



## Single Board Computer On



Qualcomm®  
**snapdragon**

## SKATE-212 Build Manual

Rev.2.0

28<sup>th</sup> August 2018



## Revision History

Rev. No.	Date	Author	Description
1.0	March 29, 2017	Software Dev. Team	Initial Release
2.0	Aug 28, 2018	Software Dev. Team	Updated for new manifest release

**PROPRIETARY NOTICE:** This document contains proprietary material for the sole use of the intended recipient(s). Do not read this document if you are not the intended recipient. Any review, use, distribution or disclosure by others is strictly prohibited. If you are not the intended recipient (or authorized to receive for the recipient), you are hereby notified that any disclosure, copying distribution or use of any of the information contained within this document is STRICTLY PROHIBITED.

### **Disclaimer**

Kemsys Technologies reserves the right to change details in this publication including but not limited to any Product specification without notice.

No warranty of accuracy is given concerning the contents of the information contained in this publication. To the extent permitted by law no liability (including liability to any person by reason of negligence) will be accepted by Kemsys Technologies, its subsidiaries or employees for any direct or indirect loss or damage caused by omissions from or inaccuracies in this document.

### **Trademarks**

All registered trademarks and product names mentioned in this document belongs to the respective companies and they are used here for representation purposes only.

### **Technical Support**

Kemsys Technologies technical support team is committed to provide the best possible support to our customers at our best. Normal reply will be sent within 24 hours and queries which requires recreation for a solution will consume more than 24 hours. Customer will be notified in such delays.

For immediate assistance contact us at,

For Sales related Queries : [salesEDS@kemsys.com](mailto:salesEDS@kemsys.com)

Website : [www.kemsys.com](http://www.kemsys.com)

## Contents

<b>Revision History .....</b>	<b>2</b>
<b>1. Introduction .....</b>	<b>6</b>
<b>2. Scope of the document .....</b>	<b>6</b>
<b>3. Installation and Setup .....</b>	<b>6</b>
3.1 Required Equipment and Software .....	6
3.2 Install Ubuntu .....	6
3.3 Configure Samba for Windows sharing (optional) .....	<b>Error! Bookmark not defined.</b>
3.4 Install JDK .....	7
3.5 Install repo .....	8
<b>4. Downloading and Building the Software .....</b>	<b>9</b>
4.1 Introduction to <a href="http://www.codeaurora.org">www.codeaurora.org</a> .....	9
4.2 Downloading HLOS software .....	9
4.3 Downloading and Building the Software using script file .....	11
4.4 Rebuild apps processor Android HLOS .....	11
<b>5. Configuring Android adb, fastboot, and Flashing Images to Target Device .....</b>	<b>12</b>
5.1 Configure adb and fastboot in Linux .....	12
5.2 Program Android images using fastboot .....	13
<b>6. More on adb Usage and Android Build .....</b>	<b>14</b>
6.1 Install applications on Android target using adb .....	14
6.2 Android source tree structure .....	14
6.3 Android device tree structure .....	15
6.4 Build the Linux kernel manually .....	16
6.5 Build the Android components manually .....	16
6.6 Other useful Android build commands .....	17

## List of Tables

Table 1 : Abbreviations .....	5
-------------------------------	---

*Table 1: Abbreviations*

SBC	Single Board Computer
LPDDR	Low Power Double Data Rate memory
OTG	On-The-Go
eMMC	embedded Multi-Media Controller
SD Card	Secure Digital Card
RDS	Radio Data System
RBDS	Radio Broadcast Data System
BLE	Bluetooth Low Energy
GPU	Graphics Processing Unit
GPS	Global Positioning System
GLONASS	Global Navigation Satellite System
GNSS	Global Navigation Satellite System
QZSS	Quasi-Zenith Satellite System
DSI	Display Serial Interface
CSI	Camera Serial Interface
TBD	To Be Decided
DNI	Do Not Import
DNP	Do Not Populate
ADB	Android Debug Bridge

## 1. Introduction

The new SKATE-212, is a member from Kemsys Technologies. SKATE BOARD family of Single Board Computers and is powered by the Qualcomm® Snapdragon™ 212 series application processor (APQ8009). It is an ultra-small, slightly bigger than credit-card size (90mm x 70mm). SKATE-212 development kit features quad core Arm Cortex A7 class computing with easy access to industry standard I/O's which creates the perfect environment for a variety of Android/Linux based applications including digital signage, industrial automation and video conferencing. Unique features include on-board support for WiFi/BLE, GPS, HDMI displays, dual-MIPI-CSI cameras, MIPI-DSI, and RJ45 interfaces.

