Qualcomm Technologies, Inc.

DragonBoard™ 410c based on Qualcomm® Snapdragon™ 410E processor

Software Build and Installation Guide, Linux Android

December 2016

Questions or comments: https://www.96boards.org/DragonBoard410c/forum

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

LM80-P0436-2 Rev J

Revision history

Revision	Date	Description
J	December 9, 2016	Fix to Section 1.3, Step 3: text of the command updated.
H	July 13, 2016	Update part number to “E”
G	March 1, 2016	Section 1.1: Added Google Android initialization links; Section 2.1: defined HLOS; changed the Java version needed for Android L build, fixed a couple of minor typos
F	February 2, 2016	Added information in section 2.2 “Downloading HLOS software” to clarify the commands for cloning the Android software
E	December 15, 2015	Added more information for adb fastboot setup for Linux and MacOS
D	June 17, 2015	Updated file & package names.
C	June 15, 2015	Miscellaneous updates.
B	May 22, 2015	Updated Revision history and © date.
A	April 21, 2015	Initial release.

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document describes how to obtain, build, and program software applicable to the Linux Android Software Product “as-is” into a reference platform including:

- Setting up a development environment and installing the software.
- Building the software and flashing it onto the DragonBoard™ 410c 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.

1.2 Required equipment and software

Table 1-1 Required equipment and software

#	Item description	Version	Source/vendor	Purpose
1	Ubuntu 12.0.4 LTS Linux distribution for 64-bit architecture	12.0.4 LTS	Ubuntu Community/ Canonical, Ltd.	Android build host OS. Note that google may recommend other Ubuntu newer versions as well, please keep upto date per the link https://source.android.com/source/initializing.html
2	Windows 7 or Windows XP workstation	Windows 7 or Windows XP	Microsoft	Windows-based programming tools
3	Java SE JDK for Linux x64	7	Oracle	Building Android
4	repo	_	Android Open Source Project	Android source management tool

1.3 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 lib32ncurses6-dev x11proto-core-dev
libx11-dev lib32readline5-dev lib32z-dev libgl1-mesa-dev g++-multilib
mingw32 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←$B"*←(Bdash to /bin/sh←$B"*←(Bbash** 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.

1.4 Configure Samba for Windows sharing (optional)

1. Use the following command to install the Samba server and configuration manager for Windows sharing:

   ```
   sudo apt-get install samba system-config-samba
   ```

2. Configure the Samba server using:

   System->Administration->Samba

   preferences->server settings:

   - vmgroup, security=user authentication
   - encrypt pw=yes, guest accnt=no guest accnt
   - add share directory=/, share name=root, description=root directory

