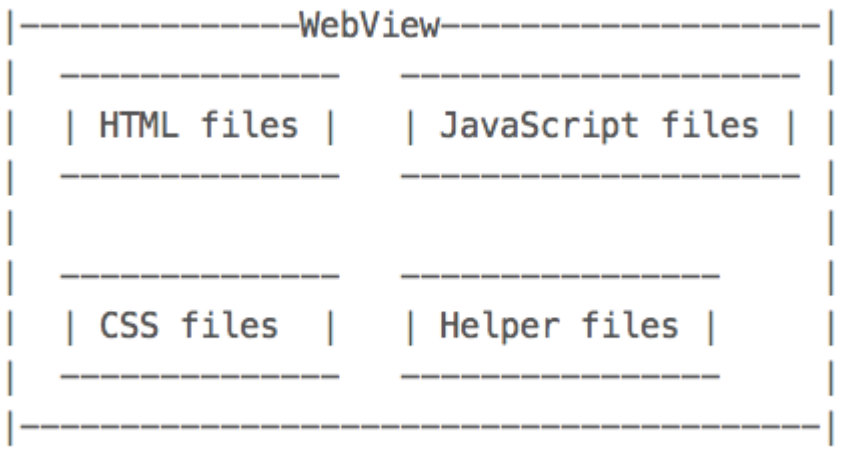
Apache Cordova Default Splashscreen

Apache Cordova - architecture - part I

- Cordova relies on web technologies at its core
- HTML5
- CSS
- JavaScript (JS)
- core architecture for app development using Cordova
- supplement this core with additional helper files
- e.g. JSON (JavaScript Object Notation) resource files
- to enable access to a device's native functionality
- JS application objects (or functions) call Cordova APIs
- Cordova APIs for different native mobile OSs, e.g.
  - *use Cordova Android for native Android functionality...*
  - *use Cordova iOS for native iOS...*
- develop our own custom plugins as necessary

Image of Apache Cordova architecture

The following diagram summarises the core architecture for Cordova application development.



Source - Apache Cordova

Apache Cordova - architecture - part 2

- core architecture creates a single screen in the native app
- single screen contains a **WebView**
- uses all of the device's available screen space (real estate)
- native WebView used to enable loading app's HTML, CSS, JS...
- WebView is a native view in each mobile OS
- allows us to display HTML based content
- allows us to leverage power and functionality of a mobile browser
- working within a contained native app

Apache Cordova - webview - part I

- using this WebView in our app
- Cordova loads the app's default startup page
  - *in essence its `index.html` page*
- passes control of the app to the native WebView
- allows user to control the app as normal
- user can interact with app in native manner
- user gets a native app experience
- user interaction can include the vast majority of standard native interaction patterns and options
- user is not aware of difference between Cordova or native developed app

Apache Cordova - webview - part 2

- WebView has an implementation in all of the major mobile OSs
- Android has a class called

```
android.webkit.WebView
```