## 2. Scope of the document

This document describes how to obtain, build, and program software applicable to the APQ8009 Linux Android Software Product (SP) “as-is” into a SKATE-212 platform including:

- Setting up a development environment and installing the software
- Building the software and flashing it onto a SKATE-212 platform
- Please also refer to <https://source.android.com/source/initializing.html> to find the most recent set-up instructions from Google on how to build Android source code on Ubuntu machines.

## 3. Installation and Setup

### 3.1 Required Equipment and Software

S. No	Item Description	Version	Source/Vendor	Purpose
1	Ubuntu 16.04 LTS Linux distribution for 64-bit architecture	16.04 LTS	Ubuntu community/Canonical, Ltd.	Android Build host os. Note that google may recommend other Ubuntu newer versions as well, please keep up-to date as per the link <a href="https://source.android.com/source/initializing.html">https://source.android.com/source/initializing.html</a>
2	Java SE JDK for Linux x64	8	Oracle	Building Android
3	Repo	-	Android Open Source Project	Android source management tool

### 3.2 Install Ubuntu

**IMPORTANT! You must be able to log in as root or use sudo to have root permissions during the installation.**

1. Create an installation CD and install it onto the computer by following the instructions at <http://releases.ubuntu.com/>.

2. After installation, perform a software update using one of the following options:

- a. Use the GUI, select System > Administration > Update Manager, OR
- b. Use the shell command line and edit the source config file directly, as follows:
  - I. `sudo vi /etc/apt/sources.list`
  - II. Edit the file to enable the universe and multiverse sources, and disable the Ubuntu installation CD source.
  - III. From the command line, perform the package list update and package upgrades:  
`sudo apt-get update`  
`sudo apt-get upgrade`

3. Use `apt-get` to install the additional required packages.

```
sudo apt-get install git-core gnupg flex bison gperf build-essential zip curl zlib1g-dev  
libc6-dev lib32ncurses5-dev x11proto-core-dev libx11-dev lib32readline6-dev lib32z-dev libgl1-  
mesa-dev g++-multilib tofrodos python-markdown libxml2-utils xsltproc
```

4. **IMPORTANT!** Make **bash** the default shell using one of the following options (Android build scripts contain bash shell dependencies that require the system default shell **/bin/sh** to invoke bash).

- a. Reconfigure the package:
  - i. Use the command:  
`sudo dpkg-reconfigure dash`
  - ii. Answer “no” when the options for the above command pop up on your screen.
- b. Manually change the symlink `/bin/sh->dash` to `/bin/sh->bash` using the following commands:  
`sudo rm /bin/sh`  
`sudo ln -s /bin/bash /bin/sh`

NOTE: See the Ubuntu Wiki page at <https://wiki.ubuntu.com/DashAsBinSh> for more information.

### 3.3 Install JDK

The Sun JDK is no longer in Ubuntu’s main package repository. To download it, add the appropriate repository and indicate to the system about the JDK being used.



```
sudo add-apt-repository ppa:openjdk-r/ppa  
sudo apt-get update
```

For latest Android releases you might need to use openjdk8, please install as below:

```
sudo apt-get install openjdk-8-jdk
```

### 3.4 Install repo

The repo tool is a source code configuration management tool used by the Android project. The repo tool is a front end to git written in Python. It uses a manifest file to help download code organized as a set of projects that are stored in different git repositories.