1.5 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 "deb http://archive.canonical.com/ lucid partner"
sudo apt-get update
For latest Android releases you might need to use openjdk7, please install
as below:
sudo apt-get install openjdk-7-jdk
```

1.6 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).

1.7 Acronyms

Acronym	Definition
APQ	Application Processor Qualcomm
APSS	Applications Processor Sub-System
DC	Direct Current
FFA	Form Factor Accurate
FS	File System
GNU	GNU's Not Unix

Acronym	Definition
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
HLOS	High Level Operating System
ID	Identification
JDK	Java Development Kit
JVM	Java Virtual Machine
LED	Light Emitting Diode
LK	Little Kernel
MSM	Mobile Station Modem
OS	Operating System
PID	Process Identification Number
SD	Secure Digital
SDK	Software Developer's Kit
SE	Standard Edition
SURF	Subscriber Unit Reference
USB	Universal Serial Bus
VID	Vendor Identification Number

1.8 Additional information

For additional information, go to https://www.96boards.org/DragonBoard410c/docs.

2 Downloading and Building the Software

2.1 Introduction to 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. 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.

2.2 Downloading HLOS software

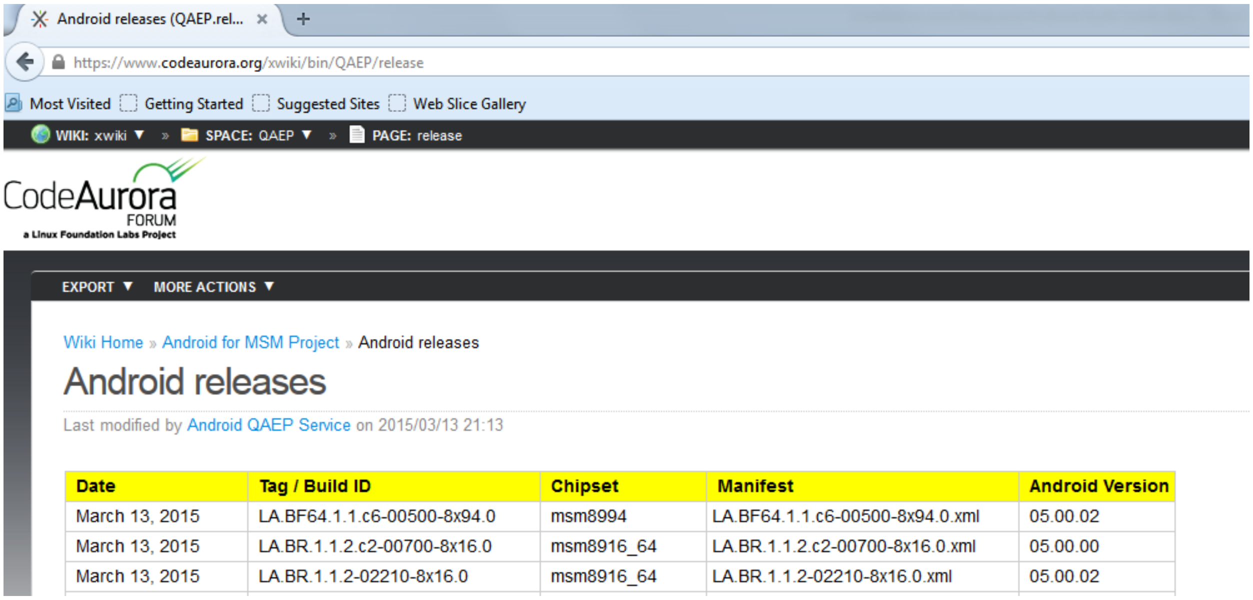
If you want to download Android source code that will work on your Dragonboard 410c, these are the instructions you should use. To download the HLOS software, do the following:

1. First, you need to decide what release you would like to download. If you want to download a board support package, the release should match the board support package. For instance, if you downloaded

   linux_android_board_support_package_vla.br_.**1.2.4-01810-8x16.0-**2.zip

   then you'll look for a manifest called **LA.BR.1.2.4-01810-8x16.0**.xml

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



Wiki Home » Android for MSM Project » Android releases

Android releases

Last modified by Android QAEP Service on 2015/03/13 21:13

Date	Tag / Build ID	Chipset	Manifest	Android Version
March 13, 2015	LA.BF64.1.1.c6-00500-8x94.0	msm8994	LA.BF64.1.1.c6-00500-8x94.0.xml	05.00.02
March 13, 2015	LA.BR.1.1.2.c2-00700-8x16.0	msm8916_64	LA.BR.1.1.2.c2-00700-8x16.0.xml	05.00.00
March 13, 2015	LA.BR.1.1.2-02210-8x16.0	msm8916_64	LA.BR.1.1.2-02210-8x16.0.xml	05.00.02

Figure 2-1 Code Aurora Forum releases page

Each Tag/Build ID as shown in Figure 2-2 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 – 8x16 for MSM8916/APQ8016E

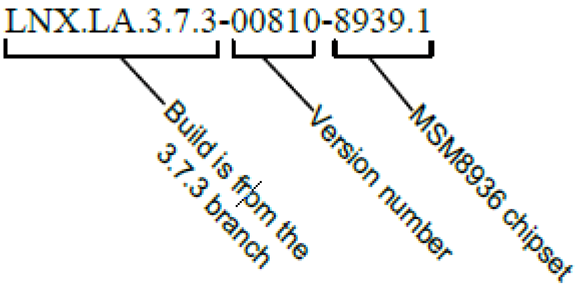


Figure 2-2 Build ID code example

NOTE: This example is for reference only and does not specify the exact Build ID for DragonBoard 410c.

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

3. Once you've found the manifest file you want, you need to actually run the command to download all the source code. In an empty directory, issue the `repo init` command with the correct branch and manifest as indicated in the branch releases table. Please note that a readymade script and build package has been created on

   https://developer.qualcomm.com/hardware/dragonboard-410c/tools

