

Comp 388/422 - Software Development for Wireless and Mobile Devices

Fall Semester 2015 - Week 3

Dr Nick Hayward

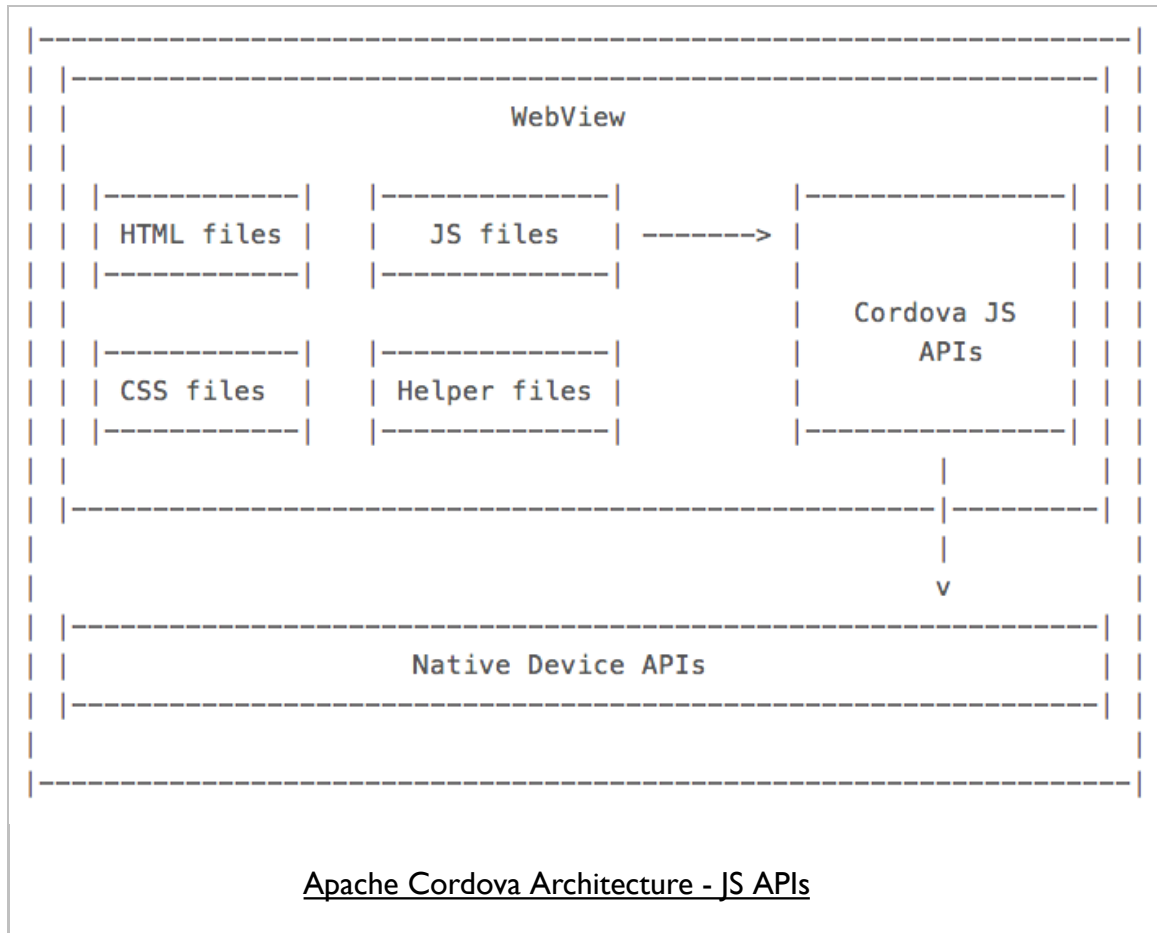
Today's class

- Cordova native functionality
- Example call
- Overview of APIs
- Cordova App intro
- Development paths
- Platform-centric workflow
- Android SDK setup
 - *Other tools and setup*
 - *Android SDK tools*
 - *Device definitions*
 - *emulator*
- Cordova app project
- Setup your own environment...