To install repo, do the following:

1. Create a ~/bin directory in your home directory, or, if you have root or sudo access, install for all system users under a common location, such as /usr/local/bin or somewhere under /opt.
2. Download the repo script.

```
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
```

3. Set the repo script attributes to executable.

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

4. Include the installed directory location for repo in your PATH.

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

5. Run repo --help to verify the installation.

```
repo --help
```

You should see a message similar to the following:

```
usage: repo COMMAND [ARGS]  
repo is not yet installed. Use "repo init" to install it here.
```

The most commonly used repo commands are as follows:

- init -> Install repo in the current working directory
- help -> Display detailed help on a command



**NOTE:** For access to the full online help, install repo (repo init).

## 4. Downloading and Building the Software

### 4.1 Introduction to [www.codeaurora.org](http://www.codeaurora.org)

Open source HLOS (High Level Operating System) software for Qualcomm® Snapdragon™ chipsets is available on the Linux Foundation hosted site [www.codeaurora.org](http://www.codeaurora.org). The abbreviation “HLOS” is used in this document.

Specifically for the Android platform, go to <https://www.codeaurora.org/xwiki/bin/QAEP/release>, which can be accessed through Wiki Home > Android for MSM™ project > Android releases.

### 4.2 Downloading HLOS software

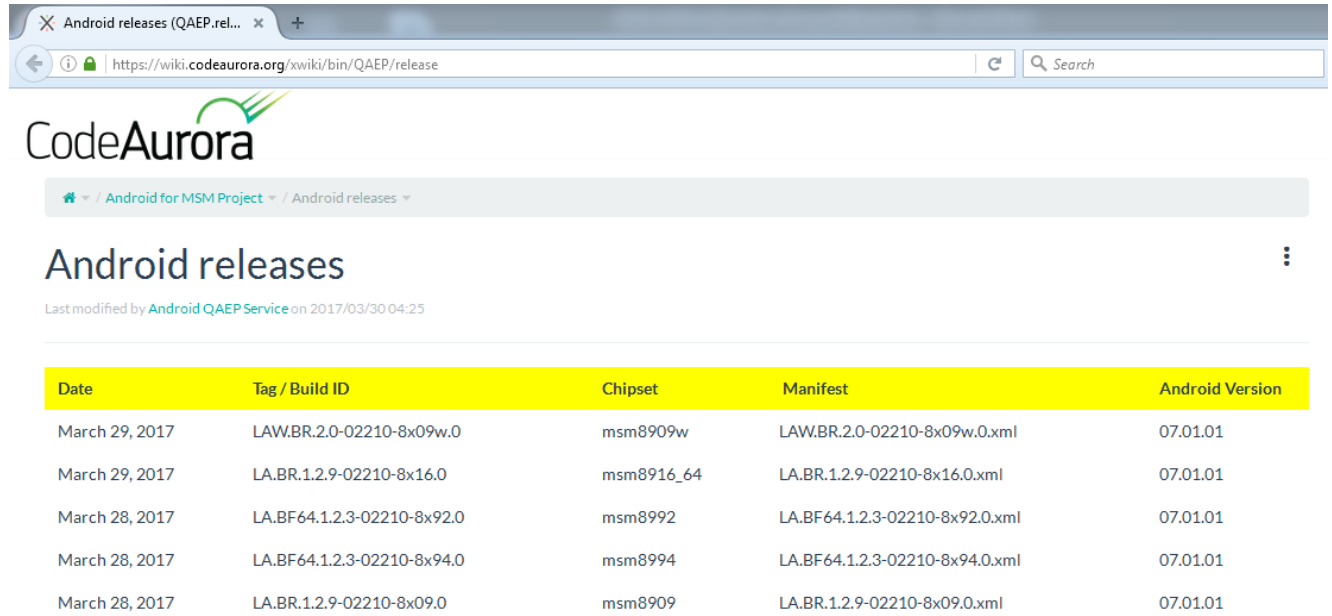
If you want to download Android source code that will work on your SKATE-212, these are the instructions you should use. To download the HLOS software, do the following:

1. First, you need to decide what release you would like to download. For instance, if you downloaded

**SKATE-212\_Android\_BSP\_LA.BR.1.2.9-04110-8x09.0-1\_Ver\_1.0.zip**

then you'll look for a manifest called **LA.BR.1.2.9-04110-8x09.0.xml**

This series of letters and numbers is called the APSS (Application processor subsystem software) Build ID.



