**Creating a native Android application**

In this lab you are going to learn how to create a native Android application from an existing Javascript project. We are going to use a tool called [Apache Cordova](https://cordova.apache.org/).

Cordova is formerly known as PhoneGap and is a platform to build native mobile applications using HTML5, CSS and Javascript.

This lab is based on [this](https://zolmeister.com/2014/01/how-to-turn-webapp-into-native-android.html) original short guide.

**1. Setup**

We need to download several SDKs and tools before we can do anything. This process will take some time, so you will need to be patient!

1.1 Download the Windows 64-bit version of the Java Development Kit (JDK), currently at version 8 (note: also referred to as Java Platform, Standard Edition 8 Update 144). You can get it from [here](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html).

1.2 Download the Android Software Development Kit (SDK) tools. This is a set of development tools used to create applications for the Android platform. You can download the SDK from [here](https://developer.android.com/studio/index.html) - scroll to the bottom of the page to download (you do not require Android Studio).

1.3 Set up your Windows environment variables (if you have never done this, see [here](https://www.computerhope.com/issues/ch000549.htm) first). Create a new environment variable for ANDROID\_HOME:

ANDROID\_HOME=C:\<installation location>\sdk-tools-windows-3859397

(Note the suffix 3859397 is simply the version number of the SDK tools I am using).

Add ANDROID\_HOME to your system PATH variable (include the % symbols)

%ANDROID\_HOME%

You can also use the SET command to set environment variables in the current window:

set ANDROID\_HOME=C:\<installation location>\sdk-tools-windows-3859397

set PATH=%PATH%;%ANDROID\_HOME%

1.4 Download the Android Build Tools. These tools are used to debug, build, run and test an Android application.The Android SDK tools include a command line tool called [sdkmanager](https://developer.android.com/studio/command-line/sdkmanager.html). This utility is located in your sdk-tools-windows-3859397\tools\bin folder.

sdkmanager is used to install the Android Build Tools, e.g. to install build tools version 26.0.1, we give the command:

sdkmanager “build-tools;26.0.1”

Be patient as the download may take a while.

While you are waiting, if you are wondering what the version number 26.0.1 means, have a read of [this](https://source.android.com/source/build-numbers) guide.

1.5 Download Gradle - get the *binary only* zip version from [here](https://gradle.org/install/#manually).

Gradle is an open source build system. In an Android development environment, Gradle is a custom build tool used to build android packages (known as Android Application Package or apk files) by managing dependencies and providing custom build logic. The APK file is a specially formatted zip file which contains the program code, resources, assets, certificates, and [manifest file](https://developer.android.com/guide/topics/manifest/manifest-intro.html).

1.6 Add the gradle bin directory to your system path, e.g.

set PATH=%PATH%;C:\gradle-4.1\bin

Now you are ready to use cordova. We will start with a simple hello world type demo. [Enable USB debugging](https://www.kingoapp.com/root-tutorials/how-to-enable-usb-debugging-mode-on-android.htm) on your Android device. In Windows, create a folder called helloWorldApp. Now open a command prompt in this folder and then follow the steps below.

*# (Assuming you are now in your project folder)*  
*# create a mobileapp folder for your app*  
**cordova create mobileapp ie.itcarlow.games3 HelloWorld  
cd mobileapp***# add android*  
**cordova platform add android**  
*# compile the app, be patient this may take a couple of minutes the first time.*  
**cordova build**  
*# if the build was successful, you will get a message showing the output directory of # your apk file. Now, plug in your testing device (your phone/tablet), and let's run # our test app on it*  
**cordova run android**

When the app is launched, watch your device for any prompts to allow usb debugging. The output from this simple app is: **Apache Cordova, device is ready**

**Tech note:** To give a little more detail, as part of the above build process, a folder called **www** is generated inside your **helloWorldApp** folder like this:

helloWorldApp\mobileapp\www

This contains the html and javascript code that we wish to execute on the device. When you issue the cordova build command, the contents of this **www** folder are copied to:

helloWorldApp\mobileapp\platforms\android\assets\www

The build process also copies some cordova script code to this location (cordova.js and cordova\_plugins.js). The files in this location are then bundled up into an apk file and loaded onto the Android device. We will learn a little more about this process below.

**2. Setting Portrait or landscape mode**

You can edit a configuration to lock your application to run in either portrait or landscape mode as follows. Locate the config.xml file in your mobileapp folder:

mobileapp/config.xml  
  
Anywhere inside the <widget> tag add the following line to lock your application to landscape mode:

<preference name="Orientation" value="landscape" />

For example, here is a snippet from my config.xml

<widget id="ie.itcarlow.games3" version="1.0.0" xmlns="http://www.w3.org/ns/widgets" xmlns:cdv="<http://cordova.apache.org/ns/1.0>">

...

**<preference name="Orientation" value="landscape" />**

<engine name="android" spec="^6.2.3" />

<plugin name="cordova-plugin-whitelist" spec="^1.3.2" />

</widget>

See [here](https://cordova.apache.org/docs/en/4.0.0/config_ref/index.html) for more information about config.xml

**3. Managing your screen resolution**

As you will be targeting your game for a specific android device, you need to think about your device’s screen resolution. The viewport is a rectangular area that provides a drawable region for your webpage. The viewport has its own pixel dimensions that it makes available to a web page. For example, a device might have a screen width of 480 pixels while the viewport has a width of 800 pixels. A web page designed for 800 pixels can be completely visible on the device if the viewport is zoomed all the way out, or left at a scale factor of 1. A very simple approach you can take is to specify that the viewport width/height matches the device’s screen width/height and the ability to zoom is disabled. This is done by adding a meta tag to your web page:

<head>

<title>Example</title>

<meta name="viewport" content="width=device-width, height=device-height,

user-scalable=no" />

</head>

See [here](https://developer.android.com/guide/webapps/targeting.html) for more information.

