NXP Semiconductors
User's Guide

Document Number: AUG
Rev. O8.1.0_1.3.1_8MQ-ear, 07/2018

Android™ User's Guide

Contents

1 Overview

This document describes how to build Android Oreo 8.1 platform for the i.MX 8 series devices. It provides instructions for:

- Configuring a Linux® OS build machine.
- Downloading, patching, and building the software components that create the Android™ system image.
- Building from sources and using pre-built images.
- Copying the images to boot media.
- Hardware/software configurations for programming the boot media and running the images.

For more information about building the Android platform, see source.android.com/source/building.html.

2 Preparation

The minimum recommended system requirements are as follows:

- 16 GB RAM
- 300 GB hard disk

For any problems on the building process related to the Jack server, see the Android website source.android.com/source/jack.html.

2.1 Setting up your computer

To build the Android source files, use a computer running the Linux OS. The Ubuntu 16.04 64bit version and openjdk-8-jdk is the most tested environment for the Android Oreo 8.1 build.

After installing the computer running Linux OS, check whether all the necessary packages are installed for an Android build. See "Setting up your machine" on the Android website source.android.com/source/initializing.html.

In addition to the packages requested on the Android website, the following packages are also needed:

```
$ sudo apt-get install uuid uuid-dev
$ sudo apt-get install zlib1g-dev liblz-dev
$ sudo apt-get install liblzo2-2 liblzo2-dev
$ sudo apt-get install lzop
$ sudo apt-get install git-core curl
$ sudo apt-get install u-boot-tools
$ sudo apt-get install mtd-utils
$ sudo apt-get install android-tools-fsutils
$ sudo apt-get install openjdk-8-jdk
$ sudo apt-get install device-tree-compiler
$ sudo apt-get install gdisk
```

NOTE

If you have trouble installing the JDK in Ubuntu, see How to install misc JDK in Ubuntu for Android build.

Configure git before use. Set the name and email as follows:

- git config --global user.name "First Last"
- git config --global user.email "first.last@company.com"

2.2 Unpacking the Android release package

After you have set up a computer running Linux OS, unpack the Android release package by using the following commands:

```
$ cd ~ (or any other directory you like)
$ tar xzvf imx-o8.1.0_1.3.1_8mq-ear.tar.gz
```

3 Building the Android platform for i.MX

3.1 Getting i.MX Android release source code

The i.MX Android release source code consists of three parts:

- NXP i.MX public source code, which is maintained in the CodeAurora Forum repository.
- AOSP Android public source code, which is maintained in android.googlesource.com.
- NXP i.MX Android proprietary source code package, which is maintained in www.NXP.com

Assume you had i.MX Android proprietary source code package imx-o8.1.0_1.3.1_8mq-ear.tar.gz under ~/. directory. To generate the i.MX Android release source code build environment, execute the following commands:

```
$ mkdir ~/bin
$ curl https://storage.googleapis.com/git-repo-downloads/repo  > ~/bin/repo
$ chmod a+x ~/bin/repo
```

```
$ export PATH=${PATH}:~/bin
$ source ~/imx-o8.1.0_1.3.1_8mq-ear/imx_android_setup.sh
# By default, the imx_android_setup.sh script will create the source code build environemnt
in the folder ~/android_build
# ${MY_ANDROID} will be refered as the i.MX Android source code root directory in all i.MX
Andorid release documentation.
$ export MY_ANDROID=~/android_build
```

3.2 Building Android images

Building the Android image is performed when the source code has been downloaded (Section 3.1) and patched (Section 3.2).

Commands **lunch <buildName-buildType>** to set up the build configuration and **make** to start the build process are executed.

The build configuration command **lunch** can be issued with an argument <Build name>-<Build type> string, such as **lunch evk_8mq-userdebug**, or can be issued without the argument, which will present a menu of options to select.

The Build Name is the Android device name found in the directory ${MY_ANDROID}/device/fsl/. The following table lists the i.MX build names.

Table 1. Build names

Build name	Description
evk_8mq	i.MX 8MQuad EVK Board

The build type is used to specify what debug options are provided in the final image. The following table lists the build types.

Table 2. Build types

Build type	Description
user	Production-ready image, no debug
userdebug	Provides image with root access and debug, similar to "user"
eng	Development image with debug tools

Android build steps are as follows:

1. Change to the top level build directory.

   ```
   $ cd ${MY_ANDROID}
   ```
2. Set up the environment for building. This only configures the current terminal.

