How to Build A Stock AOSP Rom

Version 1.0 (Revised 9-21-10)

Created by *TcrewsTcrews @ http://android.snkbitten.com/*

Hosted at ***Deodexed.comDeodexed.com Follow our tweets*** *@Deodexed*

Build/Compile an AOSP Froyo Rom for the Motorola Droid

How to configure your system to compile from AOSP (Android Open Source Project) for the Motorola Droid.

This is based on the "how to" guide found at http://howto.ccroms.net/http://howto.ccroms.net/http://howto.ccroms.net/ which I used to compile a few months ago, but I noticed some things have changed and some things I had to hunt for answers even back then. This is not the be-all-end-all of guides. It was written to be a guide that will walk you through obtaining and compiling the AOSP Froyo source with Superuser + root and Busybox included.

I have tried to include corrections for everything I encountered when initially following the guide at http://howto.ccroms.net/http://howto.ccroms.net/http://howto.ccroms.net/ as well as when I switched from 32bit to 64bit OS versions. This was one of the most frustrating things I've done...and I was doing it "just for fun".

First this is for a system running Ubuntu 10.04 x64 desktop OS (as tested). Ubuntu 32bit should be similar and 32bit and 64bit differences are noted.

I use a Windows7 Ultimate 64bit PC running OracleVirtualBox with a Virtual Machine (VM) of Ubuntu 64bit configured with 50gig of HD space, 2 gig of ram and a Dual core cpu with Virtual box Guest Additions installed.
http://www.virtualbox.org/ (oracle virtualbox)
http://releases.ubuntu.com/lucid/ubuntu-10.04.1-desktop-i386.iso (32bit workstations)
http://releases.ubuntu.com/lucid/ubuntu-10.04.1-desktop-amd64.iso (64bit workstations)

you can run 32bit or 64 bit virtual boxes on a Windows 64bit system.
you can only run 32bit virtual boxes on a Windows 32bit system

I have tried to be as complete as possible with this guide, creating a new VM and following the original guide at http://howto.ccroms.net/http://howto.ccroms.net/http://howto.ccroms.net/ and making the changes as needed and adding it to this guide.

SDK and Configuration

Start by download the latest SDK for Linux from here:

http://developer.android.com/sdk/index.html

and just save it to the default "Downloads" directory (just click save and ok)
Once it's downloaded, from a terminal:

```
cd ~
mkdir android
tar -zxvf ~/Downloads/android-sdk_r07-linux_x86.tgz
```

mv android-sdk-linux_x86 ~/android/sdk

Next we need to include ~/android/sdk/tools in our path so from the terminal type...

sudo gedit .bashrc (note...first thing to do in gedit is to select edit/preferences and then in the window at Line Numbers put a check for Display line numbers then click close. This will help you tremendously later)

and add the following to the bottom of the document that opens

export PATH=${PATH}:~/android/sdk/tools

and save and exit.

We need to add a device rule to allow our workstation and ADB to communicate with our phone.

in the terminal, again type..

sudo gedit /etc/udev/rules.d/51-android.rules

and then copy the following in to the file, save and exit.

```
SUBSYSTEM=="usb", SYSFS{idVendor}=="0502", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="413c", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="0489", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="091E", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="0bb4", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="12d1", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="0482", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="1004", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="22b8", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="0955", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="10A9", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="04e8", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="04dd", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="0fce", MODE="0666"
SUBSYSTEM=="usb", SYSFS{idVendor}=="19D2", MODE="0666"
```

Next we need to add Java. Java 5 is needed, however development is moving towards Java 6.

