Android Manual Setup

This tutorial demonstrates how to build a new Android project with Orx. It should work for all Android 2.0+ devices, though there may still be some issues with the emulators.

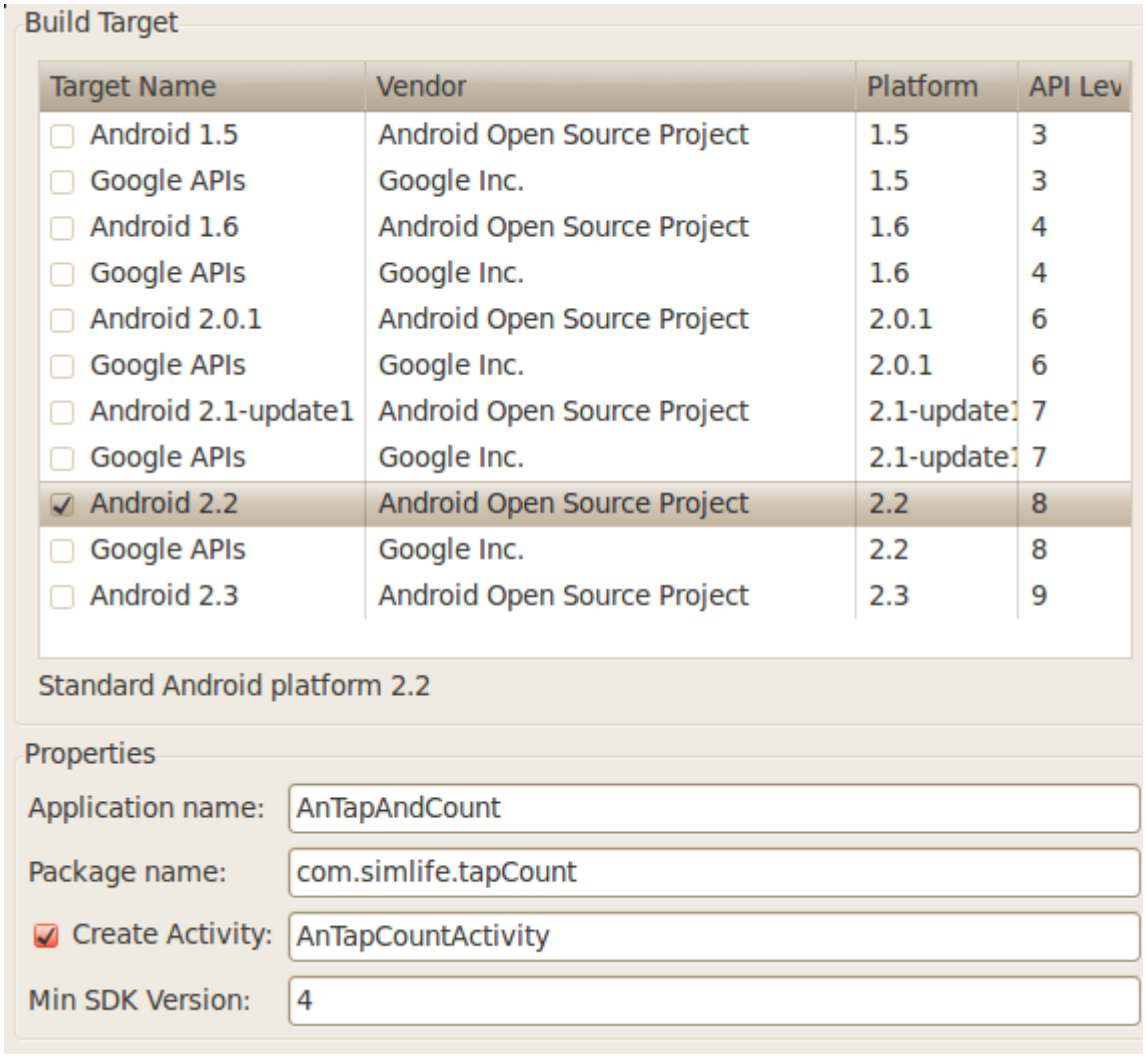
(Tutorial originally and primarily written by laschweinski, revised and edited by newton64.)