   ```
   $ source build/envsetup.sh
   ```
3. Execute the Android **lunch** command. In this example, the setup is for the production image of i.MX 8MQuad EVK Board/Platform device with userdebug type.

   ```
   $ lunch evk_8mq-userdebug
   ```
4. Execute the **make** command to generate the image.

   ```
   $ make 2>&1 | tee build-log.txt
   ```

When the **make** command is complete, the build-log.txt file contains the execution output. Check for any errors.

For BUILD_ID & BUILD_NUMBER changing, update build_id.mk in your ${MY_ANDROID} directory. For details, see the *Android™ Frequently Asked Questions* (AFAQ).

The following outputs are generated by default in ${MY_ANDROID}/out/target/product/evk_8mq:

- root/: root file system (including init, init.rc). Mounted at /.
- system/: Android system binary/libraries. Mounted at /system.
- data/: Android data area. Mounted at /data.
- recovery/: root file system when booting in "recovery" mode. Not used directly.
- boot-imx8mq.img: a composite image for i.MX 8MQuad EVK, which includes the kernel zImage, ramdisk, board's device tree binary, and boot parameters. It is used to support HDMI output.
- boot-imx8mq-dsd.img: a composite image for i.MX 8MQuad EVK, which includes the kernel zImage, ramdisk, board's device tree binary, and boot parameters. It is used to support HDMI output and Direct Stream Digital (DSD) playback.
- boot-imx8mq-mipi.img: a composite image for i.MX 8MQuad EVK, which includes the kernel zImage, ramdisk, board's device tree binary, and boot parameters. It is used to support MIPI-to-HDMI output.
- boot-imx8mq-dual.img: a composite image for i.MX 8MQuad EVK, which includes the kernel zImage, ramdisk, board's device tree binary, and boot parameters. It is used to support HDMI and MIPI-to-HDMI dual output.
- boot-imx8mq-mipi-panel.img: a composite image for i.MX 8MQuad EVK, which includes the kernel zImage, ramdisk, board's device tree binary, and boot parameters. It is used to support MIPI Panel output.
- ramdisk.img: Ramdisk image generated from "root/". Not directly used.
- system.img: EXT4 image generated from "system/". Can be programmed to "SYSTEM" partition on SD/eMMC card with "dd".
- userdata.img: EXT4 image generated from "data/".
- vbmeta-imx8mq.img: Android Verify boot metadata image for boot-imx8mq.img.
- vbmeta-imx8mq-dsd.img: Android Verify boot metadata image for boot-imx8mq-dsd.img.
- vbmeta-imx8mq-mipi.img: Android Verify boot metadata image for boot-imx8mq-mipi.img.
- vbmeta-imx8mq-dual.img: Android Verify boot metadata image for boot-imx8mq-dual.img.
- vbmeta-imx8mq-mipi-panel.img: Android Verify boot metadata image for boot-imx8mq-mipi-panel.img.
- partition-table.img: GPT partition table image. Used for 16 GB SD card and eMMC card.
- partition-table-7GB.img: GPT partition table image. Used for 8 GB SD card.
- partition-table-28GB.img: GPT partition table image. Used for 32 GB SD card.
- u-boot-imx8mq.imx: U-Boot image without padding for i.MX 8MQuad EVK.
- vendor.img: vendor image, which holds platform binaries. Mounted at **/vendor**

NOTE

- To build the U-Boot image separately, see Building U-Boot images.
- To build the kernel uImage separately, see Building a kernel image.
- To build boot.img, see Building boot.img.

3.2.1 Configuration examples of building i.MX devices

The following table shows examples of using the `lunch` command to set up different i.MX devices. After the desired i.MX device is set up, the `make` command is used to start the build.

Table 3. i.MX device lunch examples

Build name	Description
i.MX 8MQuad EVK Board	$ lunch evk_8mq-userdebug

After the `lunch` command is executed, the **make** command is issued:

```
$ make 2>&1 | tee build-log.txt
```

3.2.2 User build mode

A production release Android system image is created by using the **userdebug** Build Type. For configuration options, see Table "Build types" in Section Building Android images.

