<u>Android: How To.</u>

This is just a collection of useful information and tricks that I used during the time I was developing on the android ADP1. In some cases the information might be a little old and new and easier ways to do the same things might be available. Please let me know if you find something like this so that, I can update this document. This is not original work and neither do I claim so. This is just a collection of references that you may find useful. This is in no way a definitive guide and there is much more information out there.

Thanks

Aman Nijhawan

<u>Index</u>

Part 1: Activating your android ADP 1 without the sim card	.2
Part 2: Installing the Developing Environment	.3
Part 3: Part 3: Updating your Android phone to the latest Firmware	.4
Part 4: APIs and links to code reference and sample code	
for accessing onboard Hardware	.6
Part 5: Scripting on Android	7
Part 6: Compiling and Installing Linux Binaries for Android	9

Part 1: Activating your android ADP 1 without the sim card

- 1. Plug your Android Dev Phone into your computer via USB if you are using windows 64 bit as your OS you will need additional instructions about the driver*.
- 2. Get into the Phone's OS... Open up a terminal and type:

adb -d shell

- 3. Now, change to the Phone's settings database directory, type: cd /data/data/com.android.providers.settings/databases/
- Give yourself admin permissions by typing: su
- Now, you are going to use sqlite to alter the phone setting database, and turn provisioning on, so type: sqlite3 settings.db
- Use this SQL command to alter the settings database: INSERT INTO system (name, value) VALUES ('device_provisioned', 1);
- 7. Now, exit sqlite by typing this (don't forget the "."): .quit
- 8. Reboot the Phone by typing: **reboot**
- 9. When the phone reboots, you need to open up the settings screen to activate your WiFi settings.
- 10. First, get back into the Phone's OS by typing: adb shell
- 11. Now, run this command to open up the settings screen on the Phone, which will allow you to turn on WiFi:

am start -a android.intent.action.MAIN -n com.android.settings/.Settings

*Windows Vista 64 bit Driver Issue:

At the time of writing this document windows vista 64 bit did not have a driver for android platform, so there is an unsigned driver that is available for the phone. However the issue with windows Vista is that it won't allow you to install unsigned drivers without some hacks. (I have attached the driver in the tar file

- 1. Type "bcdedit /set testsigning on" at an admin command prompt Reboot
- 2. Check to see if the desktop says "Test Mode" on the corners after reboot.
- 3. Install the given driver now.

Part 2: Installing the Developing Environment

Once you have installed the android phone and looked around in the interface and checked out the *command prompt emulator* installed *skype* to make free calls over wifi and got the hang of the device, its time to get down to serious business.

The android development environment is basically a Stripped down version of J2ME environment.

2.1 Installing Android SDK:

 Download Eclipse(3.4) from : <u>http://www.eclipse.org/downloads/</u>
Install Jdk6 preferably if you don't have it already installed.
Download the Android SDK for your eclipse version and OS from <u>http://developer.android.com/sdk/1.5_r3/index.html</u>

4. Installing the ADT Plugin for Eclipse

Android offers a custom plugin for the Eclipse IDE, called Android Development Tools (ADT), that is designed to give you a powerful, integrated environment in which to build Android applications. It extends the capabilites of Eclipse to let you quickly set up new Android projects, create an application UI, add components based on the Android Framework API, and debug using the Android SDK tools.

Eclipse 3.4 (Ganymede)

- 1. Start Eclipse, then select Help > Software Updates....
- 2. In the dialog that appears, click the Available Software tab.
- 3. Click Add Site...
- 4. Enter this as the Location: https://dl-ssl.google.com/android/eclipse/

Alternatively, you can use http in the Location URL, if you are having trouble with https http://dl-ssl.google.com/android/eclipse/ Click OK.

- 5. Back in the Available Software view, you should see the plugin. Select the checkbox next to Developer Tools and click Install...
- 6. On the subsequent Install window, "Android Developer Tools", and "Android Editors" should both be checked. The Android Editors feature is optional, but recommended. If you choose to install it, you need the WST plugin mentioned earlier in this page. Click Next.
- 7. Accept the license agreement and click Finish.

Restart Eclipse.

Now, you just need to modify your Eclipse preferences to point to the Android SDK directory:

- 1. Select Window > Preferences... to open the Preferences panel. (Mac OS X: Eclipse > Preferences)
- 2. Select Android from the left panel.
- 3. For the SDK Location in the main panel, click Browse... and locate the SDK directory.
- 4. Click Apply, and then OK.

Done!

5. Hello World for Android: Go through the helloworld(s) give on this page to get a feel of the developing environment.

http://developer.android.com/guide/tutorials/hello-world.html

- 6. Go through
- 7. http://developer.android.com/guide/tutorials/hello-world.html

Part 3: Updating your Android phone to the latest Firmware Please go through the useful ADB Commands before you move any further: http://developer.android.com/guide/developing/tools/adb.html

Flashing the phone:

The ADP1 phones given in class need to be updated first before you could try and work with them, I found this out because some of the code that I wrote was working on the simulator but not on the phone, hence the flashing.

It is also necessary to flash the phone in order to run Android Scripting Environment.

This is a long process and the best post I could find on this was:

http://www.androidkit.com/flashing-firmware-images-to-the-adp1-device

Please go through all steps carefully.