1. Download and configure the Android development environment

- Download the latest versions of the Android SDK and NDK. The latter is required in order to build native apps (i.e., using C/C++), which we're doing.
  - In China, these can be acquired from a mirror such as http://androidappdocs.appspot.com/index.html.
- Follow the installation instructions to make sure the SDK is properly installed.
- ~~Windows users will need to download Cygwin to emulate the GCC environment~~ (no longer the case, nor required. Standard DOS window will suffice with the NDK).
  - Mac, Windows and Linux users can use the NDK directly, and may optionally add the NDK folder to their $PATH environment variable
- ~~Download Eclipse and install the ADT plugin provided by Google~~ Available now as part of the SDK bundle. See http://orx-project.org/wiki/en/orx/tutorials/setup_android.
- Make sure you have up-to-date versions of the SDK, NDK, and ADT.
- The Google/Android sites linked above have a lot of information on installing and configuring these packages, and are worth browsing if there are any issues.

2. Create and configure an Android project

- Open Eclipse and click on File → New → Android Project. If the Android SDK is installed correctly, you'll see a number of build targets to choose from. If not, go back to the installation instructions linked above and check for problems.
- In this example, we're calling our project *AnTapAndCount*. Choose a build target and fill out the Properties section as shown in the image below. Of note:
  - *AnTapAndCountActivity* will be Android's “Main” and “Launch” activity
  - Setting the “Min SDK Version” will accommodate the widest (and oldest) range of Android versions. Advanced users can of course feel free to increase the requirements as they see fit.



