Tutorial 1: Introduction to the Android Development Environment

Admin Details:

Individual Assignment: You must complete this tutorial on your own.
Due date: 9/5/2014 by 8 pm, late work is not accepted
Submission: t1.zip a zip file of your Eclipse project.
Submit files via Canvas

Value: Completion of this tutorial is worth 25 points, about 2.5% of your final grade.

Introduction

In this tutorial, you will setup your Android development environment and go through the Hello World tutorial on the Android Developer website at http://developer.android.com/training/basics/firstapp/index.html.

You will also learn about using the Eclipse debugger and logging errors and other information in a running Android application.

You may complete this on your own machines or the computers in the CS department Microlab. Eclipse and the Android development environment is already set up on the Microlab Windows machines and the public Linux machines. Be careful when using the Linux system. If you create an AVD (Android Virtual Device, an emulator to test Android apps) it can eat up your entire hard disk quota on the CS Linux system. Set the Ram and internal memory very low for your AVD.

Setup Your Environment (Skip this part if using the department's machines.)

The best way to develop applications for Android is using the Eclipse IDE and the Android plug-in for Eclipse called Android Development Tools (ADT). You can test your applications by running them on an Android Virtual Device (AVD), an Android emulator.

To setup your environment, follow each of the steps below in order. Installation instructions are available here: http://developer.android.com/sdk/index.html

1. Download and install JDK 7
   http://www.oracle.com/technetwork/java/javase/downloads/index.html

2. If you don't have Eclipse or want to try the all in one Android - Eclipse download visit http://developer.android.com/sdk/index.html and download the SDK with the ADT Bundle for your system. This includes Eclipse, the Android Development Tools (ADT plugin), the Android SDK, and other tools. This is designed to be an all in one install, although I have not used it.

3. If you already have an Eclipse IDE the same page has instructions under "USE AN EXISTING IDE". This entails downloading the SDK Tools and installing the ADT yourself.

   ("A new Android development environment called Android Studio, based on IntelliJ IDEA is available as an early access preview," but I have not worked out all the technical issues with Android Studio for the lab and for turning in projects. I require you to use Eclipse for the CS371m tutorials. You may use Android Studio on your app projects later in the semester.)

 Frank McCown at Harding University, minor changes by Mike Scott

Working on the Departmental Machines

The Android development environment is installed on the CS Department Windows and Linux machines.

Using the Windows machines is straightforward. Just run Eclipse and created AVDs. Please note the AVDs get wiped each time you log out.

If you want to work on the Linux machines you must complete some set up. The AVDs take up a lot of memory and will quickly consume your 2GB departmental disk quota.

There is a projects folder for the class you can use to store your AVDs. This will not eat into your disk quota.

The name of the projects folder is cs371m-scottm

To set up your .android directory in the projects folder to store your emulators:

make a directory in the /projects directory

something like
```
mkdir /projects/cs371m.scottm/<user_name>
```

Then you will want a `.android` directory
```
mkdir /projects/cs371m.scottm/<user_name>/.android
```

Then go to your home directory
```
cd
```

Then make the symlink
```
ln -s /projects/cs371m.scottm/<user_name>/.android
```

This pre-supposes you do not already have a /u/<user_name>/.android
directory.

If you DO, you can
```
mv .android /projects/cs371m.scottm/<user_name>
```

instead of doing the second mkdir above, and then make the symlink.

We have 20GB to start. If you have space issues while using the projects directory please let me know.

Building Your First App Tutorial

Complete the "Building Your First App" tutorial at
http://developer.android.com/training/basics/firstapp/index.html which takes you through creating an Android Virtual Device (AVD), creating an Android project, creating two Activities that interact with each other, adding some simple GUI components, editing the layout xml file, using the visual GUI editor, starting another Activity via an Intent, and running the application in the emulator. Complete all four sections of the tutorial: Creating an Android Project, Running Your Application, Building a Simple User Interface, and Starting Another Activity.