Part 4: APIs and links to code reference and	d sample code	for accessing onboard Hard
--	---------------	----------------------------

Hardware	API CLASS	Reference Links
Device		
GPS	android.location	http://developer.android.com/reference/android/lo
		cation/LocationManager.html
		Sample Application
		http://blogoscoped.com/archive/2007-11-19-
		<u>n27.html</u>
GSM Telephony	android.telephony	http://developer.android.com/reference/android/t
		elephony/package-summary.html
		Comula Analization
		Sample Application
		Listen to incoming SiVIS Application
		http://davanum.wordpress.com/200//12/15/androi
		d-listen-for-incoming-sms-messages/
		Got notification about incoming calls
		http://mylifewithandroid blogspot.com/2008/01/ph
		onecalls html
		Access Cell Details such as cell id, signal strength etc
		http://dayanum.wordpress.com/2007/11/29/androi
		d-access-cell-phone-details-cell-id-lac-signal-
		strength/
		http://davanum.wordpress.com/2007/11/30/androi
		d-poor-mans-my-location-show-current-cell-
		location/
Wifi	android.net.wifi	http://developer.android.com/reference/android/n
		et/wifi/package-summary.html
		Wifi Social Networking using maps and wifi
		http://www4.ncsu.edu/~ppmarqui/csc714/index.ht
		<u>ml</u>
XML Parsing	android.sax	http://developer.android.com/reference/android/s
		ax/package-descr.html
Soncore: Cot	Sancar	http://douglongr.ondroid.com/usfaugues/audusid/h
information	Sensor	ardware/Sensor html
about the		
about the		
on the device		
on the device.	1	

Accelerometer, Compass	SensorManager	http://developer.android.com/reference/android/h ardware/SensorManager.html
Camera	android.hardware	http://developer.android.com/reference/android/h ardware/package-summary.html

Part 5: Scripting on Android

Android Scripting allows development of applications on the phone itself, without the need to be tied to a computer. ASE or android scripting environment allows you to edit and execute scripts and interactive interpreters directly on the Android device.

Languages supported currently: **Python, Lua, Perl, Jruby, Beanshell.**

You can download ASE on the phone from:

http://android-scripting.googlecode.com/files/AndroidScriptingEnvironment-0.11-alpha.apk

Install it as an apk package on the phone: Using

adb install <source>/AndroidScriptingEnvironment-0.11-alpha.apk /<destination>

Open application & select your favorite interpreter and you will get a few sample scripts you can use these scripts and create your own script, either directly in the phone or on a computer on your favourite IDE and use adb push to push the script to the phone.

adb push myscript.py /sdcard/ase/scripts/myscript.py

For details on scripting you can visit the following links:

http://community.developer.motorola.com/t5/MOTODEV-Blog/Scripting-Android-by-Mike-Riley/ba-p/1426

http://google-opensource.blogspot.com/2009/06/introducing-android-scripting.html http://code.google.com/p/android-scripting/wiki/PythonAndroidAPI

http://code.google.com/p/android-scripting/wiki/LuaAndroidAPI

Part 6: Compiling and Installing Linux Binaries for Android Note: Your Development Environment for this should be Linux to setup the tool chain compiler

I have had some success with the process given at http://android-dls.com/wiki/index.php?title=Compiling_for_Android

apt-get install emdebian-tools emsetup --arch armel

To compile simple static applications:

arm-linux-gnueabi-gcc -static -o myapp myapp.c

So the limitation here is that the application is statically linked. A nicer setup would give us dynamically linked applications. One of the trickier aspects here is that Android uses a custom, tiny libc, (bionic). Here are some instructions on getting a working setup for compiling a dynamically-linked program with the prebuilt Android toolchain. Perhaps someone can use these instructions to create a simpler process using the emdebian toolchain.

To get the Android prebuilt toolchain:

1. First download the Android source.

2. That ships with a prebuilt toolchain with a compiler named something like arm-eabi-gcc. It's convenient to get that onto your PATH with something like the following (assuming Android source in \$HOME/src/android):

export PATH=\$HOME/src/android/prebuilt/linux-x86/toolchain/arm-eabi-4.2.1/bin

3. Next you'll actually need to compile at least the bionic libc. Without having gone through the effort of doing a minimal compile here, you can do just "make" and wait quite a while for all of Android to get compiled. Note that early on the Makefile will abort if it can't find a JDK that it likes. Presumably no Java is actually needed for the native compilation we're doing here, but again, without having wrestled that out of the Makefiles yet, you can satisfy that requirement with packages from Debian's main repository:

apt-get install openjdk-6-jdk

4. Next, there are various flags and things needed to cross-compile an application and properly link it to bionic. Andrew Ross has written a perl script to take the pain away here and with the android toolchain in your PATH you're ready to go. Grab his <u>agcc</u> script and put it into your PATH somewhere, (perhaps in \$HOME/bin or so).

5. With all those steps done you can now cross-compile an application dynamically linked to bionic

agcc -o myapp myapp.c

1

.

Agcc perl wrapper can be found at: http://plausible.org/andy/agcc

You can then push myapp using adb on to the android device,

I used it to port and run iptables on android, however this was not very stable when I did it maybe this can be improved upon.