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



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*
 - *eg: 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 = "data:image/jpeg;base64," + 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*
 - *eg: 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`
 - `FILE_URL`
 - `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 Android Camera plugin code
- 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*

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
 - *eg: accessing a camera's functionality with Windows Phone*
 - *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 architectures 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

Apache Cordova - overview of APIs

Platform APIs	Android	iOS	Windows Phone...
Accelerometer	Yes	Yes	Yes
BatteryStatus	Yes	Yes	Yes
Camera	Yes	Yes	Yes
Capture	Yes	Yes	Yes
Compass	Yes	Yes	Yes
Connection	Yes	Yes	Yes
Contacts	Yes	Yes	Yes
Device	Yes	Yes	Yes
Events	Yes	Yes	Yes
File	Yes	Yes	Yes
File Transfer	Yes	Yes	Yes (no support for onprogress or abort)
Geolocation	Yes	Yes	Yes
Globalisation	Yes	Yes	Yes
InAppBrowser	Yes	Yes	Yes
Media	Yes	Yes	Yes
Notification	Yes	Yes	Yes
Splashscreen	Yes	Yes	Yes
Storage	Yes	Yes	Yes (localStorage & indexedDB)
Vibration	Yes	Yes	Yes

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*
 - *eg: 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](#)

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 - development paths - part I

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 - 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*

Cordova App - 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 - part 2

- 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 - 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 mobile OSs such as Android, iOS, and Windows Phone/Mobile

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**
- use **Android Studio** to help build custom Android plugins for Cordova
- **Android Stand-Alone SDK Tools** sufficient to build and deploy an Android app
 - *we'll start with **stand-alone** tools*
 - *migrate to **Android Studio** later for plugin development*

Cordova App - Android SDK Tools

- we're going to use **Android Stand-Alone SDK Tools** for initial dev work
- options for installing Android SDK tools,
 - *Android - Installing the Stand-alone SDK Tools*
- stand-alone tools include a set of command line tools and helpful GUIs
- choose SDK for your preferred desktop OS,
 - *Android - SDK Tools Only*
- download the required package
- move the extracted directory to a preferred location on your local machine
- we'll return to **Android Studio** later (install instructions include)
 - *Android Studio*

Cordova App - setup and configure 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}/tools:${ANDROID_HOME}/platform-tools
```

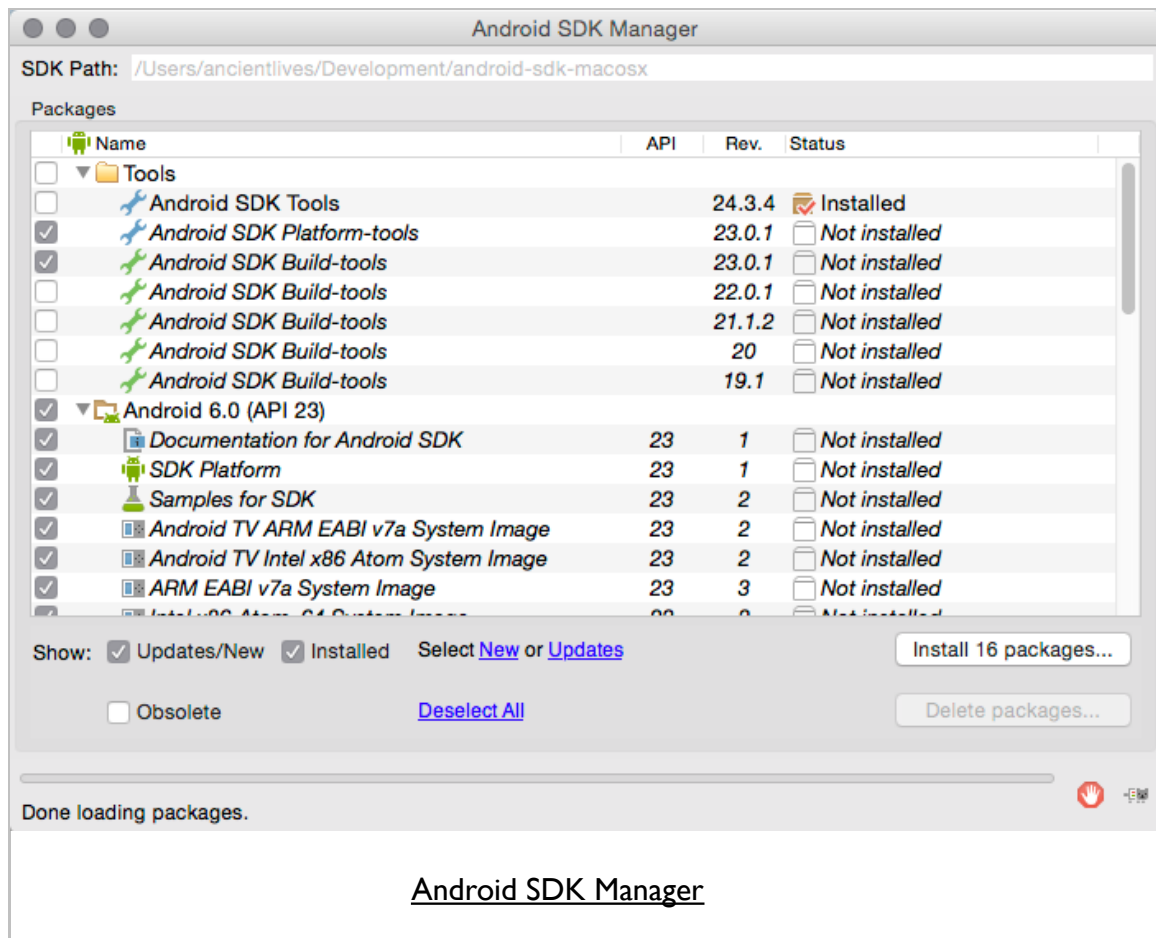
- Windows