**Note 1:** If everything appears too large on your device, try adjusting the scale setting. The tag below tells the device to use its native dimensions, but then adds a scale factor that tells it to not zoom in as much as it would normally:

<meta name="viewport" content="user-scalable=no, initial-scale=0.5, maximum-scale=1, width=device-width, height=device-height" />

**Note 2:** Scaling appears to be buggy with the current release of Cordova (<https://forums.adobe.com/thread/2223009>), but there is a work around that does not require using ctx.scale (where ctx is a Javascript context).

Inside your Cordova project in this location:

*mobileapp\platforms\android\src\ie\itcarlow\games3*

you will find a Java file called MainActivity.java which Cordova runs on the device. If you append these lines to the end of the onCreate() method:

if (appView == null) {

init();

}

((WebView) appView.getView()).getSettings().setUseWideViewPort(true);

((WebView) appView.getView()).setInitialScale(70);

appView.loadUrlIntoView(launchUrl, true);

// loadUrl(launchUrl); **// NB, don't forget to comment this line out.**

In this example, the scaling is set to 70%, i.e.

((WebView) appView.getView()).setInitialScale(70);

but you can use whatever value works best.

Make sure you comment out the existing call to loadUrl(launchURL).

Finally, add this import statement to the top of the class (below the other import statements):

import android.webkit.WebView;

**4. Adding content**

Finally, we can add our Javascript project, as a website. From your helloWorldApp folder, open your www folder:

helloWorldApp\mobileapp\www

Locate the solution to your very first lab - the one with the random squares drawn. If for example your solution has the following names lab1.html, lab1.css and lab1.js, then copy these files to the following locations:

helloWorldApp\mobileapp\www\js\lab1.js

helloWorldApp\mobileapp\www\css\lab1.css

helloWorldApp\mobileapp\www\lab1.html

Edit your lab1.html so it has the correct path to both the css and js files, e.g.:

// inside lab1.html

<link rel="stylesheet" href="css/empty.css" />

<script src="js/lab1.js"></script>

Next, open the file **index.html** which should be in the root of the www directory and replace its contents with the below (the important points are explained next):

<!doctype html>

<html lang="en">

<head>

<title>Lab 1 Android Demo</title>

<script type="text/javascript" charset="utf-8" src="cordova.js"></script>

<script>

document.addEventListener("deviceready", onDeviceReady, false);

function onDeviceReady() {

navigator.splashscreen.hide();

if (navigator.network.connection.type == Connection.NONE) {

networkError()

} else {

loadApp()

}

}

function loadApp() {

navigator.app.loadUrl("file:///android\_asset/www/lab1.html");

}

function networkError() {

navigator.notification.alert('Zoggle requires an internet connection')

var $net = document.createElement('div')

$net.innerHTML = 'Zoggle requires an internet connection'

document.body.appendChild($net)

}

</script>

<style>

body {

padding: 15px;

background: #23252e;

color: #01ced3;

text-align: center;

}

div {

font-size: 20px;

}

</style>

</head>

<body>

</body>

</html>

The onDeviceReady() function is a callback that fires when the Android device has loaded the file index.html. How is this file loaded? Recall the build process generates an apk file which contains everything in this folder:

helloWorldApp\mobileapp\platforms\android\assets\www

The apk file is loaded onto the device and cordova ensures the file called index.html is loaded first (from the apk).

Going back to the onDeviceReady() function, if no network connection is detected on your device, then it aborts - please note that network connection here means Wifi connection - a mobile data connection will not be recognised (**in which case you should comment out this code and just call loadApp()**).

The function loadApp() then loads your lab1.html file from the local file system on Android.

function loadApp() {

navigator.app.loadUrl("file:///android\_asset/www/lab1.html");

}

On the Android device the path:

android\_asset/www/ equates to helloWorldApp\mobileapp\platforms\android\assets\www

Once lab1.html loads, the associated javascript file (lab1.js) executes.

Now you are ready to test your application on the device - you just need to issue the build (cordova build) and deploy (cordova run android) commands.

Finally, note that you are running a debug version of the apk. If you want to publish your apk on the google play store, there are further steps you must follow. This includes building a release apk, signing your apk, setting up a google play developer account, creating suitable icons etc.

You can read more about this process [here](https://zolmeister.com/2014/01/how-to-turn-webapp-into-native-android.html) (steps 3 and 4).

Some final notes:

1) Keep in mind that even if your Javascript files are kept in this location on the Android device:

helloWorldApp\mobileapp\www\js\lab1.js

They still execute from the root www folder, i.e.

helloWorldApp\mobileapp\www

2) If you have assets such as image or audio files, I suggest you keep them all in lowercase and use underscores instead of camel casing. Android is based on the Linux kernel and is case sensitive to filenames.

3) Logcat is a command-line tool that dumps a log of system messages, including stack traces when the device throws an error and messages that you have written from your Javascript application with console.log(). Logcat is accessed by running the adb (Android Debug Bridge) command. Adb.exe is located inside the sdk-tools-windows folder, e.g.

**Sdk-tools-windows-3859397\platform-tools\adb**

Some useful adb commands:

adb devices // Show what android devices are currently connected to this PC

adb logcat -c // Clear logcat - good idea to do this before executing your app

adb logcat > log.txt // Copy logcat file from the Android device to current folder on Windows PC

If you have console.log messages in your Javascript code, you can search log.txt for the output from these messages.