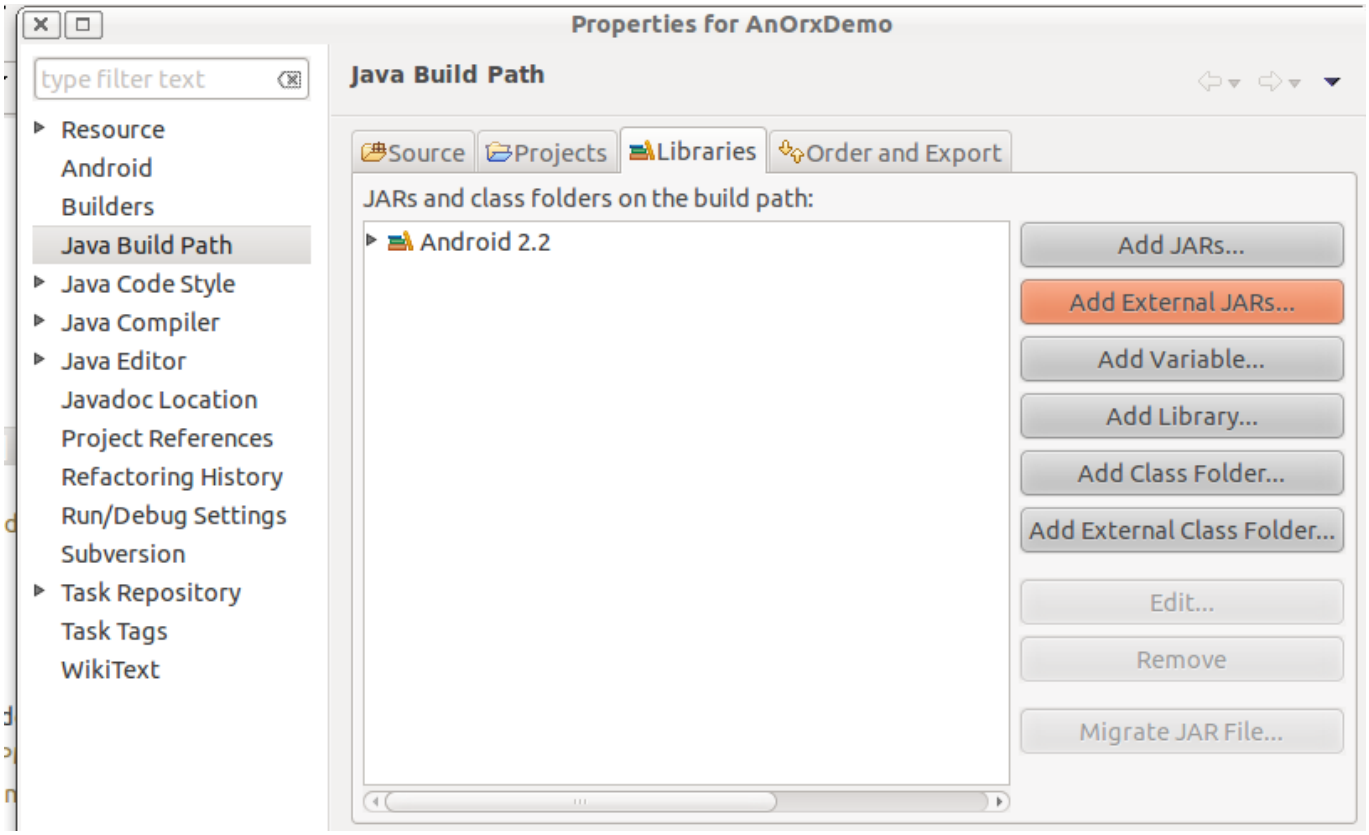
- Click “Finish” to create the project and populate it with certain default files.
- Double-click *AndroidManifest.xml* in your Package Explorer. In all likelihood, your editor window will now show a GUI for editing the XML manifest. The parameters discussed below can be entered in the GUI fields and drop-downs, or you can edit the XML file manually by clicking on the “AndroidManifest.xml” tab at the bottom of the editor window. To learn more about the AndroidManifest.xml file, check out the help pages here.
- The “android:screenOrientation” field can be set to “portrait” or “landscape.” Set this variable if you wish to lock the screen to a particular orientation; the default (blank) behaviour is unlocked, where the screen will rotate based on the orientation of the device. We're using “portrait” here.
- The “android:minSdkVersion” is set to “4” in this example. This is equivalent to Android v1.6, which should satisfy most users.
- The “android:name=“android.permission.WRITE_EXTERNAL_STORAGE”” field is **very important**. We will be writing data to external storage (the SD card), so this permission **has** to be enabled.
- When you're finished editing, the AndroidManifest.xml file should look something like this:

```
<application android:icon="@drawable/icon" android:label="@string/app_name"
android:debuggable="true">
  <activity android:name="..AnTapCountActivity"
android:label="@string/app_name" android:screenOrientation="portrait"
android:theme="@android:style/Theme.Black.NoTitleBar.Fullscreen">
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
```

```
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
</application>
<uses-sdk android:minSdkVersion="4" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

3. Add required sources and assets to the project

- Download the latest compiled version of Orx here. Look for the “full-android” package, and unzip it wherever you like to keep your code libraries.
- The *AnTapAndCount* project we're creating must have a reference to orxJava.jar, the Java wrapper around Orx functions. There are two ways to go about this:
  - Add an external reference to the project. Go to Project → Properties, and in the window that pops up go to the Java Build Path section and choose the Libraries tab. Click on “Add External JARs...”, and browse to the .jar file, which should be /path/to/orx/code/build/android/orxJavaLib/orxJava.jar



OR

- Use your file browser to navigate to /path/to/orx/code/build/android/orxJavaLib/src. Copy the org/ folder there, and paste it into your own application's src/ directory. Your application's src/ folder should now contain com/ and org/ sub-directories.
- The latter method will make working with Java within Eclipse a bit easier, as there are fewer dependencies to manage. It is recommended you use this latter method.

FORMERLY INCLUDED, BUT PROBABLY DEPRECATED: ~~b. copy and paste the headers and lib of orx from the orx release folder to your project, the path of it could be defined by yourself, but you should~~

~~modify the Android.mk as what I describe at the next section. note that you could also link to the lib in the orx release folder. in this case, no any move need to be applied. But some configuration has to set, same as the next section illustrate.~~

- Create a directory named “jni” in your highest-level project folder. We'll store our C/C++ code here.
- When we compile our game into a .apk file, our assets/ folder will not have write permissions. Read-only assets (images, sounds, &c.) can stay in this folder; however, anything that will need to be written/overwritten (.ini config files, for example) will need to be placed on the device's SD card.
- Within the assets/ folder, create another sub-folder with the same name as Orx activity (we'll get to that below): in this case, *TapAndCount*. This folder will be moved to the device's SD card when the program launches. **[Or is this only when the program is installed?]**
- All other asset files (again: images, sounds, &c.) can be placed in the higher-level assets/ folder
- Your current project structure should look something like this (where the media and code files are, of course, placeholders):

- AnTapAndCount
  - src/
    - com.simlife.tapCount
    - org.orxproject.lib
  - gen/
  - Android (version number)
  - assets/
    - TapAndCount
      - ... all your .ini files...
    - someimage.png
    - somemusic.ogg
    - ...other media...
  - jni/
    - myheader.h
    - mycode.cpp

4. Edit the Makefile and compile and build the program

- In the directory /path/to/orx/code/build/android/jni, there is a file called *Android.mk*. This is the makefile required by the NDK to compile your program.
- Copy Android.mk into your application's jni/ directory.
- Edit the first few lines to look something like this:

```
#the name of the lib
ORX_MODULE_NAME := orxTapAndCount
#set all of sources
ORXAPP_SRCS := *.c *.cpp
ORXLIB     := extern/lib/android
ORXINCLUDE := extern/include
BOX2DLIB     := extern/Box2D_2.1.3/lib/android
BOX2DORXINCLUDE := extern/Box2D_2.1.3/include
          SOILLIB         := extern/SOIL/lib/android
```

- A few notes:
  - ORX_MODULE_NAME is the the name of NDK library for this Orx application (e.g., if you specify “orxFoo,” NDK will compile a library named liborxFoo.so; this is reflected in the Java loader as well).
  - ORXAPP_SRCS is the path of all sources that require compilation. To compile all source files, ORXAPP_SRCS can simply be assigned “*.cpp *.c”, as above.
  - ORXLIB and ORXINCLUDE set the paths of the Orx libraries and includes. On a PC, these are all relative to the local jni/ directory.
- Once the makefile is ready, and presuming the NDK was installed properly in Step 1, you can use ndk-build to generate the JNI library that will be used by your Android app (again: this is compiling your Orx C/C++ code into a library, which will then be loaded and run by a Java program on the device).
- Run ndk-build from the command line. If you haven't added the NDK directories to your system $PATH, you'll have to give the full path to ndk-build. Either way, the output should resemble the following, and the required libraries will be generated in libs/armeabi:

```
laschweinski@laschweinski-desktop:~/android/workspace/AnTapAndCount$ ls
AndroidManifest.xml  bin              gen  proguard.cfg  src
assets               default.properties  jni  res
laschweinski@laschweinski-desktop:~/android/workspace/AnTapAndCount$ ndk-build
Prebuilt       : libjnigraphics.so <= jni/extern/orx/lib/
Install        : libjnigraphics.so => libs/armeabi/libjnigraphics.so
Prebuilt       : liborx.so <= jni/extern/orx/lib/
Install        : liborx.so => libs/armeabi/liborx.so
Compile thumb  : orxApp <= orxTest.c
Compile++ thumb  : orxApp <= android-support.cpp
StaticLibrary  : libstdc++.a
SharedLibrary  : liborxApp.so
Install        : liborxApp.so => libs/armeabi/liborxApp.so
```

- If you're compiling in debug mode, use “ndk-build ORX_DEBUG=true -B” to compile the application, in addition to setting the isDebug Java parameter to true, as shown below.

- Our *Activity file (in this case the AnTapCountActivity) must be modified. The file is *AnTapCountActivity.java* in src/com.simlife.tapCount/.
- Add the onCreate() function, written below, to the file:

```
protected void onCreate(Bundle savedInstanceState) {
    isDebug = false;
    //when you need to run in the emulator it should set to true
    usingGLES1 = false;
    appPath = "bounce_demo";
    appName = "orxTest";
    orxAppLibName = "orxTapAndCount";
    // load the .so lib in AnOrxActivity
    super.onCreate(savedInstanceState);
    // So we can call stuff from static callbacks
    // TODO keep the screen on, I know it is not a perfect decision,
    // acticity should resume from pausing.
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON,
            WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    //build the gl surface, start to init and run the loop
    mSurface = new ORXSurface(this);
    setContentView(mSurface);
```

```
            SurfaceHolder holder = mSurface.getHolder();
            holder.setType(SurfaceHolder.SURFACE_TYPE_GPU);
        }
```

- A few more notes:
  - *appPath*: As described in the previous section, several files requiring write permissions will be moved to the device's SD card. This parameter names the folder containing those files.
  - *AppName*: The name of this app, which determines the name of the .ini config file loaded automatically by Orx.
  - *orxAppLibName*: The name of the library generated by the NDK. Note that when debug mode is selected, the library will have the suffix 'd' added automatically.
  - *isDebug*: As mentioned above, this declares whether the app is being built in debug mode (which will add the 'd' suffix to the library name).
  - *UsingGLES1*: If true, only the GLES1.0 OpenGL library will be loaded. This will use GLES1.0 for all rendering, although this will remove support for shaders. If this is set to false, the device will load its default version of GLES; presumably with shader support. Basically, this is a flag to ensure that your game (without shaders) will run on older hardware. **[I think]**
  - *mSurface*: This is the surface that is creating the main glContext; essentially the surface for the whole application.
- Finally, with the C/C++ libraries built (in the previous section), and the Java files configured, just go to Project → Build Project to actually build an .apk file. This file, long story short, is your game.

5. Transfer the game to an Android device and enjoy!

- Using USB or Bluetooth, transfer the .apk file to your Android device.
- Using the device's file browser, run the file. It may complain about running files not from the app store, but in this case you can probably trust your own code. ;)
- The lighting demo contained in /path/to/orx/code/demo/android-light-demo is a good example or template from which to build.

From newton64: **Good luck!**

From laschweinski: **enjoy yourself in game development of orx. Thank you.**