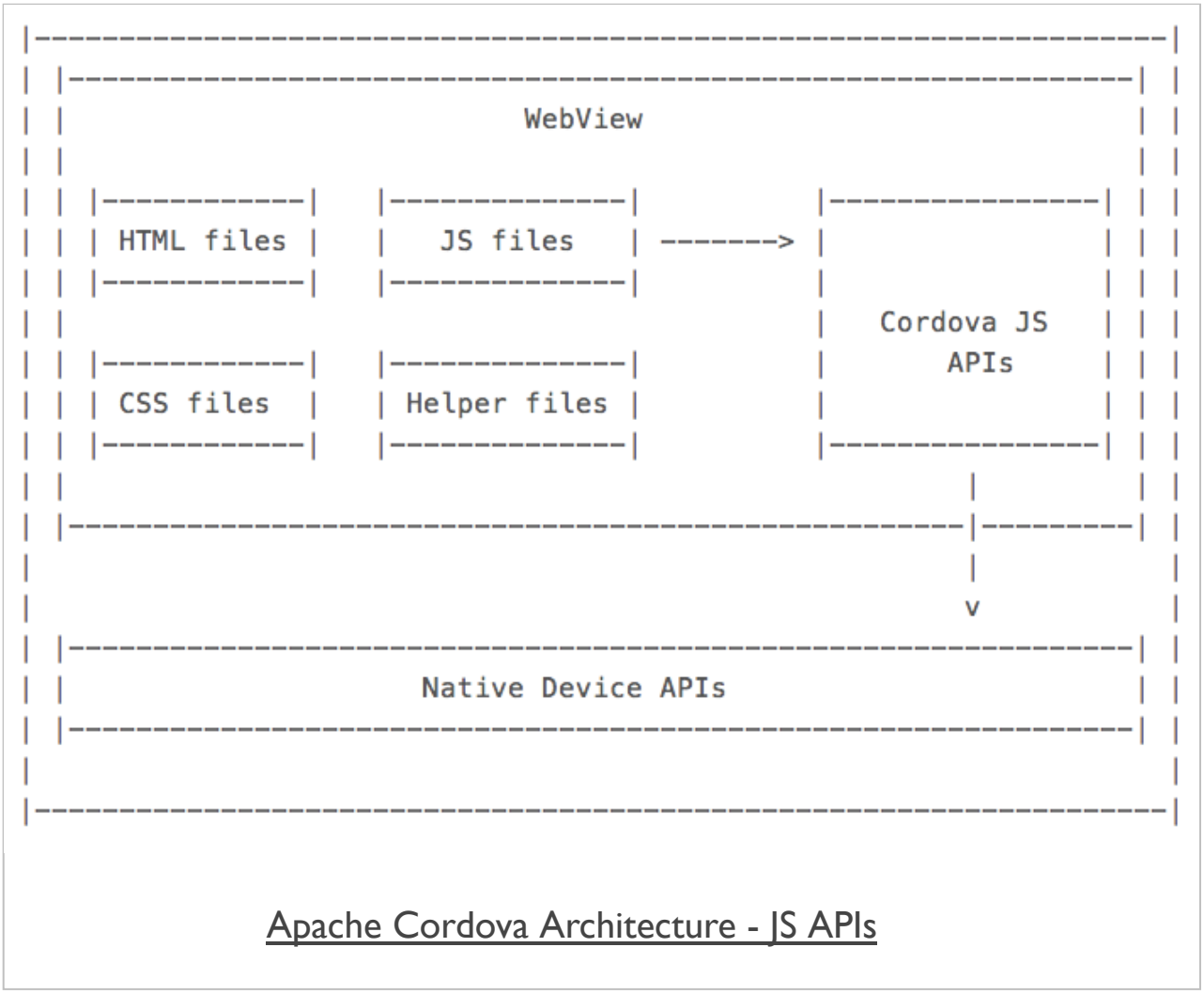
- iOs references the `UIWebView`
- part of the `UIKit` framework
- Window Phone refers to a WebView class called

```
Windows.UI.Xaml.Controls
```

Apache Cordova - native functionality - part I

- provides access to many types of native functionality, including
  - *sound and audio*
  - *recording*
  - *camera capture*
  - *photo access*
  - *geolocation*
  - *sensors...*
- Cordova leverages JavaScript APIs to provide native functionality

Image - Apache Cordova Native Functionality



Apache Cordova Architecture - JS APIs

Source - Apache Cordova

Apache Cordova - native functionality - part 2

- architecture is an elegant approach to solving cross-platform issues
- allows developers to leverage unified API interface
  - *perform specific native functions*
  - *calls to native functionality transparent across platforms*
    - strength of using JavaScript APIs
- Cordova JavaScript APIs
  - *call the required native OS API*
  - *e.g. Cordova's Android or iOS API*
- plugins give Cordova its power and flexibility

Apache Cordova - example call - part I

If we want to get a picture from the camera, we call the following using Cordova

```
navigator.camera.getPicture(onSuccess, onFail, { quality: 75,
  destinationType: Camera.DestinationType.DATA_URL
});

function onSuccess(imageData) {
  var image = document.getElementById('Image');
  image.src = "" + imageData;
}

function onFail(message) {
  alert('Error: ' + message);
}
```

Apache Cordova - example call - part 2

- making a simple call to the method `getPicture()` of the `camera` object
- call is performed with 3 parameters
- **onSuccess**
  - *callback allows us to tell the app what to do if the call and returned data is successful*
- **onFail**
  - *another callback tells the app how to handle an error or false return*
  - *e.g. an error is thrown, callback will handle output of a suitable error message*
- **quality**

Apache Cordova - example call - part 3

```
quality: 75, destinationType: Camera.DestinationType.DATA_URL
```

- slightly different as it contains a JS object with configuration parameters
- two parameters are for `quality` and `destinationType`
- `quality` can be from `0 to 100`
- `destinationType` refers to the required format for the returned data value
  - *can be set to one of 3 possible values*
  - *`DATA_URL` - format of the returned image will be a Base64 encoded string*
  - *`FILE_URL` - returns the image file URL*
  - *`NATIVE_URI` - refers to the images native URI*

Apache Cordova - example call - part 4

- if the return is a success we will get a Base64 encoded string
  - *string of the image just captured using the native camera*
- leveraging the power of the Apache Cordova camera plugin code, e.g. Android camera plugin
- power of the underlying Android class
  - *wrapped in a layer that we can call from our JavaScript code*
- plugin is written natively for Android
  - *we access it using JS with Cordova*
- plugins for other platforms follow the same pattern
  - *e.g. iOS camera plugin...*