In the terminal type..

sudo gedit /etc/apt/sources.list

and add the following two lines at the bottom, then save and exit.

deb
http://us.archive.ubuntu.com/ubuntu/http://us.archive.ubuntu.com/ubuntu/http://us.archive.ubuntu.com/ubuntu/ jaunty multiverse
deb
http://us.archive.ubuntu.com/ubuntu/http://us.archive.ubuntu.com/ubuntu/http://us.archive.ubuntu.com/ubuntu/ jaunty-updates multiverse

Now in the terminal type each line..

sudo apt-get update

sudo apt-get install sun-java5-jdk (answer Y to the question)
when the java install screen pops up, hit 'TAB' to move to OK hit ENTER then cursor to YES and hit ENTER

sudo update-alternatives --config java (if it returns "nothing to configure" that is ok)

When that is done we want to remove the lines we added earlier to the sources.list file.

In the terminal..

sudo gedit /etc/apt/sources.list

and delete the bottom two lines you added ...

deb
http://us.archive.ubuntu.com/ubuntu/http://us.archive.ubuntu.com/ubuntu/http://us.archive.ubuntu.com/ubuntu/ jaunty multiverse
deb
http://us.archive.ubuntu.com/ubuntu/http://us.archive.ubuntu.com/ubuntu/http://us.archive.ubuntu.com/ubuntu/ jaunty-updates multiverse

save and exit.

Now we install all the dependencies needed...

in the terminal..

(for 32 bit)

sudo apt-get install git-core gnupg flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl libncurses5-dev zlib1g-dev sun-java5-jdk
(answer Y to the question)

(for 64 bit)

sudo apt-get install git-core gnupg sun-java5-jdk flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl libncurses5-dev zlib1g-dev valgrind lib32readline5-dev gcc-multilib g++-multilib libc6-dev-i386 lib32ncurses5-dev ia32-libs x11proto-core-dev libx11-dev lib32readline5-dev lib32z-dev
(answer Y to the question)

Getting the source

Next up REPO (and GIT) is installed.

in terminal...

```
cd ~
mkdir bin
sudo gedit .bashrc
```

and add the following text to the bottom and then save and close.

```
export PATH=${PATH}:~/bin
```

at this point I recommend closing your terminal window and opening a new one to use the PATH additions we've done.

next..in the new terminal..

```
curl
http://android.git.kernel.org/repohttp://android.git.kernel.org/repohttp://android.git.kernel.org/repo
>~/bin/repo
```

```
chmod a+x ~/bin/repo
```

and we are done until we pull the repository to build/compile.

```
cd ~
cd android
mkdir system
```

```
cd ~/android/system
```

```
repo init -u git://android.git.kernel.org/platform/manifest.git -b froyo
```

You may get asked to enter your Name and Email Address, then if that info is correct and possibly if you want "Testing colorized......", just Y and Y those and you get
"repo initialized in home/(your system name)/android/system" if you were successful.

Now we pull the source to our local workstation

type..

```
repo sync
```

This will take a while as it's hundreds of megs of data. Take a break, watch some TV, make a sandwich, etc...
Next we verify the GIT tags..

type..

gpg --import

Nothing will appear to be happening, this is fine. Paste the following into the terminal window and then do a CTRL+D to close and save. This will return you back to the terminal command prompt.

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.2.2 (GNU/Linux)