The notable differences between the **user** and **eng** build types are as follows:

- Limited Android System image access for security reasons.
- Lack of debugging tools.
- Installation modules tagged with user.
- APKs and tools according to product definition files, which are found in PRODUCT_PACKAGES in the sources folder ${MY_ANDROID}/device/fsl/imx8/imx8.mk. To add customized packages, add the package MODULE_NAME or PACKAGE_NAME to this list.
- The properties are set as: `ro.secure=1` and `ro.debuggable=0`.
- `adb` is disabled by default.

There are two methods for the build of Android image.

Method 1: Set the environment first and then issue the `make` command:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh   #set env
$ make PRODUCT-XXX        #XXX depends on different board, see table below
```

Table 4. Android system image production build method 1

i.MX development tool	Description	Image build command
Evaluation Kit	i.MX 8MQuad EVK	$ make PRODUCT-evk_8mq-userdebug>&1 | tee buildlog.txt

Method 2: Set the environment and then use `lunch` command to configure argument. See table below. An example for the i.MX 8MQuad EVK board is as follows:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_8mq-userdebug
$ make
```

Table 5. Android system image production build method 2

i.MX development tool	Description	Lunch configuration
Evaluation Kit	i.MX 8MQuad EVK	evk_8mq-userdebug

To create Android platform over-the-air, OTA, and package, the following make target is specified:

```
$ make otapackage
```

For more Android platform building information, see source.android.com/source/building.html.

3.3 Building U-Boot images

U-Boot image for i.MX 8MQuad EVK board:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_8mq-userdebug
$ make bootloader
```

3.4 Building a kernel image

Kernel image is automatically built when building the Android root file system.

The following are the default Android build commands to build the kernel image:

```
$ cd ${MY_ANDROID}/vendor/nxp-opensource/kernel-imx
$ echo $ARCH && echo $CROSS_COMPILE
```

Make sure that you have those two environment variables set. If the two variables are not set, set them as follows:

```
$ export ARCH=arm64
$ export CROSS_COMPILE=${MY_ANDROID}/prebuilts/gcc/linux-x86/aarch64/aarch64-linux-
android-4.9/bin/aarch64-linux-android-

# Generate ".config" according to default config file under arch/arm64/configs/
android_defconfig.
# To build the kernel zImage for i.MX 8MQuad
$ make android_defconfig
$ make KCFLAGS=-mno-android

# to build the zImage which is used in MFGTOOL
# zImage is under mfgtools\Profiles\Linux\OS Firmware\firmware\
$ make defconfig
$ make KCFLAGS=-mno-android -j4
```

The kernel images are found in ${MY_ANDROID}/out/target/product/evk_8mq/obj/KERNEL_OBJ/arch/arm64/boot/Image.

3.5 Building boot.img

boot.img and boota are default booting commands.

As outlined in Running the Android Platform with a Prebuilt Image, we use boot.img and boota as default commands to boot instead of the uramdisk and zImage we used before.

Use this command to generate boot.img under Android environment:

```
# Boot image for i.MX 8MQuad EVK board
$ source build/envsetup.sh
$ lunch evk_8mq-userdebug
$ make bootimage
```

4 Running the Android Platform with a Prebuilt Image

To test the Android platform before building any code, use the prebuilt images from the following packages and go to "Download Images" and "Boot".

Table 6. Image packages

Image package	Description
android_o8.1.0_1.3.1_8mq-ear_image_8mqevk.tar.gz	Prebuilt-image for i.MX 8MQuad EVK board, which includes NXP extended features.

The following tables list the detailed contents of android_o8.1.0_1.3.1_8mq-ear_image_8mqevk.tar.gz image package.

The table below shows the prebuilt images to support the system boot from SD and eMMC on i.MX 8MQuad EVK boards.

Table 7. Images for i.MX 8MQuad EVK

i.MX 8MQuad EVK image	Description
u-boot-imx8mq.imx	Bootloader (with padding) for i.MX 8MQuad EVK board.
partition-table.img	GPT table image for 16 GB SD card and eMMC.
partition-table-7GB.img	GPT table image for 8 GB SD card.
partition-table-28GB.img	GPT table image for 32 GB SD card.
boot-imx8mq.img	Boot image for i.MX 8MQuad EVK board to support HDMI output.
boot-imx8mq-dsd.img	Boot image for i.MX 8MQuad EVK board to support HDMI output and DSD playback.
boot-imx8mq-mipi.img	Boot image for i.MX 8MQuad EVK board to support MIPI-to-HDMI output.
boot-imx8mq-dual.img	Boot image for i.MX 8MQuad EVK board to support HDMI and MIPI-to-HDMI dual output.
boot-imx8mq-mipi-panel.img	Boot image for i.MX 8MQuad EVK board to support MIPI panel output.
system.img	System Boot image.
vbmeta-imx8mq.img	Android Verify Boot metadata Image for i.MX 8MQuad EVK board to support HDMI output.
vbmeta-imx8mq-dsd.img	Android Verify Boot metadata image for i.MX 8MQuad EVK board to support HDMI output and DSD playback.
vbmeta-imx8mq-mipi.img	Android Verify Boot metadata image for i.MX 8MQuad EVK board to support MIPI-to-HDMI output.
vbmeta-imx8mq-dual.img	Android Verify Boot metadata image for i.MX 8MQuad EVK board to support HDMI and MIPI-to-HDMI dual output.
vbmeta-imx8mq-mipi-panel.img	Android Verify Boot metadata image for i.MX 8MQuad EVK board to support MIPI panel output.
vendor.img	Vendor image for i.MX 8MQuad EVK board

NOTE

boot.img is an Android image that stores kernel Image and ramdisk together. It also stores other information such as the kernel boot command line, machine name. This information can be configured in android.mk. It can avoid touching the boot loader code to change any default boot arguments.

5 Programming Images

The images from the prebuilt release package or created from source code contain the U-Boot boot loader, system image, GPT image, vendor image, and vbmeta image. At a minium, the storage devices on the development system (MMC/SD or NAND) must be programmed with the U-Boot boot loader. The i.MX 8 series boot process determines what storage device to access based on the switch settings. When the boot loader is loaded and begins execution, the U-Boot environment space is then read to determine how to proceed with the boot process. For U-Boot environment settings, see Section Booting.

The following download methods can be used to write the Android System Image:

- MFGTool to download all images to eMMC/SD card.

5.1 System on eMMC/SD

The images needed to create an Android system on eMMC/SD can either be obtained from the release package or be built from source.

The images needed to create an Android system on eMMC/SD are listed below:

- U-Boot image: u-boot.imx
- Android boot image: boot.img
- Android system image: system.img
- Android verify boot metadata image: vbmeta.img
- GPT table image: partition-table.img
- Android vendor image: vendor.img

5.1.1 Storage partitions

The layout of the eMMC/SD/TF card for Android system is shown below:

- [Partition type/index] which is defined in the GPT.
- [Start Offset] shows where partition is started, unit in MB.

The system partition is used to put the built-out Android system image. The userdata parition is used to put the unpacked codes/data of the applications, system configuration database, etc. In normal boot mode, the root file system is mounted from the system partition. In recovery mode, the root file system is mounted from the boot partition.

Table 8. Storage partitions

Partition type/index	Name	Start offset	Size	File system	Content
N/A	bootloader	33 KB	8 MB - 33 KB	N/A	bootloader
1	boot_a	8 MB	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
2	boot_b	Follow boot_a	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
3	system_a	Follow boot_b	1536 MB	EXT4. Mount as / system	Android system files under / system/dir
4	system_b	Follow system_a	1536MB	EXT4. Mount as / system	Android system files under / system/dir
5	misc	Follow system_b	4 MB	N/A	For recovery store bootloader message, reserve
6	datafooter	Follow misc	2 MB	N/A	For crypto footer of DATA partition encryption
7	metadata	Follow datafootor	2 MB	N/A	For system slide show
8	presistdata	Follow metadata	1 MB	N/A	Option to operate unlock \unlock
9	vendor_a	Follow persistdata	112 MB	EXT4. Mount at / vendor	vendor.img

Table continues on the next page...

Table 8. Storage partitions (continued)

Partition type/index	Name	Start offset	Size	File system	Content
10	vendor_b	Follow vendor_a	112 MB	EXT4. Mount at / vendor	vendor.img
11	userdata	Follow vendor_b	Remained space	EXT4. Mount at /data	Application data storage for system application, and for internal media partition, in /mnt/sdcard/ dir.
12	fbmisc	Follow userdata	1 MB	N/A	For storing the state of lock \unlock
13	vbmeta_a	Follow fbmisc	1 MB	N/A	For storing the verify boot's metadata
14	vbmeta_b	Follow vbmeta_a	1 MB	N/A	For storing the verify boot's metadata

To create these partitions, use fsl-sdcard-partition.sh described in the *Android™ Quick Start Guide* (AQSUG), or use format tools in the prebuilt directory.

The script below can be used to partition an SD card as shown in the partition table above:

```
$ cd ${MY_ANDROID}/
$ sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> /dev/sdX
# <soc_name> can be as imx8mq.
```

NOTE

- The minimum size of the SD card is 8 GB.
- /dev/sdX, the X is the disk index from 'a' to 'z'. That may be different on each computer running Linux OS.
- If the SD card is 8 GB, use "sudo fsl-sdcard-partition.sh -f imx8mq -c 7 /dev/sdX" to flash images.
- If the SD card is 16 GB, use "sudo fsl-sdcard-partition.sh -f imx8mq /dev/sdX" to flash images.
- If the SD card is 32 GB, use "sudo fsl-sdcard-partition.sh -f imx8mq -c 28 /dev/sdX" to flash images.
- Unmount all the SD card partitions before running the script.
- Put the related bootloader, boot image, system image, recovery image in your current directory. This script requires to install the simg2img tool on the computer. simg2img is a tool that converts the sparse system image to raw system image on the Linux OS host computer. The android-tools-fsutils package includes the simg2img command for Ubuntu Linux OS.
- If the SD card is 8 GB, copy partition-table-7GB.img and rename it to partition-table.img. If the SD card is 16 GB, use the default partition-table.img. If the SD card is 32 GB, copy partition-table-28GB.img and rename it to partition-table.img.

5.1.2 Downloading images with MFGTool

MFGTool can be used to download all images into a target device. It is a quick and easy tool for downloading images. See *Android™ Quick Start Guide* (AQSUG) for a detailed description of MFGTool.

6 Booting

This chapter describes booting from MMC/SD.

6.1 Booting from eMMC/SD

6.1.1 Booting from SD/eMMC on the i.MX 8MQuad EVK board

The following tables list the boot switch settings to control the boot storage.

Table 9. Boot device switch settings

Boot device switch	External SDcard	eMMC
SW801 (from 1 bit to 4 bit)	1100	0010

Table 10. Boot mode switch settings

Boot mode switch	Download Mode (MFGTool mode)	Boot mode
SW802 (from 1 bit to 2 bit)	01	10

To test booting from SD, change the board Boot_Mode switch to 10 (from 1 bit to 2 bit) and SW801 1100 (from 1 bit to 4 bit).

To test booting from eMMC, change the board Boot_Mode switch to 10 (from 1 bit to 2 bit) and SW801 0010 (from 1 bit to 4 bit).

The default environment in boot.img is booting from eMMC. To use the default environment in boot.img, use the following command:

```
U-Boot > setenv bootargs
```

To clear the bootargs environment, use the following command:

```
U-Boot >  setenv bootargs console=ttymxc0,115200 earlycon=imxuart,0x30860000,115200 init=/
init androidboot.gui_resolution=1080p androidboot.console=ttymxc0 consoleblank=0
androidboot.hardware=freescale androidboot.fbTileSupport=enable cma=1280M
androidboot.primary_display=imx-drm firmware_class.path=/vendor/firmware   [Optional]
U-Boot > saveenv                          [Save the environments]
```

NOTE

bootargs environment is an optional setting for boota. The boot.img includes a default bootargs, which is used if there is no definition about the bootargs environment. This bootargs is default for HDMI output. To test other outputs, see *Android™ Quick Start Guide* (AQSUG).

6.2 Boot-up configurations

This section explains some common boot-up configurations such as U-Boot environments, kernel command lines, and DM-verity configuartions.

6.2.1 U-Boot environment

- bootcmd: the first variable to run after U-Boot boot.
- bootargs: the kernel command line, which the bootloader passes to the kernel. As described in Kernel command line (bootargs), bootargs environment is optional for booti. boot.img already has bootargs. If you do not define the bootargs environment, it uses the default bootargs inside the image. If you have the environment, it is then used.

  To use the default environment in boot.img, use the following command to clear the bootargs environment.