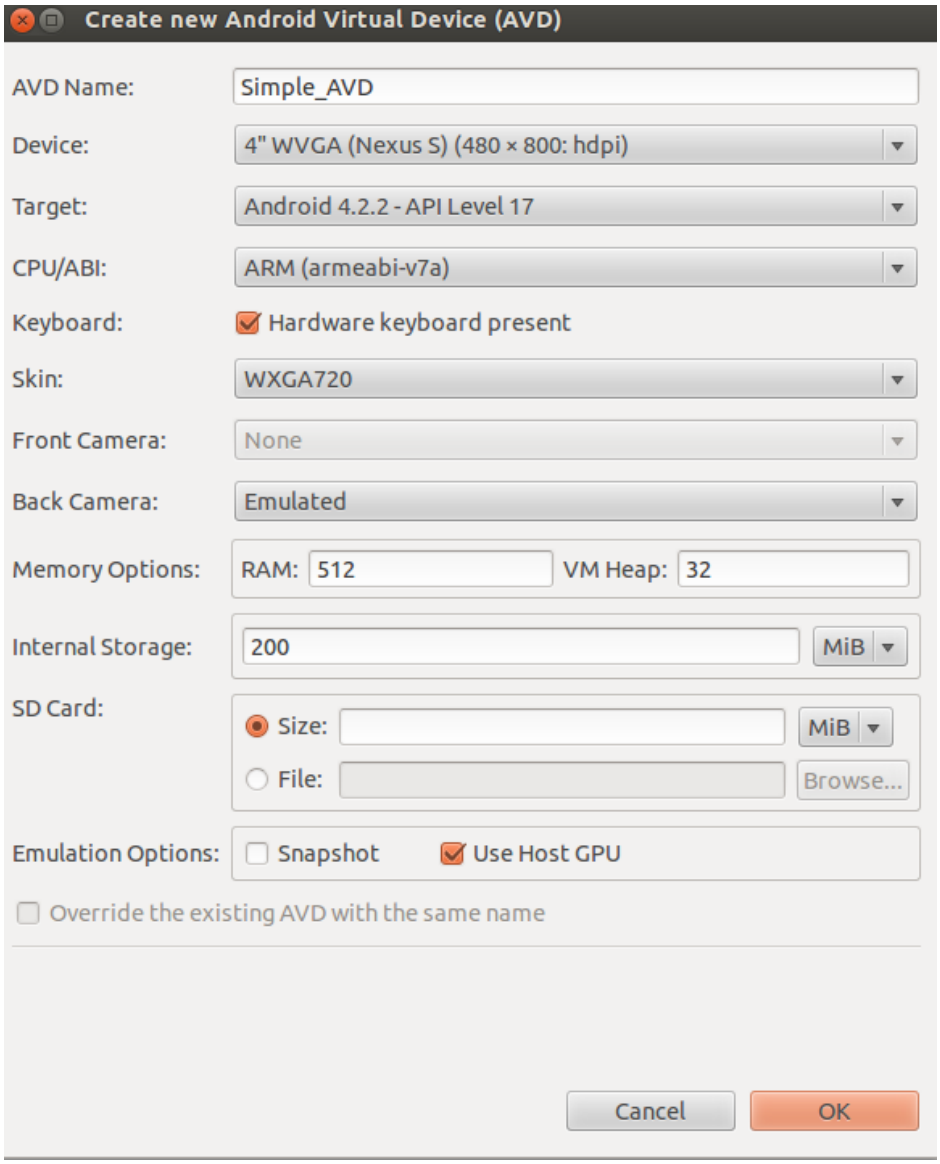
 Frank McCown at Harding University, minor changes by Mike Scott

Use a min API level of 14 (version Android 4.0, Ice Cream Sandwich) and a target API level of 17 (Android 4.2, JellyBean).

If you are working on the Microlab Windows machines or public Linux machines, you can create your own AVDs via the AVD manager.

If you are working on the CS Linux machines, please note the following: **Eclipse is installed on the machines and you can start in by running `eclipse` on the command line. When you open it for the first time, a box will come up asking where to find the Android stuff. Hit the radio button by "Use existing SDKs" and enter** /lusr/opt/android-sdk-linux-r23_0_2 **in the box.**