Apache Cordova - example call - part 5

- we issue a call from JS using Cordova to the native code in the plugin
- plugin processes this request
  - *returns the appropriate value*
  - *either for a success or a failure*
- in our example, if request to the camera is successful
  - *Android plugin will return a string to the JS Cordova client, as requested*
- use similar pattern for other mobile OSs
  - *e.g. accessing a camera's functionality with iOS...*
  - *appropriate plugin required for necessary mobile OS*
  - *if not, we can write a custom plugin*

Apache Cordova - cross-platform power

- implement capturing a photo from device's native camera on multiple mobile platforms
- Cordova plugin architecture removes
  - *need to understand how the photo capture is implemented or handled natively*
- Cordova plugin handles the native calls
- Cordova plugin handles processing for each native device

Cordova - CLI - Useful commands

A few initial useful CLI commands

command	example	description
cordova	cordova	general command - outputs overview with 5 categories of information and help
-v	cordova -v	check current installed version of cordova
requirements	cordova requirements	check requirements for each installed platform
create	cordova create basic com.example.basic 422Basic	creates new project with additional arguments for directory name, domain-style identifier, and the app's display title
platform add	cordova platform add android --save	specify target platforms, eg: Android, iOS... (NB: SDK support required on local machine)
platform ls	cordova platform ls	checks current platforms for cordova development on local machine and lists those available
platform remove (platform rm)	cordova platform rm android	remove an existing platform
build	cordova build	iteratively builds the project for the available platforms
build ios	cordova build ios	limit scope of build to a specific platform (useful for testing a single platform...)
prepare	cordova prepare ios	prepare a project, and then open and build &c. with native IDE (eg: XCode, Android Studio...)

command	example	description
compile	cordova compile ios	compile ios specific version of app
emulate	cordova emulate android	rebuilds an app and then launches it in a specific platform's emulator
run	cordova run android	run an app on a native device connected to the local machine
run --list	cordova run --list	check available emulators, e.g. Android AVDs

- more commands will be added as we work with Cordova, NPM...

Course resources - Cordova install and setup

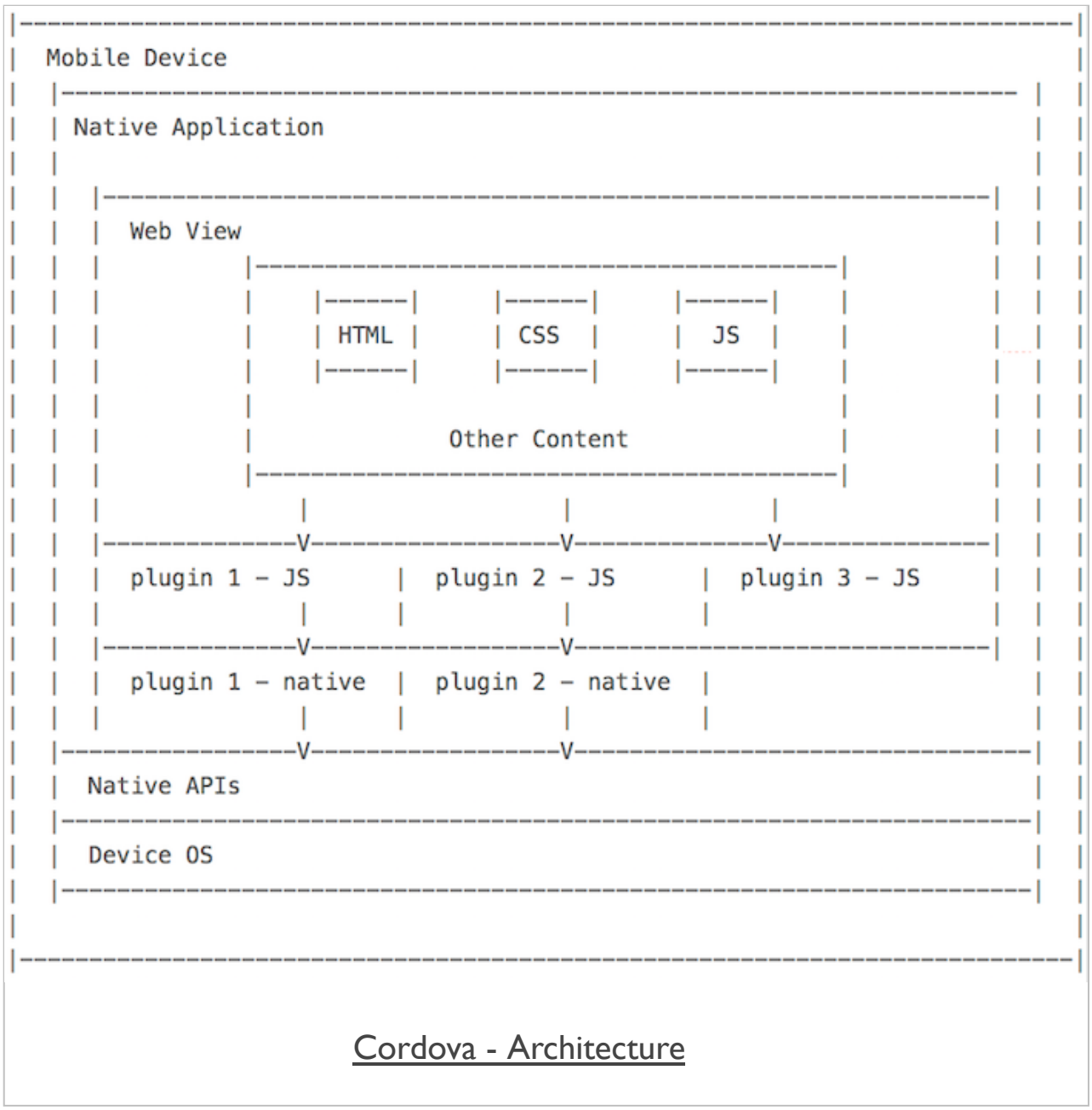
Latest course resources for install and setup of Cordova include,