```
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
```

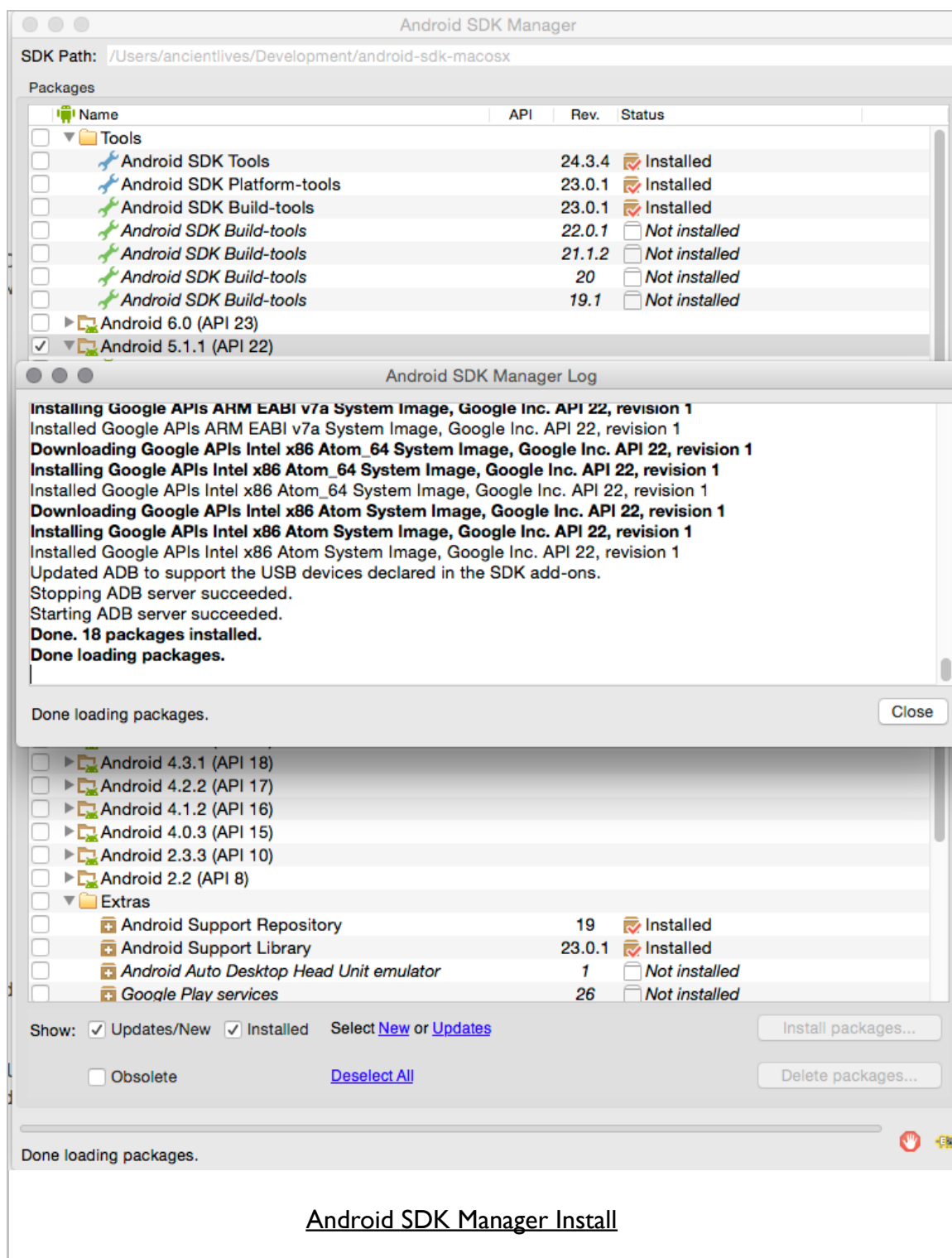
Image - Android SDK Manager



Cordova App - Android SDK extras

- initially, we need to add the following extra SDK packages for Cordova,
 - *Android 5.1.1 (API 22) platform SDK*
 - *Android SDK Build-tools (latest version)*
 - *Android Support Repository (listed in the Extras section)*
- downloading and installing many different APIs for Android images
 - *eg: Android Wear, Android TV, standard Android...*