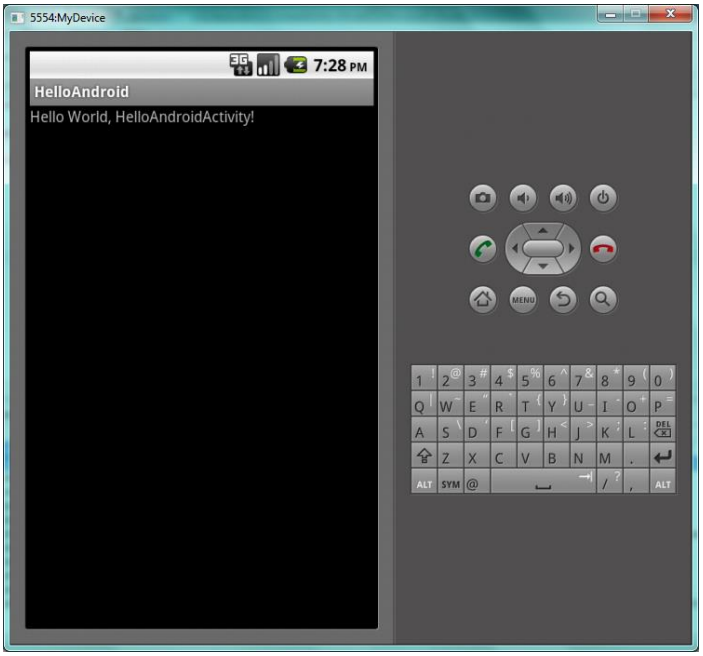
Here is a screenshot of an AVD I created on the lab machines. Note the choices I made for the emulator's properties. This configuration led to an emulator that started up fairly quickly:



When you run an Android application, Eclipse will first start an emulator if one isn't already running. If you have multiple AVD configurations, it's usually a good idea to *first* start your emulator of choice so you can control

which emulator you want to use. It may take a few minutes for your emulator to appear because of a lengthy initialization period… you'll have to be patient. (http://stackoverflow.com/questions/1554099/slow-android-emulator) You can watch the status of the emulator from Eclipse's Console window. It will look something like the lines displayed below which show how the .apk file is first installed and then launched:

```
[2011-10-22 14:22:34 - HelloAndroid] ------------------------------
[2011-10-22 14:22:34 - HelloAndroid] Android Launch!
[2011-10-22 14:22:34 - HelloAndroid] adb is running normally.
[2011-10-22 14:22:34 - HelloAndroid] Performing com.example.HelloAndroidActivity activity launch
[2011-10-22 14:22:34 - HelloAndroid] Automatic Target Mode: using existing emulator 'emulator-5554'
running compatible AVD 'MyDevice'
[2011-10-22 14:22:34 - HelloAndroid] Uploading HelloAndroid.apk onto device 'emulator-5554'
[2011-10-22 14:22:34 - HelloAndroid] Installing HelloAndroid.apk...
[2011-10-22 14:22:38 - HelloAndroid] Success!
[2011-10-22 14:22:38 - HelloAndroid] Starting activity com.example.HelloAndroidActivity on device
emulator-5554
[2011-10-22 14:22:40 - HelloAndroid] ActivityManager: Starting: Intent {
act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER]
cmp=com.example/.HelloAndroidActivity }
```



You can also open up the LogCat to see that the emulator startup process is making progress. In Eclipse go to **Window -> Show View -> Other -> Android -> LogCat**

Once the application is started, you will see it running like the figure on the right. Note that this is the default Android 2.3 AVD skin. If you use a different version of Android, your emulator will look somewhat different.

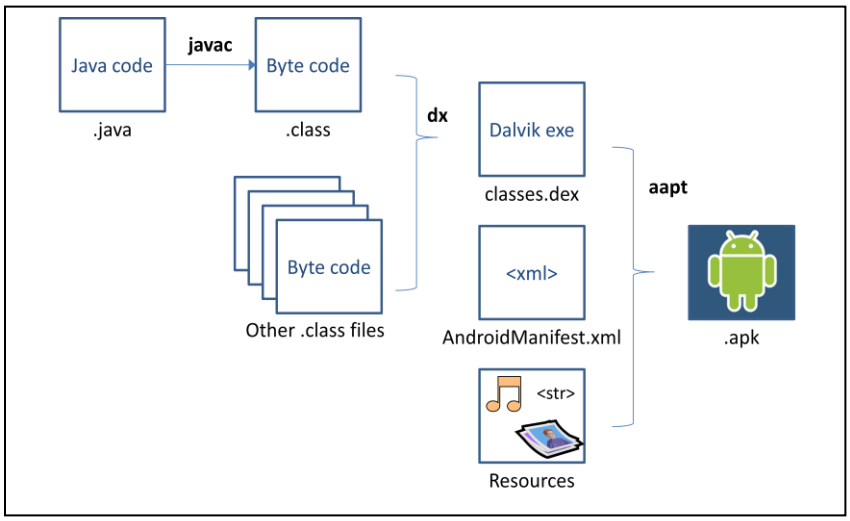
You can use the emulator as you would a real Android device by using your mouse to click on the screen and using your keyboard to type.