mQGiBEnnWD4RBACt9/h4v9xnnGDou13y3dvOx6/t43LPPIxeJ8eX9WB+8LLuROSV
lFhpHawsVAcFlmi7f7jdSRF+OvtZL9ShPKdLfwBJMNkU66/TZmPewS4m782ndtw7
8tR1cXb197Ob8kOfQB3A9yk2XZ4ei4ZC3i6wVdqHLRxABdncwu5hOF9KXwCgkxMD
u4PVgChaAJzTYJ1EG+UYBlUEAJmfearb0qRAN7dEoff0FeXsEaUA6U90sEoVks0Z
wNj96SA8BL+a1OoEUUfpMhiHyLuQSftxisJxTh+2QclzDviDyaTrkANjdYY7p2cq
/HMdOY7LJlHaqtXmZxXjjtw5Uc2QG8UY8aziU3lE9nTjSwCXeJnuyvoizl9/l1S5
jU5SA/9Wwlps4SC84ieIlXiGWEqq6i6/sk4l9q1YemZF2XVVKnml1F4iCMtNKsR4
MGSa1gA8s4iQbsKNWPgp7M3a51JCVCu6l/8zTpA+uUGapw4tWCp4o0dpIvDPBEa9
b/aF/ygcR8mh5hgUfpF9IpXdknOsbKCvM9lSSfRciETykZc4wrRCVGhlIEFuZHJv
aWQgT3BlbiBTb3VyY2UgUHJvamVjdCA8aW5pdGlhbC1jb250cmlidXRpb25AYW5k
cm9pZC5jb20+iGAEExECACAFAknnWD4CGwMGCwkIBwMCBBUCCAMEFgIDAQIeAQIX
gAAKCRDorT+BmrEOeNr+AJ42Xy6tEW7r3KzrJxnRX8mij9z8tgCdFfQYiHpYngkI
2t09Ed+9Bm4gmEO5Ag0ESedYRBAIAKVW1JcMBWvV/0Bo9WiByJ9WJ5swMN36/vAI
QN4mWRhfzDOk/Rosdb0csAO/l8Kz0gKQPOfObtyYjvI8JMC3rmi+LIvSUT9806Up
hisyEmmHv6U8gUb/xHLIanXGxwhYzjgeuAXVCsv+EvoPIHbY4L/KvP5x+oCJIDbk
C2b1TvVk9PryzmE4BPIQL/NtgR1oLWm/uWR9zRUFtBnE411aMAN3qnAHBBMZzKMX
LWBGWE0znfRrnczl5p49i2YZJAjyX1P2WzmScK49CV82dzLo71MnrF6fj+Udtb5+
OgTg7Cow+8PRaTkJEW5Y2JIZpnRUq0CYxAmHYX79EMKHDSThf/8AAwUIAJPWsB/M
pK+KMs/s3r6nJrnYLTfdZhtmQXimpoDMJg1zxmL8UfNUKiQZ6esoAWtDgpqt7Y7s
KZ8laHRARonte394hidZzM5nb6hQvpPjt2OlPRsyqVxw4c/KsjADtAuKW9/d8phb
N8bTyOJo856qg4oOEzKG9eeF7oaZTYBy33BTL0408sEBxiMior6b8LrZrAhkqDjA
vUXRwm/fFKgpsOysxC6xi553CxBUCH2omNV6Ka1LNMwzSp9ILz8jEGqmUtkBszwo
G1S8fXgE0Lq3cdDM/GJ4QXP/p6LiwNF99faDMTV3+2SAOGvytOX6KjKVzKOSsfJQ
hN0DlsIw8hqJc0WISQQYEQIACQUCSedYRAIbDAAKCRDorT+BmrEOeCUOAJ9qmR0l
EXzeoxcdoafxqf6gZlJZlACgkWF7wi2YLW3Oa+jv2QSTlrx4KLM=
=Wi5D
-----END PGP PUBLIC KEY BLOCK-----
```

Now we will do a generic make, this will take a while...the faster your workstation the faster this process.

Make sure you are in android/system

cd ~/android/system

and type
make
(I suggest just letting this finish before moving on to the next step.)

Eclipse setup/configuration

Now to install and configure Eclipse.

From
http://www.eclipse.org/downloads/http://www.eclipse.org/downloads/http://www.eclipse.org/downloads/http://www.eclipse.org/downloads/ , download an older version of Eclipse Classic, version 3.5.2 is used in this text. Make sure to download the appropriate version (32bit or 64bit depending on your setup). I checked android open source project and they still recommend 3.5.x vs. 3.6.
http://www.eclipse.org/downloads/packages/release/galileo/sr2
should take you to the older version (3.5.2)

for 32bit:
cd ~
tar -zxvf ~/Downloads/eclipse-SDK-3.5.2-linux-gtk.tar.gz

for 64bit:
cd ~
tar -zxvf ~/Downloads/eclipse-SDK-3.5.2-linux-gtk-x86_64.tar.gz

Right click on the desktop and select "Create Launcher", use whatever Name: (Eclipse is fine) and in the area for command, click the browse button and find your Eclipse directory and choose the eclipse application in that folder and click open. Click OK.

This is a desktop icon to launch Eclipse, double click it to start Eclipse.

Click OK to accept the default workspace, when it loads, from the menus select "help" and then "install new software".
Click "add" in the top right corner of the window that opens and add the following.

Name: android-sdk
Location: https://dl-ssl.google.com/android/eclipse
click OK.

Click the dropdown box next to add and choose android-sdk
Click the box next to developer tools then click next.
Android DDMS and Android Development Tools should show, click Next and then agree to the terms and click Finish. (you may get a warning you are installing software that is unsigned. Allow it. Click Restart Now.

When Eclipse restarts..(ok default home space)
From the menus, select "Window" then "Preferences" then "android"
It will prompt you to set the path to your SDK and ask if you want to provide Google with information. Browse to your SDK directory (should be under the android directory and named sdk) and select it. Click OK. Click OK again.

Next from the menus, select "Window", "android sdk avd manager", in the pop up window click on Available Packages and check the box beside the link shown and then click "Install Selected".
Next window pop-up you need to check "Accept All" then "Install". This will take some time as well as it is downloading all the SDK files for all version of Android.

When done....it will ask if it can restart because a service for ADB needs to restart. Restart and then click "close", then close the window (click the x for the Android SDK and AVD Manager).