Date	Tag / Build ID	Chipset	Manifest	Android Version
March 29, 2017	LAW.BR.2.0-02210-8x09w.0	msm8909w	LAW.BR.2.0-02210-8x09w.0.xml	07.01.01
March 29, 2017	LA.BR.1.2.9-02210-8x16.0	msm8916_64	LA.BR.1.2.9-02210-8x16.0.xml	07.01.01
March 28, 2017	LA.BF64.1.2.3-02210-8x92.0	msm8992	LA.BF64.1.2.3-02210-8x92.0.xml	07.01.01
March 28, 2017	LA.BF64.1.2.3-02210-8x94.0	msm8994	LA.BF64.1.2.3-02210-8x94.0.xml	07.01.01
March 28, 2017	LA.BR.1.2.9-02210-8x09.0	msm8909	LA.BR.1.2.9-02210-8x09.0.xml	07.01.01

Figure 1 Code Aurora Forum releases page

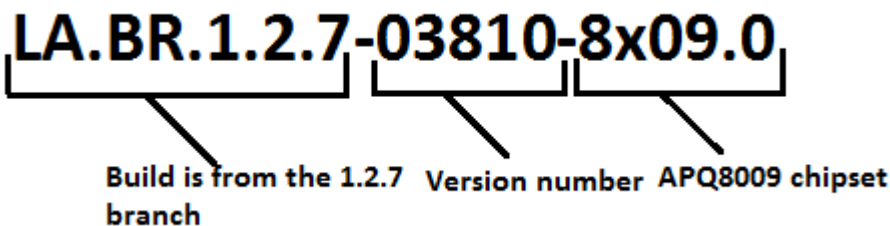
Each Tag/Build ID as shown in Figure 1 is identified by a unique build identification code which follows this naming convention:

<PL\_Image>-<Version>-<Chipset>

Where:

- PL\_Image – LNX.LA.Branch for Linux Android
- Version – Variable number of digits used to represent the build ID version.
- Chipset – 8x09 for MSM8909/APQ8009

**LA.BR.1.2.7-03810-8x09.0**



Build is from the 1.2.7 Version number APQ8009 chipset branch

Figure 2 Build ID code example

**NOTE:** This example is for reference only and does not specify the exact Build ID for SKATE-212.

2. The APPS Build ID TAG can be used to find the appropriate open source release manifest XML file from the cross reference table at

<https://www.codeaurora.org/xwiki/bin/QAEP/release>

### 4.3 Downloading and Building the Software using script file

- 1) Make sure you have at least ~60GB of free space, then download the [SKATE-212\\_Android\\_BSP\\_LA.BR.1.2.9-04110-8x09.0-1\\_Ver\\_1.0.zip](#) file
- 2) This package has the [skate-212\\_build\\_ver1.0.sh](#), Proprietary tar file, README.txt, source patches and a license file.
- 3) After unzipping, run the command `chmod a+x skate-212_build_ver1.0.sh` and run bash shell using the bash command at the Linux prompt. Then run the build script as [./skate-212\\_build\\_ver1.0.sh](#) or ["sh skate-212\\_build\\_ver1.0.sh"](#) in a bash shell.
  - a) Accept your name and email ID, when prompted, for the clone to begin. The script `skate-212_build_ver1.0.sh` clones the code from [www.codeaurora.org](#), uses the files in the Linux Android Board Support Package that includes Proprietary-XXXX-x.tgz, and applies relevant patches, if needed.

**NOTE:** Where XXXX-x is the "www.codeaurora.org build manifest ID"- "version number"

- b) Source code clone and build takes about an hour or longer depending on the configuration of the Ubuntu machine. You can edit the `skate-212_build_ver1.0.sh` for build command and change

[make -j4](#)

or

[make -j8](#) depending on your Ubuntu machine configuration.

**NOTE:** Sometimes the clone from [www.codeaurora.org](#) may fail due to network issues; please try again.

- 4) Once the build is completed, go to step 5 for flashing the images to the SKATE-212 device.