When you make changes to your program, **DO NOT CLOSE THE EMULATOR** Just edit your program, save it (which makes Eclipse re-build it), and re-run your new program (Ctrl-F11); the emulator will replace the previous program with the new one and execute it (you'll have to be patient… it can take up to a minute to reload).

The Android Package

The .apk file, the Android Package, is the application file that was installed and executed on the emulator. The .apk file is produced by Eclipse in a number of steps as illustrated in the figure below.

 Frank McCown at Harding University, minor changes by Mike Scott



1. The Java source code is compiled into Java bytecode (.class files) as is traditionally done in Java programming.
2. The dx tool converts the Java bytecode into Android bytecode (.dex files), a compact format that can be executed by a Dalvik virtual machine.
3. Finally, the Android Asset Packaging Tool (aapt) is used to combine .dex files with the AndroidManifest.xml and other resources making up your application. It produces the .apk which is really just a zip file.

You can find .apk files on the Web and install them directly to your emulator or Android device although you must be careful about installing un-trusted .apk files that could contain malicious code.

Using the Debugger

Review these pages on using the Android debugger:

- http://developer.android.com/tools/debugging/index.html
- http://developer.android.com/tools/debugging/ddms.html

Logging Messages

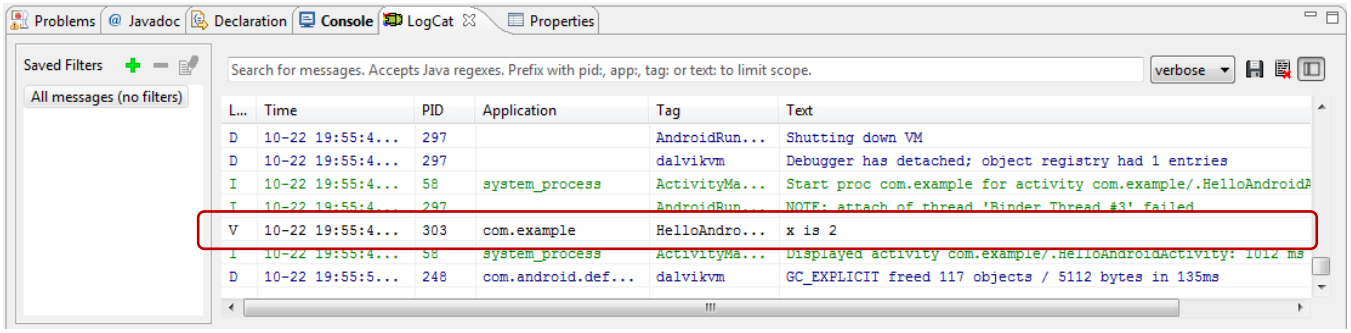
An alternative to the debugger for simple tracing is the Android logging class (`android.util.Log`) The log class is used to send messages to the **LogCat**. The LogCat should be visible in one of the tabs near the bottom of the Eclipse window. If you don't see the LogCat, select **Window > Show View > Other…, Android > LogCat** to make it visible.

To log messages, first import `android.util.Log` and declare a log tag (a unique identifier for your log messages). Add a call to `Log.v()` in your program like so:

 Frank McCown at Harding University, minor changes by Mike Scott

```
package com.example.helloandroid;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MyFirstActivity extends Activity {

    private static final String LOG_TAG = "HelloAndroid_tag";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        int x = 2;
        Log.v(LOG_TAG, "x is " + x);
    }
}
```