Our "make" from the source is done, Eclipse is installed and the Android SDK is downloaded, open a terminal and type...

cd ~/android/system
cp development/ide/eclipse/.classpath .
chmod u+w .classpath

This is a "get you started" file so we need to edit it.

gedit .classpath

fine line 109 (it should be what I have listed below)

<classpathentry kind="lib" path="out/target/common/obj/JAVA_LIBRARIES/**google**-common_intermediates/javalib.jar"/>

and change it to this..

<classpathentry kind="lib" path="out/target/common/obj/JAVA_LIBRARIES/**android**-common_intermediates/javalib.jar"/>

Delete line 110 (delete the empty line so that 111 becomes 110)

Now we modify Eclipse settings to run better for Android.

cd ~/eclipse
gedit eclipse.ini

We are changing the memory values used by Eclipse so it will function better.

find these values..

-Xms40m
-Xmx256m

and change them to these values..

-Xms128m
-Xmx512m

Save and close the file.

Open Eclipse (ok the default workspace again) and from the menus click File, New, Project then select Java Project. click Next.

Check "Create Project from existing source", click Browse and navigate to the ~/android/system folder. Click OK
(when selected should now show Location: /home/(your system)/android/system with system as the Project name:)

Click Finish. This will take a while as Eclipse loads and check the entire source tree for android.

You will have 2 errors, in the source tree listing on the left, click the |> next to system to open the contents of source and scroll down to see the two errored sources (packages/providers/CalendarProvider/src and packages/providers/ContactsProvider/src)

1. In the Eclipse Package Explorer expand the folder: packages/providers/CalendarProvider/src (click the |> next to it)

2. right click on the package com.android.providers.calendar, this brings up a context menu

3. select "new" then "file"

4. bottom left: click the "Advanced >>" button, this exposes a checkbox: "Link to file in the file system"

5. check that checkbox. This makes the "Browse..." button become active.

6. click the "Browse..." button, browse to and select the file (e.g., out/target/common/obj/APPS/CalendarProvider_intermediates/src/src/com/android/providers/calendar/EventLogTags.java)

7 click OK (in "Select Link Target" window)

8. click Finish (in "New File")

9. Repeat this for packages/providers/ContacsProvider/src but browse to out/target/common/obj/APPS/ContactsrProvider_intermediates/src/src/com/android/providers/contacts/EventLogTags.java)

You will be left with just warning. Exit Eclipse.

Creating an Emulator to run our generic "make" rom

The next section is not required so you can skip if you choose:

You've compiled the source for a generic rom already. You can load it in the emulator if you wanted to see what is what. You will need to create a "Virtual Device" first and copy the generic system.img to that Virtual Device's configuration folder.

Browse to android/sdk/tools and find android. Double click it....click "run in terminal" and it will open the Android SDK and AVD Manager. Select Virtual Devices and click 'New"
Name it whatever (Generic) but no spaces in the name. I'll example it with Generic..

Name: Generic
Target: Google APIs (Google Inc.) - API Level 8
SD Card: 1024 Mib (1 gig, you can use smaller)
Skin: Built-in: WVGA854
Hardware: Add it all
(click new, ok until there are no more choices)
change...
Device ram size - 256
Track-ball support - no
Click Create AVD, it will take a while as it writes out a 1gig empty file for the "sd card" emulation.

Next you'll want to copy the system.img file from ~/android/system/out/target/product/generic
click on system.img and select copy.

Browse to your home folder, click "view" select "show hidden files" and then browse to .android, avd, the name of your Virtual Device (Generic.avd) and right click in the folder and select paste.
You should see now
config.ini
sdcard.img
system.img
userdate.img

Go to the Android SDK and AVD Manager and click on your Virtual Device (Generic) and select Start...

This will take a while....
Hint...think of the mouse pointer as your finger tip and use it that way, clicking the mouse is touching the screen so click and hold it down for how you would be doing long presses or dragging icons, etc..
You can look in Menu/Settings/About Phone to see your signature signing.....your user name, master-key # etc...