   for different Android releases. The script `DB410c_build.sh` inside the `android_board_support_package_vla.br_XXXXX.zip` file automatically downloads the right source files and uses a proprietary **tar** file in the same directory for the build. One can directly skip to step 5 for clone and build. Steps 1-4 are for informational purposes only.

   ```
   $ repo init -u git://codeaurora.org/platform/manifest.git -b release
   -m [manifest] –repo-url=git://codeaurora.org/tools/repo.git
   ```

4. Type the repo sync command:

   ```
   $ repo sync
   ```

5. Make sure you have at least ~60GB of free space, then download the `android_board_support_package_vla.br_ XXXX.zip` file depending on the version you want from https://developer.qualcomm.com/db410c-tools and copy to the directory where you want to build Android images. This package has the `DB410c_build.sh`, proprietary **tar** file, readme and a license file.

6. After unzipping, run the command `chmod a+x DB410c_build.sh` and run **bash** shell using the **bash** command at the Linux prompt. Then run the build script as `./DB410c_build.sh` or "`sh DB410c_build.sh`" in a **bash** shell.

   a. Accept your name and email ID, when prompted, for the clone to begin.

      The script `DB410c_build.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.

      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 `DB410c_build.sh` for build command and change `make -j4 BUILD_ID=APQ8016_$ITCVER` to:

      `make -j8 BUILD_ID=APQ8016_$ITCVER` or

      `make -j16 BUILD_ID=APQ8016_$ITCVER` depending on your Ubuntu machine configuration.

NOTE: Sometimes the clone from www.codeaurora.org may fail due to network issues; please try again.

2.3 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 msm8916_64-userdebug` (for 64-bit kernel space and 64 bit user space)

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:

   `make -j8` → to build everything

   `make -j8 aboot` → to build LK bootloader emmc_appsboot.mbn

   `make -j8 bootimage` → to build kernel boot.img

   `make -j8 clean` → 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 msm8916_64-userdebug
rm -rf out/target/product/msm8916_64/obj/KERNEL_OBJ/
rm out/target/product/msm8916_64/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

3 Configuring Android adb, fastboot, and Flashing Images to Target Device

3.1 Install Android adb, fastboot, and USB driver for Windows

DragonBoard 410c supports requires the following USB device support:

- Android USB Driver (android_winusb.inf)
- Android adb Interface
- Android Boot Loader Interface (fastboot)

1. Before installing the drivers, edit the **qcmdm.inf** and **qcser.inf** files to make sure that they contain support for the DragonBoard 410c VID/PID (Qualcomm SURF/FFA) with appropriate entries in each section, as indicated in Step 4.
2. Go to http://developer.android.com/sdk/win-usb.html, and follow the instructions to install the SDK and USB driver.
3. Right-click My Computer, and select Properties > Advanced > Environment Variables, and set the path to include the directory c:\android-sdk-windows\tools.
4. The Android USB driver for adb and fastboot needs to add the DragonBoard 410c VID/PID, which supports the connection to the DragonBoard 410c. Edit this file to add the DragonBoard 410c VID/PID lines to each section: **android-sdk-windows\usb_driver\android_winusb.inf**.