- Cordova Install and Setup
- Android Platform Guide
- iOS Platform Guide

Cordova Design - architecture - intro

- quickly recap the architecture and design behind a Cordova Native application
- Cordova effectively consists of the following components
  - *source code to allow us to build a native application container*
  - *specific to the mobile platforms we choose to add to our project, eg: Android, iOS...*
  - *a collection of various APIs, implemented by Cordova as plugins*
  - *web application running within the container*
  - *access to native device functionality, APIs, and applications*
  - *provides a useful set of tools that help us manage our projects*
  - *creating a project, project files...*
  - *manage required plugins*
  - *build native applications using the native SDK*
  - *testing of applications using emulators, simulators...*

Cordova Design - architecture - diagram



Cordova - Architecture

Cordova Design - architecture

JS & Web plugins

- outline architecture includes the option for JavaScript only plugins
- JS plugins in Cordova normally a bridge from our web container to the native APIs
  - *useful way to expose native device functionality to the web application*
- use and develop plugins purely in JS
  - *add an existing library to help with data visualisations, graphics...*
- create our own focused plugins
  - *abstraction of application features and logic, other specific requirements...*
- greater support for native functionality at the web application level
- HTML5 APIs

Cordova Design - architecture - web container - part 1

- Cordova development
  - *uses many of the same underlying technologies as standard web application development*
  - *a few limitations relative to network access that we need to consider*
- hybrid mobile application with Cordova
  - *a web application needs to be written as a self-contained application*
  - *needs to be able to run within web container on native device*
  - *constantly fetching external resources not good practice*
  - *mix of local and remote resources preferable for most apps*
  - *external resources an issue if we lose a network connection*
- `index.html` file will normally be the only HTML file we use
  - *separate pages will be containers within this file*

Cordova Design - architecture - web container - part 2

- rethink our approach to building such mobile web stack applications
  - *help us leverage the inherent capabilities of Cordova*
- self-contained applications need to ensure
  - *any application files and data are initially available*
  - *allows the application to launch and load on the native device*
  - *without initial calls to a remote server*
  - *load the application and render the UI*
- application can then optionally fetch data
  - *remote server, API, search query, stream media...*
- consider stages of design for our app's container

Cordova Design - architecture - SDKs and OSs

- build our Cordova applications
  - *including default Cordova APIs or additional APIs*
  - *each app has to be packaged into a native application*
  - *allows app to run on the host native device*
- each native SDK has its own set of custom or proprietary tools
  - *building and packaging their specific native applications*
- build our Cordova applications for a native device
  - *web content portion of app is added to a project*
  - *applicable to the chosen mobile platforms,*
  - *e.g. Android, iOS, Windows 10 Universal Platform...*
  - *project is then built for each required platform*
  - *using Cordova CLI, for example*
  - *uses each of the applicable platform specific set of tools to help build*

References

- Android Platform Guide
- Android - Stand-alone SDK Tools (see foot of page)
- Android Studio 2.1
- Apache Cordova
- Apache Cordova Documentation & Guide
- Git
- Java - JDK 8
- jQuery Mobile API
  - *Pagecontainer widget*
- jQuery Mobile 1.4 Browser Support
- MDN - default event
- Node.js