Okay...enough of that...back to working on making this work on your Motorola Droid. Just close the emulator, shutting down takes forever....

Basic Device/Vendor Setup (Motorola Droid)

We are using Koush's setup for the Motorola Droid. This will create the device we need and his scripts will extract the proprietary files from your phone and create the appropriate vendor setup.

First, in the terminal.

cd ~/android/system/device
git clone
http://github.com/koush/android_device_generic.githttp://github.com/koush/android_device_generic.githttp://github.com/koush/android_device_generic.git generic

(this creates the folder generic under android/system/device and clones the info from Koush's generic on github.)

Next type..

mkdir motorola
cd motorola
git clone
http://github.com/koush/android_device_motorola_sholes.githttp://github.com/koush/android_device_motorola_sholes.githttp://github.com/koush/android_device_motorola_sholes.git sholes

(this is cloning the device setup for the Motorola Droid (sholes) under android/system/device/motorola/sholes)

Now we need the proprietary files from your phone, connect it to the USB on your computer.

cd sholes
. extract-files.sh

This will extract the files and create a folder /vendor/motorola under android/system.
It is best to use a rooted stock FRG22D phone to do the extraction on. Watch the file list process, if any error, make a note of the location and manually copy it from your phone (root explorer works great) or extract the file from the download of a Deodexed FRG22D.zip file.
Disconnect your phone from the USB.

Next we need to remove some libraries being called that aren't part of the stock AOSP from koush's BoardConfig.mk file.

gedit BoardConfig.mk

find line 72 and 74 and comment them out by adding # at the start of those two lines.

TARGET_RECOVERY_UI_LIB := librecovery_ui_sholes librecovery_ui_generic

TARGET_RECOVERY_UPDATER_LIBS += librecovery_updater_generic

become

#TARGET_RECOVERY_UI_LIB := librecovery_ui_sholes librecovery_ui_generic

#TARGET_RECOVERY_UPDATER_LIBS += librecovery_updater_generic

save and exit.

With all the files in place we are ready to compile...and remember the next commands..you will use them each time you start another compile...

cd ~/android/system
. build/envsetup.sh
lunch generic_sholes-userdebug (or type lunch, press enter and type the number matching generic_sholes-userdebug)

make

If this compiles successfully you can do a

make -jx **otapackage**
(substitute x with twice the number of processor/cores you have, for me with a dual core setup, what I type is make -j**4** otapackage)

Do NOT flash this to your phone. We are just making sure everything is working properly. To verify, look in ~/android/system/out/target/product/sholes for a file named
We will make it so you at least have Superuser + root before you install it to your phone.

Adding Superuser + root to your build

We need to add the Superuser app as well as su to our builds to allow root and control of root.

cd ~/android/system/packages/apps
git clone git://github.com/SnkBitten/android_packages_apps_Superuser.git Superuser

This the Superuser app, now we need to modify su for root access.

gedit ~/android/system/system/extras/su/Android.mk

in the text box that opens, highlight everything and delete it and then copy and paste the following.

ifeq ($(BUILD_ORIGINAL_SU),true)

LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES:= su.c

LOCAL_MODULE:= su

LOCAL_FORCE_STATIC_EXECUTABLE := true

LOCAL_STATIC_LIBRARIES := libc

LOCAL_MODULE_PATH := $(TARGET_OUT_OPTIONAL_EXECUTABLES)
LOCAL_MODULE_TAGS := debug

include $(BUILD_EXECUTABLE)

endif

save and close. All builds will now have Superuser app and root (su).

Adding Busbybox to the build

Busybox, thanks to modification by cvpcs, is now able to be compiled without making any changes to the android source, making it a truly separate module that can be added to any android build.

We need two repositories added to our system, busybox and libc-ext.

cd ~/android/system/external

git clone git://github.com/cvpcs/android_external_busybox.git busybox
git clone git://github.com/cvpcs/android_external_libc-ext.git libc-ext

and now we need to edit a file to include the libc-ext for this to work.

gedit ~/android/system/build/core/prelink-linux-arm.map

In the text box that opens, add the following statement (should be at line 23, place your cursor at the start of line 23 and press enter, then copy the following to line 23).