  ```
  > setenv bootargs
  ```

  If the environment variable "append_bootargs" is set, the value of "append_bootargs" is appended to "bootargs" automatically.

- dhcp: get the IP address by BOOTP protocol, and load the kernel image ($bootfile env) from the TFTP server.
- boota:

  boota command parses the boot.img header to get the zImage and ramdisk. It also passes the bootargs as needed (it only passes bootargs in boot.img when it cannot find "bootargs" var in your U-Boot environment). To boot from mmcX, do the following:

  ```
  > boota mmcX
  ```

  To read the boot partition (the partition store boot.img, in this instance, mmcblk0p1), the X is the eMMC bus number, which is the hardware eMMC bus number, in SABRE-SD boards. eMMC is mmc2 or you can add the partition ID after mmcX.

  Add partition ID after mmcX.

  ```
  > boota mmcX boot   #  boot is default
  > boota mmcX recovery      # boot from the recovery partition
  ```

  If you have read the boot.img into memory, use this command to boot from

  ```
  > boota 0xXXXXXXXX
  ```

6.2.2 Kernel command line (bootargs)

Depending on the different booting/usage scenarios, you may need different kernel boot parameters set for bootargs.

Table 11. Kernel boot parameters

Kernel parameter	Description	Typical value	Used when
console	Where to output kernel log by printk.	console=ttymxc0,115200	All use cases.
init	Tells kernel where the init file is located.	init=/init	All use cases. "init" in the Android platform is located in "/" instead of in "/sbin".

Table continues on the next page...

Table 11. Kernel boot parameters (continued)

Kernel parameter	Description	Typical value	Used when
androidboot.hardware	Specifies hardware name for this product. Android init process loads the configuration file init.$ (androidboot.hardware).rc in the root directory.	androidboot.hardware=freescale	All use cases.
androidboot.console	The Android shell console. It should be the same as console=.	androidboot.console=ttymxc0	To use the default shell job control, such as Ctrl+C to terminate a running process, set this for the kernel.
fec_mac	Sets up the FEC MAC address.	fec_mac=00:04:9f:00:ea:d3	On SABRE-SD board, the SoC does not have a MAC address fused in. To use FEC, assign this parameter to the kernel.
cma	CMA memory size for GPU/VPU physical memory allocation.	cma=1280M	It is 1280 MB by default.
androidboot.selinux	Argument to disable selinux check and enable serial input when connecting a host computer to the target board's USB UART port. For details about selinux, see Security-Enhanced Linux in Android.	androidboot.selinux=permissive	Android Nougat 7.1 CTS requirement: The serial input should be disabled by default. Setting this argument enables console serial input, which violates the CTS requirement.
androidboot.lcd_density	It is used to set the display density and over-write ro.sf.lcd_density in init.freescale.rc for MIPI-to-HDMI display.	androidboot.lcd_density=160	-
androidboot.primary_display	It is used to choose and fix the primary display.	androidboot.primary_display=imx-drm	androidboot.primary_display=mxsfb-drm is only used for the MIPI-to-HDMI display.
androidboot.fbTileSupport	It is used to enable framebuffer super tile output.	androidboot.fbTileSupport=enable	It should not be set when connecting the MIPI-to-HDMI display or MIPI panel display.
firmware_class.path	It is used to set the Wi-Fi firmware path.	firmware_class.path=/vendor/firmware	-

6.2.3 DM-verity configuration

DM-verity (device-mapper-verity) provides transparent integrity checking of block devices. It can prevent device from running unauthorized images. This feature is enabled by default. Replacing one or more partitions (boot, vendor, system, vbmeta) will make the board unbootable. Disabling DM-verity provides convience for developers, but the device is unprotected.

To disable DM-verity, perform the following steps:

1. Unlock the device.
    a. Boot up the device.
    b. Choose **Settings** -> **Developer Options** -> **OEM Unlocking** to enable OEM unlocking.
    c. Enter Fastboot mode on the device. Execute the following command on the target side:

    ```
    reboot bootloader
    ```

    d. Unlock the device. Execute the following command on the host side:

    ```
    fastboot oem unlock
    ```

    e. Wait until the unlock process is complete.
2. Disable DM-verity.
    a. Boot up the device.
    b. Disable the DM-verity feature. Execute the following command on the host side:

    ```
    adb root
    adb disable-verity
    adb reboot
    ```

7 OTA (Over-The-Air) Update

7.1 Building OTA update packages

7.1.1 Building target files

You can use the following commands to generate target files under the Android environment:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_8mq-userdebug
$ make target-files-package -j4
```

After building is complete, you can find the target files in the following path:

```
${MY_ANDROID}/out/target/product/evk_8mq/obj/PACKAGING/target_files_intermediates/evk_8mq-
target_files-${date}-${soc}.zip
```

7.1.2 Building a full update package

A full update is one where the entire final state of the device (system, boot, and vendor partitions) is contained in the package.