```
android_winusb.inf
[Google.NTx86]
;Qualcomm SURF/FFA
%SingleAdbInterface% = USB_Install, USB\VID_05C6&PID_9025
%CompositeAdbInterface% = USB_Install, USB\VID_05C6&PID_9025&MI_01
%SingleBootLoaderInterface% = USB_Install, USB\VID_18D1&PID_D00D

[Google.NTamd64]
;Qualcomm SURF/FFA
%SingleAdbInterface% = USB_Install, USB\VID_05C6&PID_9025
%CompositeAdbInterface% = USB_Install, USB\VID_05C6&PID_9025&MI_01
%SingleBootLoaderInterface% = USB_Install, USB\VID_18D1&PID_D00D
In addition, make sure that there are matching entries under the
[Strings] section.
```

```
[Strings]
SingleAdbInterface = "Android ADB Interface"
CompositeAdbInterface = "Android Composite ADB Interface"
SingleBootLoaderInterface = "Android Bootloader Interface"
```

5. The adb client (adb.exe) supports a built-in list of recognized USB VID/PID devices. To add another device to the list of recognized devices, which is not included in the built-in support list, create this directory if it does not exist: **%USERPROFILE%\.android**.
6. Navigate to the **%USERPROFILE%\.android** directory.
7. In the directory **%USERPROFILE%\.android**, create or edit the file **adb_usb.ini**. If the file exists, it contains a DO NOT EDIT message. Disregard this message and edit the file anyway. Add a line to the end of the file containing 0x05C6.

NOTE: Do *not* run `android update adb` or it resets the contents of this file and overwrites the line just added.

After editing, the adb_usb.ini file must look like the following:

```
# ANDROID 3RD PARTY USB VENDOR ID LIST—DO NOT EDIT.
# USE 'android update adb' TO GENERATE.
# 1 USB VENDOR ID PER LINE.
0x05C6
```

3.2 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
   ```
   The result must be similar to the following output:
   ```
   #Sooner low-level bootloader
   SUBSYSTEM=="usb", SYSFS{idVendor}=="18d1", SYSFS{idProduct}=="d00d",
   MODE="0664", GROUP="plugdev"

   # adb composite interface device 9025
   SUBSYSTEM=="usb", SYSFS{idVendor}=="05C6", SYSFS{idProduct}=="9025",
   MODE="0664", GROUP="plugdev"
   ```
3. After editing the file, to see the list of target devices connected to the Linux box, type:
   ```
   lsusb
   ```
4. 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).
   a. Type the command:
      ```
      source build/envsetup.sh
      ```

b. Type the command:

```
lunch msm8916_64-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/

3.3 Program Android images using fastboot

1. Hold down the VOL- key, connect the DC supply to the DragonBoard 410c, and press the power ON button. Plug the USB cable into the target. This will bring the device into fastboot mode.
2. Depending on your build environment, choose *one* of the following options

   a. From the Windows command shell, run:
   ```
   fastboot devices
   ```

   b. From Linux:

      i. Navigate to the following directory:

      cd <$BUILDROOT>/device/out/host/linux-x86/bin

      ii. Run: `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. During power up you should see USER LED #4 glowing green, indicating bootup sign of life.

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.

4 More on adb Usage and Android Build

4.1 Install applications on Android target using adb

1. Plug the USB cable into the target.
2. Navigate to the following directory:
   ```
   cd <root>/LINUX/device/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>`

4.2 Android device 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., msm8916

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

4.3 Android source 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

4.4 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 msm8916_64-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/msm8916_64/obj/KERNEL_OBJ/

4.5 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
      ```

4.6 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 of 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.