```
libc-ext.so             0xAFC80000 # [<64K]
```

Your file after you do the above should look similar to below....lines 22, 23 and 24.

```
libc.so                 0xAFD00000 # [~2M]
libc-ext.so             0xAFC80000 # [<64K]
libstdc++.so            0xAFC00000 # [<64K]
```

once both repositories are cloned and libc-ext is added to the pre-link map, any further builds will have a fully functional busybox with symlinks and all included in the build.

IMPORTANT

There is a very important step to do after our final build. Let's go through the steps to compile our package .zip file and make it ready to flash to the phone.

cd ~/android/system
. build/envsetup.sh
lunch generic_sholes-userdebug
make -j2 otapackage

(remember you can replace the -j2 with twice the number of CPU you have to speed things up)

Wait until the build is finsihed then browse to the android/system/out/target/product/sholes and look for your **generic_sholes-ota-eng.(your login name).zip**.
My login is tcrews so mine is generic_sholes-ota-eng.tcrews.zip

Double click the .zip file and delete the following:
the recovery foler
check_prereq

You'll also need to edit the build.prop file in the .zip that sits in the system folder.
double click system
double click build.prop (this will open it in gedit)
look for (mine was line 17)

ro.product.name=generic_sholes

and change it to

ro.product.name=voles

save and exit, then close the .zip file.
If you use Rom Manger you can just copy it to your SD Card and then use "Install Rom from SD Card" and select it.

Otherwise (using SPRecovery) you need to rename it to update.zip and install it that way.

Note ** it will not be a "usable" rom, no market, no gmail, no maps...nothing related to google as those are not source but proprietary files. From what I've heard...most just copy the google apps from the FRG22D (deodexed version).zip file to the app folder in their rom. I never tried that as I moved on to creating my own repo and vendor/device setups and working it that way. You could also download one of the gapps.zip packages from cyanogen, shadowrom, cvpcs and install that after you install your ROM.

You will also want to look at what you need to do to modify your build.prop file to emulate a valid build.prop file (for market purposes....you read as a test build, not a release).

THANK YOU!

Thanks to all below for making this guide possible.

CVPCS

Sniffle

Koush

InsaneNemesis

http://howto.ccroms.net/

http://android.snkbitten.com/