### 4.4 Rebuild apps processor Android HLOS

To build the apps processor Android HLOS, do the following:

1. In a bash shell, navigate to the Android source tree base directory.  
[cd \\$BUILDROOT](#)
2. Enter the following command to configure the build environment shell settings:  
[source build/envsetup.sh](#)

**NOTE:** You must use the source command so the environment settings are defined in the current shell.

3. Enter the lunch command to select the build configuration, or enter it with no parameters to see an interactive menu for making selections.

*lunch msm8909-userdebug*

4. Run make to start the build. To run parallel builds for faster build times on a multicore build machine, run the make command as follows:

<i>make -j8</i>	to build everything
<i>make -j8 aboot</i>	to build LK bootloader emmc_appsboot.mbn
<i>make -j8 bootimage</i>	to build kernel boot.img
<i>make -j8 clean</i>	cleans the out directory completely

For user space components standard, running with “*mm*” and “*mm -B*” (to rebuild) also works. For more information, see Chapter 4.

Sometimes for certain changes which may not take effect in kernel, it may be useful to manually remove kernel obj files:

*lunch msm8909-userdebug*  
*rm -rf out/target/product/msm8909/obj/KERNEL\_OBJ/*  
*rm out/target/product/msm8909/boot.img*  
*make -j8 bootimage*

Output is generated in \$BUILDROOT/out/target/product/msm8916/. The list of files is as follows:

- emmc\_appsboot.mbn
- boot.img
- cache.img
- userdata.img
- system.img
- persist.img
- recovery.img

## 5. Configuring Android adb, fastboot, and Flashing Images to Target Device

### 5.1 Configure adb and fastboot in Linux

1. To configure adb, modify the USB driver by navigating to the following directory:

*cd /etc/udev/rules.d/*

2. Enter the command:

*sudo vi 50-android.rules*

3. Copy the below lines to the file and save it.

*#Sooner low-level bootloader*

```
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="d00d",MODE="0664",  
GROUP="plugdev"
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="2717", ATTR{idProduct}=="ff68",MODE="0664",  
GROUP="plugdev"
```

```
# adb composite interface device 9091
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="05C6", ATTR{idProduct}=="9091", MODE="0664",  
GROUP="plugdev"
```

```
# adb composite interface device 0169
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="0FCE", ATTR{idProduct}=="0169", MODE="0664",  
GROUP="plugdev"
```

```
# adb composite interface device 9025
```

```
SUBSYSTEM=="usb", ATTR{idVendor}=="05C6", ATTR{idProduct}=="9025", MODE="0664",  
GROUP="plugdev"
```

4. After editing the file, to see the list of target devices connected to the Linux box, type:  
*lsusb*

5. After a build is complete, the adb and fastboot executables for Linux are located in the android\out\host\linux-x86\bin directory in the Android software release. If the android\out\host\linux-x86\bin directory is not in the executable search path, use the following steps to add it from \$BUILDROOT (build directory).

Run the below commands:

```
source build/envsetup.sh  
lunch msm8909-userdebug
```

**NOTE:** To run adb or fastboot, sudo or root access on the Linux machine may be required. This is a useful link for Linux and MAC OS to setup adb and fastboot (note that device ID "9025" is needed in 50-android.rules or 51-android.rules) : <https://code.google.com/p/adb-fastboot-install/>

## 5.2 Program Android images using fastboot

1. Hold down the VOL- key, connect the DC supply to the SKATE-212, and power up the board. Plug the USB cable into the target. This will bring the device into fastboot mode.

2. Navigate to the following directory:

```
cd <$BUILDDROOT>/out/host/linux-x86/bin
```

```
sudo ./fastboot devices
```

The list of registered devices is displayed.

3. Once the device is detected, flash the binaries to the target. Run the following commands to flash all the android apps images:

```
fastboot flash aboot <path to emmc_appsboot.mbn>
```

```
fastboot flash boot <path to boot.img>
```

```
fastboot flash system <path to system.img>
```

```
fastboot flash userdata <path to userdata.img>
```

```
fastboot flash persist <path to persist.img>
```

```
fastboot flash recovery <path to recovery.img>
```

4. Reboot the board.

NOTE: When re-flashing Android Images, the initial boot will take up to 3 minutes as Android initializes user space databases and files. This boot time will reduce to less than a minute on subsequent boots.