EXHIBIT 1

PLEASE READ THIS LICENSE AGREEMENT (“AGREEMENT”) CAREFULLY. THIS AGREEMENT IS A BINDING LEGAL AGREEMENT ENTERED INTO BY AND BETWEEN YOU (OR IF YOU ARE ENTERING INTO THIS AGREEMENT ON BEHALF OF AN ENTITY, THEN THE ENTITY THAT YOU REPRESENT) AND QUALCOMM TECHNOLOGIES, INC. (“QTI” “WE” “OUR” OR “US”). THIS IS THE AGREEMENT THAT APPLIES TO YOUR USE OF THE DESIGNATED AND/OR ATTACHED DOCUMENTATION AND ANY UPDATES OR IMPROVEMENTS THEREOF (COLLECTIVELY, “MATERIALS”). BY USING OR COMPLETING THE INSTALLATION OF THE MATERIALS, YOU ARE ACCEPTING THIS AGREEMENT AND YOU AGREE TO BE BOUND BY ITS TERMS AND CONDITIONS. IF YOU DO NOT AGREE TO THESE TERMS, QTI IS UNWILLING TO AND DOES NOT LICENSE THE MATERIALS TO YOU. IF YOU DO NOT AGREE TO THESE TERMS YOU MUST DISCONTINUE AND YOU MAY NOT USE THE MATERIALS OR RETAIN ANY COPIES OF THE MATERIALS. ANY USE OR POSSESSION OF THE MATERIALS BY YOU IS SUBJECT TO THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT.

1.1 **License.** Subject to the terms and conditions of this Agreement, including, without limitation, the restrictions, conditions, limitations and exclusions set forth in this Agreement, Qualcomm Technologies, Inc. (“QTI”) hereby grants to you a nonexclusive, limited license under QTI’s copyrights to use the attached Materials; and to reproduce and redistribute a reasonable number of copies of the Materials. You may not use Qualcomm Technologies or its affiliates or subsidiaries name, logo or trademarks; and copyright, trademark, patent and any other notices that appear on the Materials may not be removed or obscured. QTI shall be free to use suggestions, feedback or other information received from You, without obligation of any kind to You. QTI may immediately terminate this Agreement upon your breach. Upon termination of this Agreement, Sections 1.2-4 shall survive.

1.2 **Indemnification.** You agree to indemnify and hold harmless QTI and its officers, directors, employees and successors and assigns against any and all third party claims, demands, causes of action, losses, liabilities, damages, costs and expenses, incurred by QTI (including but not limited to costs of defense, investigation and reasonable attorney’s fees) arising out of, resulting from or related to: (i) any breach of this Agreement by You; and (ii) your acts, omissions, products and services. If requested by QTI, You agree to defend QTI in connection with any third party claims, demands, or causes of action resulting from, arising out of or in connection with any of the foregoing.

1.3 **Ownership.** QTI (or its licensors) shall retain title and all ownership rights in and to the Materials and all copies thereof, and nothing herein shall be deemed to grant any right to You under any of QTI's or its affiliates’ patents. You shall not subject the Materials to any third party license terms (e.g., open source license terms). You shall not use the Materials for the purpose of identifying or providing evidence to support any potential patent infringement claim against QTI, its affiliates, or any of QTI’s or QTI’s affiliates’ suppliers and/or direct or indirect customers. QTI hereby reserves all rights not expressly granted herein.

1.4 **WARRANTY DISCLAIMER.** YOU EXPRESSLY ACKNOWLEDGE AND AGREE THAT THE USE OF THE MATERIALS IS AT YOUR SOLE RISK. THE MATERIALS AND TECHNICAL SUPPORT, IF ANY, ARE PROVIDED "AS IS" AND WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED. QTI ITS LICENSORS AND AFFILIATES MAKE NO WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THE MATERIALS OR ANY OTHER INFORMATION OR DOCUMENTATION PROVIDED UNDER THIS AGREEMENT, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR AGAINST INFRINGEMENT, OR ANY EXPRESS OR IMPLIED WARRANTY ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. NOTHING CONTAINED IN THIS AGREEMENT SHALL BE CONSTRUED AS (I) A WARRANTY OR REPRESENTATION BY QTI, ITS LICENSORS OR AFFILIATES AS TO THE VALIDITY OR SCOPE OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT OR (II) A WARRANTY OR REPRESENTATION BY QTI THAT ANY MANUFACTURE OR USE WILL BE FREE FROM INFRINGEMENT OF PATENTS, COPYRIGHTS OR OTHER INTELLECTUAL PROPERTY RIGHTS OF OTHERS, AND IT SHALL BE THE SOLE RESPONSIBILITY OF YOU TO MAKE SUCH DETERMINATION AS IS NECESSARY WITH RESPECT TO THE ACQUISITION OF LICENSES UNDER PATENTS AND OTHER INTELLECTUAL PROPERTY OF THIRD PARTIES.