- 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...*
- can take a few minutes or so to complete

Image - Android SDK Manager Install



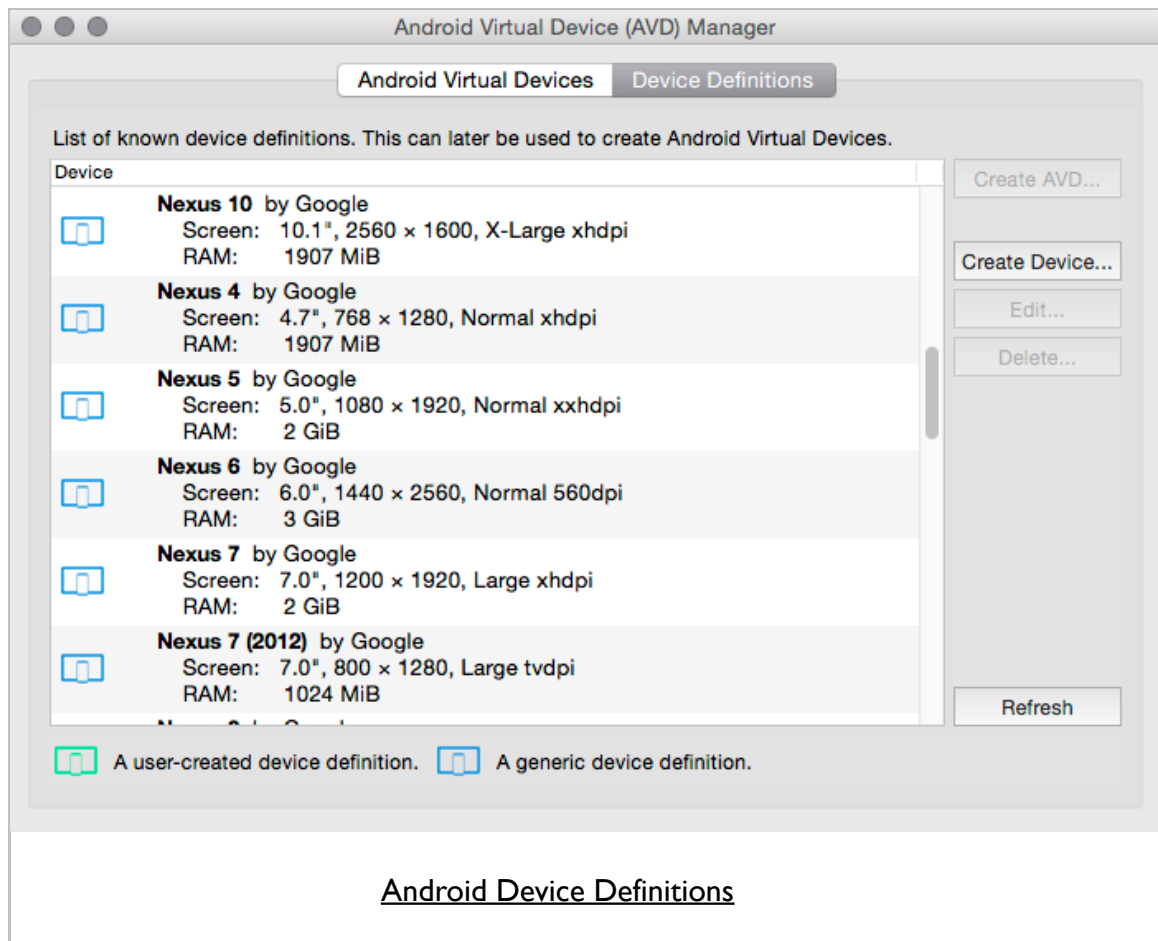
Cordova App - Android emulator - part I

- close **Android package manager** to ensure that portion of setup is complete
- configure an emulator for testing and development of our applications
- Android SDK does not provide an existing instance of an emulator
- need to create a new one using the SDK tools
- open these tools from the command line using the command,

