How to build an Android .apk bundle using openFrameworks and Xcode

Notes on the meeting with Smilen Dimitrov

Group 3
Audio Augmented Reality
Aalborg Universitet
A.C. Meyers 15
2450, Copenhagen

***Abstract*—The following depicts the main steps and command in order to build an application and test it.**

I. Introduction

First of all, we should be aware of all the header files that we need in our project. Every time Xcode build fails and returns some error such as: `*.h: file not found` we have to take care adding them to the Header Search Path inside the Build Settings. It should be take into consideration not to use any special characters, and numbers as well, to name the project and its main directories (for more information here). In this particular case, it has been necessary to rename all the files containing numbers and that command has been prompted:

```
for ix in $(grep -rl 001getcoordinates .); do echo
    $ix; sed -i".bckp" "s/001getcoordinates/
    getcoordinatesA/g" $ix
```

II. Building .apk bundle

A. Before to start

`TARGET_ANDROID` must not be inserted as a preprocessor macro. The idea is to build a more general desktop application using Xcode and define between `ifDef` and `endif` all the specific functions for a mobile environment. Therefore, `TARGET_ANDROID` is invoked directly from the terminal when we prompt make `AndroidDebug`.

B. Main steps

1) : Once we built the specific Android project using the terminal command make `AndroidDebug`, it needs to copy some specific file in order to fill the Android project and let Android Studio build the correct `gradle` file.

2) : The so-copied files probably needs some manual modification such as the project name, the SDK involved etc.

3) : The project should be imported in Android Studio to make that IDE create for us all the files we need in order to build our app bundle using the terminal, i.e. outside that platform. Specifically, the most important files are *gradle*, *gradlew* and *local properties*. Then `./gradlew` command can be invoked in the terminal window.
NB: it could need to edit the local properties file to select the correct Android SDK path, otherwise the command invoked failed. It also could need to change permissions for `gradlew` if it?s not executable. To do so, it needs to prompt the command:

```
chmod +x gradlew.
```

4) : Those operations eventually make an APK, i.e. an application bundle that serves as installer in the Android environment. The application can be installed either on an emulator or on an Android mobile to be tested.

5) : When the `./gradlew` tasks command is launched on the terminal for the first time for the selected project, firstly gradle downloads itself inside the project directory - it happens once only for a project. Adding the `tasks` word to the `gradle` command let us see all the available command we might use (under the section *All tasks runnable from root project* in the terminal window). In particular, two of those are the most interesting: `assembleDebug` and `assembleRelease` - they generate the .apk bundle. NB: the compiler might fail and show this error:

```
No resources found.
```

If that happens, we should copy from an Android-based project (such as androidEmptyExample inside openFrameworks) all the subfolder inside the RES directory. Such a project like that in fact already contains all the files that gradle needs in order to compile for Android target. Check and compare all the files and directories and just add the missing ones.

6) : Once the compiled has built successfully, it could see a new file called yourProjectName-debug.apk. That bundle can then installed on whatever Android platform. However, a problem can occur: the `assembleDebug` command builds only for ARM processor while Android platform can take place on three different processor types. The `assembleRelease` command takes care of all the processor but present some problem though, e.g. it requires a signature to let the application be installed, otherwise the OS will refuse that. To avoid such problem, it needs to add to the config.make file this snippet of code:

```
ABIS_TO_COMPILE_DEBUG = armv7 neon x86.
```

III. Testing the app

Once the application successfully run on an Android platform we maybe want to test its effectiveness. To do so, there is a utility called *ADB* whose terminal command `logcat` ?dumps a log of system messages, which include things such as stack traces when the emulator throws an error and messages that you have written from your application by using the Log class (source here). The events are printed in real time so it is necessary to stop the call when the specific error is recognised - otherwise it continues to dump. Basically, it should be noticed something such as *START* {*something*} when the application is launched.
As example, in that case the application was launched:

```
START act=android.intent.action.MAIN cat=
[android.intent.category.LAUNCHER] flg=0x10200000
cmp=cc.openframeworks.getcoordinatesA/.OFActivity
bnds= [192,424][360,616] u=0 from pid 836
```

but unexpectedly quit because of this:

```
E/AndroidRuntime( 7569):
java.lang.SecurityException: Provider network
requires ACCESS_FINE_LOCATION or
ACCESS_COARSE_LOCATION permission.
```

Here is a list of the error we have encountered so far.

1) Log2 error: due to the absence of on Android. Fixed using this snippet of code:

```
double log2( double n )
   {
   // log(n)/log(2) is log2.
   return log( n ) / log( 2 );
   }
```

inside the ofApp.cpp file - not the header file, otherwise the compiler calls the function twice and an error would be encountered.

2) OfAndroid.init error: due to the absence in the project. Fixed copying that file from the AndroidEmptyExample. The log showed:

```
java.lang.UnsatisfiedLinkError:
Native method not found:
cc.openframeworks.OFAndroid.init:()V
```

3) Permission to access error: in this case, permissions were missing (as explained here). Fixed adding to *AndroidManifest.xml* these lines:

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION
</uses-permission>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION
</uses-permission>
```

in order to allow for GPS utilize.