## 6. More on adb Usage and Android Build

### 6.1 Install applications on Android target using adb

1. Plug the USB cable into the target.

2. Navigate to the following directory:

```
cd <$BUILDDROOT>/out/host/linux-x86/bin
```

3. Enter the command below to register a device:

```
sudo ./adb devices
```

4. Push the files as follows:

```
./adb push AppName.apk /system/app/.
```

or

```
./adb install AppName.apk
```

NOTE: In general, the syntax is: `adb push <file_name> <location_on_the_target>`

### 6.2 Android source tree structure

The Android device tree structure, for example, the <Android device tree root>, is laid out as follows:

**build/** – Build environment setup and makefiles

**bionic/** – Android C library

**dalvik/** – Android JVM

**kernel/** – Linux kernel framework/

– Android platform layer (system libraries and Java components)

**system/** – Android system (utilities and libraries, fastboot, logcat, liblog)

**external/** – Non-Android-specific Open Source projects required for Android

**prebuilt/** – Precompiled binaries for building Android, e.g., cross-compilers

**packages/** – Standard Android Java applications and components

**development/** – Android reference applications and tools for developers

**hardware/** – HAL (audio, sensors) and Qualcomm specific hardware wrappers

**vendor/qcom/** – Qualcomm target definitions, e.g., msm8909

**vendor/qcom/proprietary/** – Qualcomm-proprietary components

**out/** – Built files created by user

**out/host/** – Host executables created by the Android build

**out/target/product/<product>** – Target files appsboot\*.mbn – Applications boot loader

**boot.img** – Android boot image (Linux kernel + root FS)

**system.img** – Android components (/system)

**userdata.img** – Android development applications and database

**root/** – Root FS directory, which compiles into ramdisk.img and merged into boot.img

**system/** – System FS directory, which compiles into system.img

**obj/** – Intermediate object files

**include/** – Compiled include files from components lib/ STATIC\_LIBRARIES/

SHARED\_LIBRARIES / EXECUTABLES / APPS /

**symbols/** – Symbols for all target binaries

### 6.3 Android device tree structure

The Android source tree structure is laid out as follows:

**/** – Root directory (ramdisk.img, read-only)

**init.rc** – Initialization configuration files (device config, service startups) init.qcom.rc

**dev/** – Device nodes

- proc/** – Process information
- sys/** – System/kernel configuration
- sbin/** – System startup binaries (adb daemon; read-only)
- system/** – From system.img (read-write) – bin/ – Android system binaries
  - **lib/** – Android system libraries
  - **xbin/** – Nonessential binaries
  - **framework/** – Android framework components (Java)
  - **app/** – Android applications (Java)
  - **etc/** – Android configuration files
- sdcard/** – Mount point for SD card
- data/** – From userdata.img (read-write)
  - **app/** – User installed Android applications
  - **tombstones/** – Android crash logs

## 6.4 Build the Linux kernel manually

1. Set up the Android build environment. From source root directory type the below commands:

```
source build/envsetup.sh  
lunch msm8909-userdebug
```

2. *make -j8 bootimage*

NOTE: In theory, it is possible to use `-jn` as long as 'n' is smaller than the number of processors in the server where the build is being created.