```
android
```

- select the **Tools** menu and the menu item **Manage AVDs**
 - *AVDs = Android Virtual Devices*
- choose an item from the list of **Device Definitions**

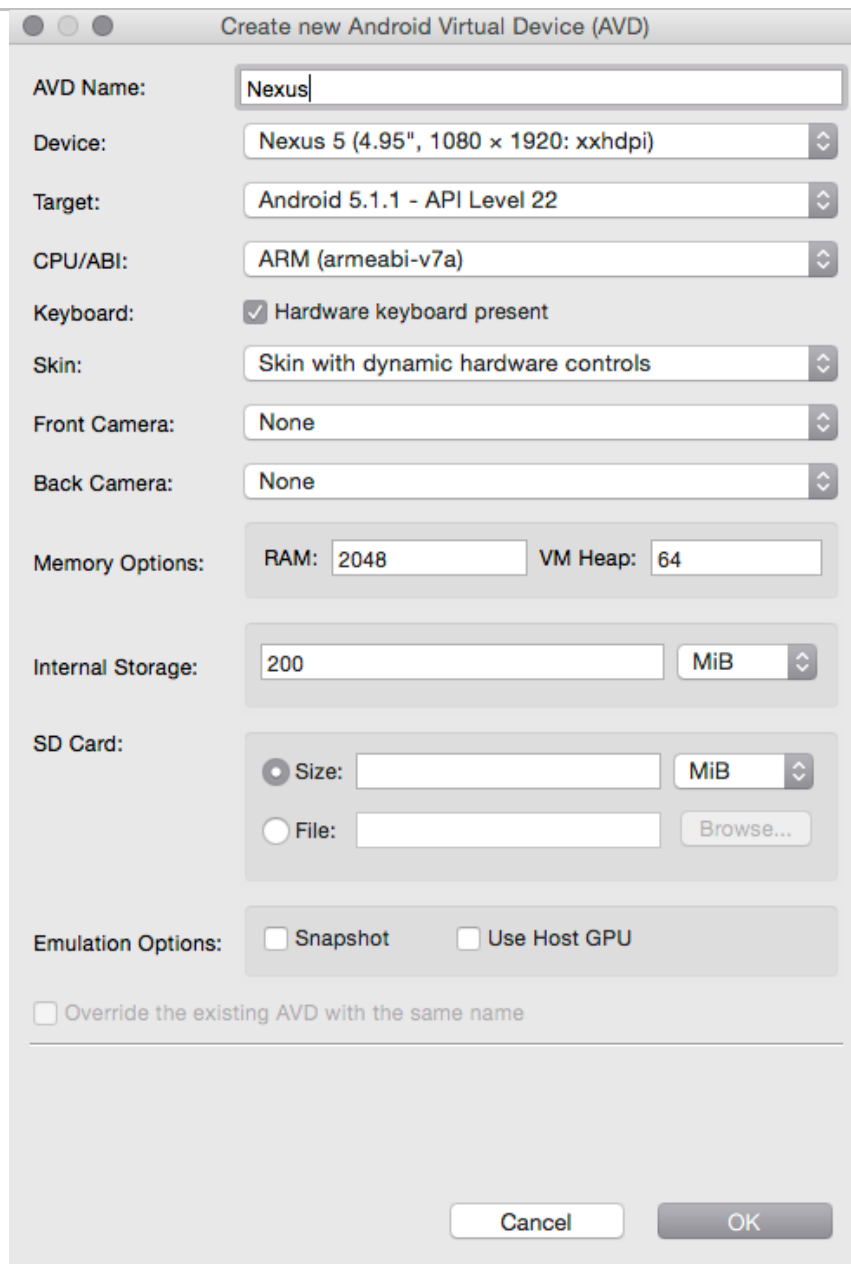
Image - Android Device Definitions



Cordova App - Android emulator - part 2

- many different device definitions
 - eg: *Nexus devices (5, 6, 7...), generic definitions...*
- pick an average device, such as the Nexus 5 or 6
 - *we can others later*
- select your preferred *Device Definition* from the available list
 - eg: *Nexus 5*
- then **Create AVD**
- modify a few settings to help device creation
 - *optionally customise the **AVD Name***
 - *set the target to **Android 5.1.1 - API Level 22***
 - *set the CPU/ABI to **ARM (armeabi-v7a)***
 - *set skin to **Skin with dynamic hardware controls***

Image - Android Virtual Device



Create new Android Virtual Device (AVD)

AVD Name:

Device:

Target:

CPU/ABI:

Keyboard: ☒ Hardware keyboard present