1.5 **LIMITATION OF LIABILITY.** IN NO EVENT SHALL QTI, QTI’S AFFILIATES OR ITS LICENSORS BE LIABLE TO YOU FOR ANY INCIDENTAL, CONSEQUENTIAL OR SPECIAL DAMAGES, INCLUDING BUT NOT LIMITED TO ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL DAMAGES, ARISING OUT OF THE USE OR INABILITY TO USE, OR THE DELIVERY OR FAILURE TO DELIVER, ANY OF THE MATERIALS, OR ANY BREACH OF ANY OBLIGATION UNDER THIS AGREEMENT, EVEN IF QTI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE FOREGOING LIMITATION OF LIABILITY SHALL REMAIN IN FULL FORCE AND EFFECT REGARDLESS OF WHETHER YOUR REMEDIES HEREUNDER ARE DETERMINED TO HAVE FAILED OF THEIR ESSENTIAL PURPOSE. THE ENTIRE LIABILITY OF QTI, QTI’s AFFILIATES AND ITS LICENSORS, AND THE SOLE AND EXCLUSIVE REMEDY OF YOU, FOR ANY CLAIM OR CAUSE OF ACTION ARISING HEREUNDER (WHETHER IN CONTRACT, TORT, OR OTHERWISE) SHALL NOT EXCEED US$10.

2. **COMPLIANCE WITH LAWS; APPLICABLE LAW.** You agree to comply with all applicable local, international and national laws and regulations and with U.S. Export Administration Regulations, as they apply to the subject matter of this Agreement. This Agreement is governed by the laws of the State of California, excluding California’s choice of law rules.

3. **CONTRACTING PARTIES.** If the Materials are downloaded on any computer owned by a corporation or other legal entity, then this Agreement is formed by and between QTI and such entity. The individual accepting the terms of this Agreement represents and warrants to QTI that they have the authority to bind such entity to the terms and conditions of this Agreement.

4. **MISCELLANEOUS PROVISIONS.** This Agreement, together with all exhibits attached hereto, which are incorporated herein by this reference, constitutes the entire agreement between QTI and You and supersedes all prior negotiations, representations and agreements between the parties with respect to the subject matter hereof. No addition or modification of this Agreement shall be effective unless made in writing and signed by the respective representatives of QTI and You. The restrictions, limitations, exclusions and conditions set forth in this Agreement shall apply even if QTI or any of its affiliates becomes aware of or fails to act in a manner to address any violation or failure to comply therewith. You hereby acknowledge and agree that the restrictions, limitations, conditions and exclusions imposed in this Agreement on the rights granted in this Agreement are not a derogation of the benefits of such rights. You further acknowledges that, in the absence of such restrictions, limitations, conditions and exclusions, QTI would not have entered into this Agreement with You. Each party shall be responsible for and shall bear its own expenses in connection with this Agreement. If any of the provisions of this Agreement are determined to be invalid, illegal, or otherwise unenforceable, the remaining provisions shall remain in full force and effect. This Agreement is entered into solely in the English language, and if for any reason any other language version is prepared by any party, it shall be solely for convenience and the English version shall govern and control all aspects. If You are located in the province of Quebec, Canada, the following applies: The Parties hereby confirm they have requested this Agreement and all related documents be prepared in English.