3. To start with a clean tree, use the following commands:

- a. To remove object files:

```
make -j8 clean
```

you can also use the below command to delete the kernel object files and run the build command again:

```
rm -rf out/target/product/msm8909/obj/KERNEL_OBJ/
```

## 6.5 Build the Android components manually

1. Set up the Android build environment (*envsetup.sh/lunch*).
2. Change to the main Android directory.
3. Build with the make command:

```
make -j4
```
4. To build individual components, choose one of the following options:
  - a. To run make from the top of the tree, use the command:



*m <component name>* # Example: *m libril-qc-1*

b. To build all of the modules in the current directory, change to the component directory and use the command:

*mm*

5. To delete individual component object files, choose one of the following options:

a. To delete a particular module, use the command:

*m clean-<module name>*

b. To delete a module within a given path, use the commands:

*rm -rf out/target/product/\*/obj/STATIC\_LIBRARIES/<module name>\_intermediates*

*rm -rf out/target/product/\*/obj/SHARED\_LIBRARIES/<module name>\_intermediates*

*rm -rf out/target/product/\*/obj/EXECUTABLES/ < module name >\_ intermediates*

## 6.6 Other useful Android build commands

Other important Android build commands are as follows:

- **printconfig** - Prints the current configuration as set by the choosecombo commands.
- **m** – Runs make from the top of the tree. This command is useful because the user can run make from within subdirectories.

If you have the TOP environment variable set, the commands use that. If you do not have the TOP variable set, the commands look up the tree from the current directory, trying to find the top of the tree.

- **mm** – Builds all of the modules in the current directory.
- **mmm** – Builds all of the modules in the supplied directories.
- **croot** – cd to the top of the tree.
- **sgrep** – grep for the regex you provide in all .c, .cpp, .h, .java, and .xml files below the current directory.
- **clean-\$(LOCAL\_MODULE)** and **clean-\$(LOCAL\_PACKAGE\_NAME)**. Let you selectively clean one target.

For example, if you type `make clean-libutils`, it deletes libutils.so and all of the intermediate files. If you type `make clean-Home`, it cleans just the Home application.

- **make clean** – Cleanly deletes all of the output and intermediate files for this configuration.

This is the same as *rm -rf out/<configuration>/*.

Android makefiles (Android.mk) are similar to regular GNU makefiles; some differences are:

- Predefined **variables** to assign for source files, include paths, compiler flags, library includes, etc.

- Predefined **action** for compiling executables, shared libraries, static libraries, Android packages, using precompiled binaries, etc.

For example:

- Variables
  - LOCAL\_SRC\_FILES – List of all source files to include
  - LOCAL\_MODULE – Module name (used for “m”)
  - LOCAL\_CFLAGS – C compiler flags override
  - LOCAL\_SHARED\_LIBRARIES – Shared libraries to include
- Action
  - include \$(CLEAR\_VARS) – Clears LOCAL\* variables for the following sections:
    - include \$(BUILD\_EXECUTABLE)
    - include \$(BUILD\_SHARED\_LIBRARIES)
    - include \$(BUILD\_STATIC\_LIBRARIES)

NOTE: Paths in Android.mk are always relative to the Android device tree root directory.

To add a new module to the Android source tree, perform the following steps:

1. Create a directory to contain the new module source files and Android.mk file.
2. In the Android.mk file, define the LOCAL\_MODULE variable with the name of the new module name to be generated from your Android.mk.

NOTE: For Applications modules, use LOCAL\_PACKAGE\_NAME instead.

The local path in your new module is LOCAL\_PATH. This is the directory your Android.mk file is in. You can set it by inserting the following as the first line in your Android.mk file:

```
LOCAL_PATH := $(call my-dir).  
LOCAL_SRC_FILES
```

The build system looks at LOCAL\_SRC\_FILES to find out which source files to compile, .cpp, .c, .y, .l, and/or .java. For .lex and .yacc files, the intermediate .h and .c/.cpp files are generated automatically. If the files are in a subdirectory of the one containing the Android.mk file, you must prefix the files with the directory name:

```
LOCAL_SRC_FILES := \  
    file1.cpp \  
    dir/file2.cpp
```

The new module can be configured with the following:

LOCAL\_STATIC\_LIBRARIES – These are the static libraries that you want to include in your module.

```
LOCAL_STATIC_LIBRARIES := \
```

libutils \  
libtinyxml

LOCAL\_MODULE\_PATH – Instructs the build system to put the module somewhere other than what is normal for its type. If you override this path, make sure that you also set LOCAL\_UNSTRIPPED\_PATH if it is an executable or a shared library so the unstripped binary also has somewhere to go; otherwise, an error occurs.