Skin:

Front Camera:

Back Camera:

Memory Options: RAM: VM Heap:

Internal Storage:

SD Card:

☒ Size:

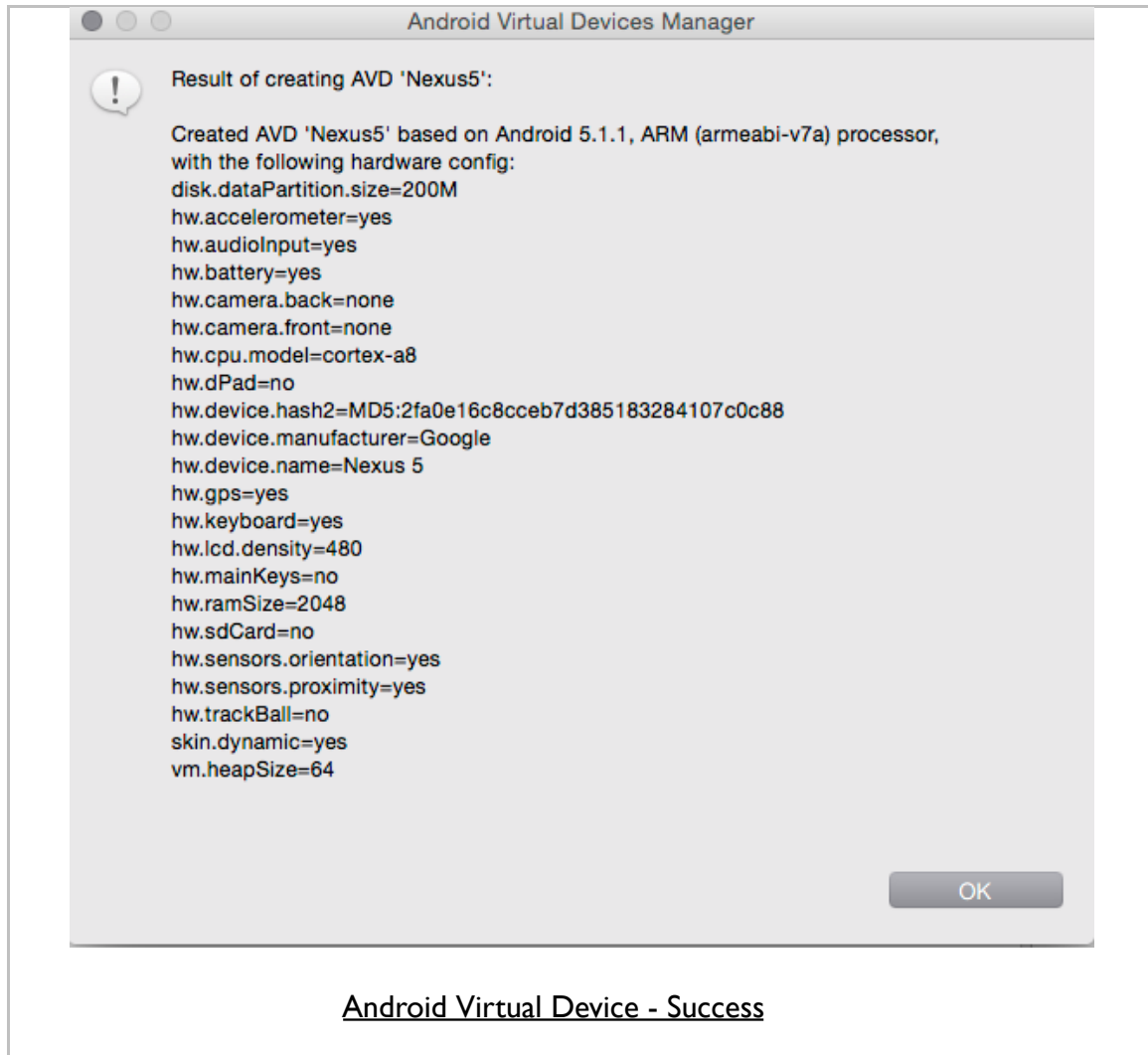
☐ File:

Emulation Options: ☐ Snapshot ☐ Use Host GPU

☐ Override the existing AVD with the same name

Android Virtual Device

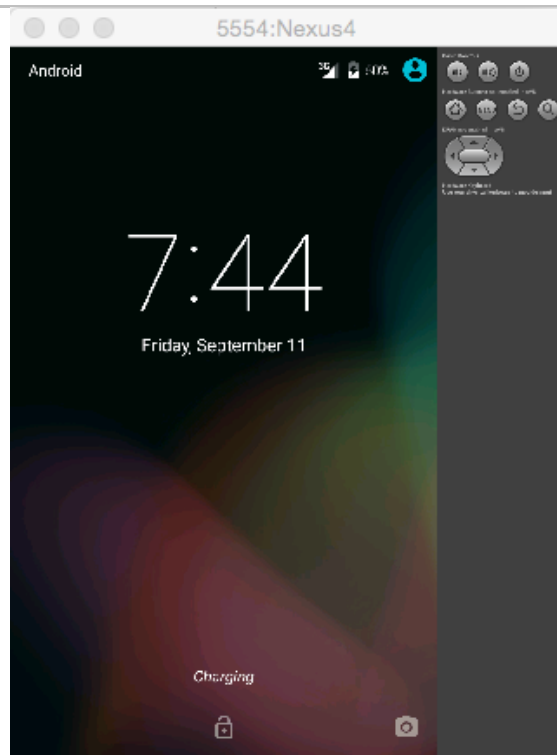
Image - Android Virtual Device - Success



Cordova App - Android emulator - part 3

- new **Android Virtual Device**, Nexus 4, 5, 6...
- test in its default state
- select required device from list in the **Android Virtual Device Manager**
 - click **Start** to load emulator
- virtual device starts same as a standard device
 - including hardware buttons in pane on right hand side

Image - Android Virtual Nexus Device



Android Virtual Device - Nexus 4

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*

```
sudo 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
```

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
cordova build
```

- then launch this new Cordova project application, eg: 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

```
config.xml
hooks
  - README.md
platforms
  - android
  - platforms.json
plugins
  - android.json
  - cordova-plugin-whitelist
  - fetch.json
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
 - eg: *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

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

Cordova App - anatomy of a template - part 6

```
<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`, `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

Image - Cordova Splash Screen



Apache Cordova Default Splashscreen

References

- [Android Platform Guide](#)
- [Android - SDK Tools Only](#)
- [Android - Installing the Stand-alone SDK Tools](#)
- [Android Studio](#)
- [Apache Cordova](#)
- [Apache Cordova Documentation](#)
- [Cordova Platform Guides](#)
- [Git](#)
- [Java - JDK](#)
- [Node.js](#)