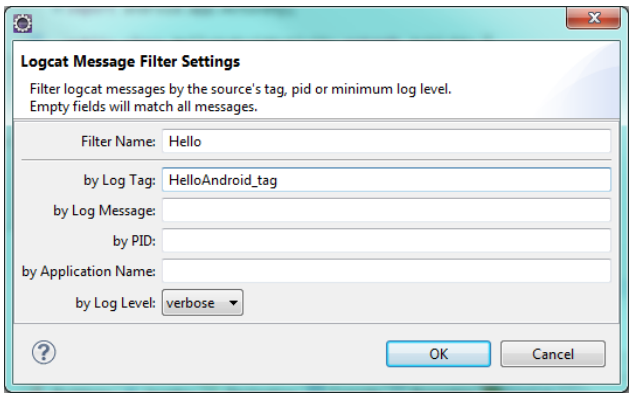
Now run your program. When the `onCreate()` method runs, the `Log.v()` call will output "`x is 2`" to the LogCat as shown below.



Note that there are different levels of log messages:

Verbose:	`Log.v()`
Debug:	`Log.d()`
Information	`Log.i()`
Warning	`Log.w()`
Error	`Log.e()`

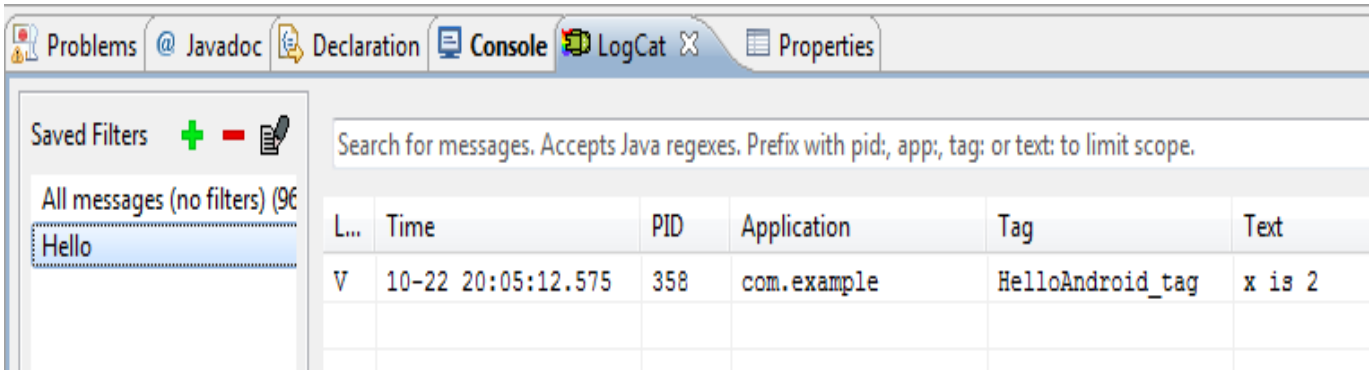
Typically, you'll want to use the appropriate method to log the severity of the error message.



Because the log messages can easily get lost in the numerous messages shown in the LogCat, it's a good idea to create a filter so only the messages logged by our app are displayed.

First click the green + next to Saved Filters (left side of the LogCat window). This will launch a dialog box as shown to the right. Enter **Hello** for the Filter Name and **HelloAndroid_tag** as the Log Tag and press OK.

Now when "Hello" is selected from the list of filters, you should only be able to see the messages logged by the app as shown below. If you'd like to see all the messages, click "All messages".



Submission

Zip up your Eclipse project with the two Activities that communicate back and forth. Name it t1.zip. Submit your program via Canvas.

Note on Grading (from the, Nathan former TA)

There are roughly 50 students in the class. Last semester, there were only about 30, and the grading on even that few was taxing at times. If you turn in an app that doesn't even run on the emulator or device, chances are I won't have time to look through your code and find out what was wrong. Test your app before you turn it in on at least one other environment.

I've already said it, but make sure you test your apps. Last semester, I graded with a device with API 15 and the emulator with API 10, so if it didn't work under one environment, I could check it under a second. (You are to use minimum API level of 14 and a target API level of 17) I found that most of the issues last semester were not from different APIs, but from people not testing their app thoroughly. Since these apps are relatively simple to test (you're not programming an OS or writing an AVL tree that can't really be understood by non-CS majors), you can have a friend or roommate play it once or twice just to make sure.

Finally, the tutorials are relatively simple. However, you will occasionally run into some issues if you change your environment (API version, computer you're using, etc). So start early and get them done hassle-free.