You can use the following commands to build a full update package under the Android environment:

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_8mq-userdebug
$ make otapackage -j4
```

After building is complete, you can find the OTA packages in the following path:

```
${MY_ANDROID}/out/target/proudct/evk_8mq/evk_8mq-ota-${date}-${soc}.zip
```

`evk_8mq-ota-${date}-${soc}.zip` includes `payload.bin` and `payload_properties.txt`. The two files are used for full update.

NOTE

- `${date}` is the BUILD_NUMBER in `build_id.mk`.
- `${soc}` is the SoC name, such as imx8mq, imx8mq-dsd, imx8mq-dual, imx8mq-mipi, and imx8mq-mipi-panel.

7.1.3 Building an incremental update package

An incremental update contains a set of binary patches to be applied to the data that is already on the device. This can result in considerably smaller update packages:

- Files that have not changed do not need to be included.
- Files that have changed are often very similar to their previous versions, so the package only needs to contain encoding of the differences between the two files. You can install the incremental update package only on a device that has the old or source build used when constructing the package.

Before building an incremental update package, see Section 7.1.1 to build two target files:

- PREVIOUS-target_files.zip: one old package that has already been applied on the device.
- NEW-target_files.zip: the latest package that is waiting to be applied on the device.

Then use the following commands to generate the incremental update package under the Android environment:

```
$ cd ${MY_ANDROID}
$ ./build/tools/releasetools/ota_from_target_files -i PREVIOUS-target_files.zip NEW-
target_files.zip incremental_ota_update.zip
```

`${MY_ANDROID}/incremental_ota_update.zip` includes `payload.bin` and `payload_properties.txt`. The two files are used for incremental update.

7.2 Implementing OTA update

7.2.1 Useing update_engine_client to update the Android platform

update_engine_client is a pre-built tool to support A/B (seamless) system updates.

- Copy `ota_update.zip` or `incremental_ota_update.zip` (generated on 7.1.2 and 7.1.3) to the HTTP server (for example, 192.168.1.1:/var/www/).
- Unzip the packages to get `payload.bin` and `payload_properties.txt`.
- Cat the content of `payload_properties.txt` like this:
  - FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
  - FILE_SIZE=379074366
  - METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ=
  - METADATA_SIZE=46866

- Input the following command on the board's console to update:

```
update_engine_client --payload=http://192.168.1.1:10888/payload.bin --update --
headers="FILE_HASH=0fSBbXonyTjaAzMpwTBgM9AVtlBeyOigpCCgkoOfHKY=
FILE_SIZE=379074366
METADATA_HASH=Icrs3NqoglzyppyCZouWKbo5f08IPokhlUfHDmz77WQ/de8Dgp9zFXt8Fo
+Hxccp465uTOvKNsteWU=
METADATA_SIZE=46866"
```

NOTE

Make sure to use a new line for every payload_properties parameter here.

- The system will update in the background. After it finishes, it will show "Update successfully applied, waiting to reboot" in the logcat.

7.2.2 Using a customized application to update the Android platform

There is a reference OTA application unter `${MY_ANDROID}/vendor/nxp-opensource/fsl_imx_demo/FSLOta`, which can do the OTA operations:

1. Get `payload_properties.txt` and `payload.bin` from a specific address.
2. Use the update_engine service to update the Android platform.

Perform the following steps to use this application:

1. Set up the HTTP server (eg., lighttpd, apache).
   You need one HTTP server to hold OTA packages.
   - For full OTA update, execute the following commands:

     ```
     cp ${MY_ANDROID}/out/target/product/evk_8mq/system/build.prop ${server_ota_folder}
     cp ${MY_ANDROID}/out/target/product/evk_8mq/evk_8mq-ota-${date}-${soc}.zip $
     {server_ota_folder}
     cd ${server_ota_folder}
     unzip evk_8mq-ota-${date}-${soc}.zip
     mv payload_properties.txt payload_properties-imx8mq.txt
     mv payload.bin payload-imx8mq.bin
     ```
   - For incremental OTA update, execute the following commands:

     ```
     cp ${old_build.prop} ${server_ota_folder}/old_build.prop
     cp ${MY_ANDROID}/out/target/product/evk_8mq/system/build.prop ${server_ota_folder}/
     build_diff.prop
     cp ${MY_ANDROID}/incremental_ota_update.zip ${server_ota_folder}
     cd ${server_ota_folder}
     unzip incremental_ota_update.zip
     mv payload_properties.txt payload_properties-imx8mq_diff.txt
     mv payload.bin payload-imx8mq_diff.bin
     echo -n "base." >> build_diff.prop
     grep "ro.build.date.utc" old_build.prop >> build_diff.prop
     ```

   For example, the server_ota_folder content is like this:

   ```
   build@server:/var/www/evk_8mq_oreo_8.1.0$ ls
   build.prop build_diff.prop payload-imx8mq.bin payload-imx8mq_diff.bin
   payload_properties-imx8mq.txt payload_properties-imx8mq_diff.txt
   ```

   NOTE

   - server_ota_folder: ${http_root}/evk_8mq_${ota_folder_suffix}_${version}.
   - ${old_build.prop} is the old image's build.prop.
   - evk_8mq-ota-${date}-${soc}.zip and incremental_ota_update.zip are built from Section 7.1.2 "Building a full update package" and Section 7.1.3 "Building an incremental update package".
   - ${ota_folder_suffix} is stored at board's /vendor/etc/ota.conf.

- ${version} can be obtained by the following command on the board's console: $getprop | grep "ro.build.version.release".
- These file and folder names should align with this example, or modify the OTA application source code correspondingly.

2. Configure the OTA server IP address and HTTP port number.
   The OTA configuration file (/vendor/etc/ota.conf) content is like this:

```
server=192.168.1.100
port=10888
ota_folder_suffix=oreo
```

   Modify it to fit the environment.

3. Open the OTA application and click the **Update** button.
   The reference application is a dialogue box activity, and can be enabled through the **Settings** -> **About tablet** -> **Additional system Update** menu. There are two buttons on the dialogue box:
   - **Upgrade**: Performs full OTA.
   - **Diff Upgrade**: Performs incremental OTA.

   Click one button to update the Android platform. After update is complete, click the **Reboot** button on the dialogue box.

NOTE

- This application uses the "ro.build.date.utc=1528987645" property to decide whether it can perform full OTA or incremental OTA.
- local utc = $getprop | grep "ro.build.date.utc".
- remote utc = cat ${server_ota_folder}/build.prop | grep "ro.build.date.utc".
- remote diff utc = cat ${server_ota_folder}/build_diff.prop | grep "ro.build.date.utc".
- remote diff base utc = cat ${server_ota_folder}/build_diff.prop | grep "base.ro.build.date.utc" (base.ro.build.date.utc should be added manually, which is the "ro.build.date.utc" value in PREVIOUS-target_files.zip's system/build.prop).
- Full OTA condition:
  - local utc < remote utc
- Incremental OTA condition:
  - local utc = remote diff base utc
  - local utc < remote diff utc

NOTE

For detailed information about A/B OTA updates, see https://source.android.com/devices/tech/ota/ab/.

8 Camera

8.1 Configuring the rear and front cameras

Property "back_camera_name" and "front_camera_name" are used to configure which camera to be used as the rear camera or front camera.

The name should be either `v4l2_dbg_chip_ident.match.name` returned from v4l2's IOCTL VIDIOC_DBG_G_CHIP_IDENT or `v4l2_capability.driver` returned from v4l2's IOCTL VIDIOC_QUERYCAP.

Camera HAL goes through all the V4L2 devices in the system. Camera HAL chooses the first matched name in property settings as the corresponding camera. Comma is used as a delimiter of different camera name among multiple-camera selection.

The following is an example set in ${MY_ANDROID}/device/fsl/evk_8mq/init.rc.

```
setprop back_camera_name mx6s-csi
setprop front_camera_name uvc
```

`media_profiles_V1_0.xml` in `/vendor/etc` is used to configure the parameters used in the recording video. NXP provides several media profile examples that help customer align the parameters with their camera module capability and device definition.

Table 12. Media profile parameters

Profile file name	Rear camera	Front camera
media_profiles_1080p.xml	Maximum to 1080P, 30FPS and 8 Mbps for recording video	Maximum to 720P, 30FPS, and 3 Mbps for recording video
media_profiles_720p.xml	Maximum to 720P, 30FPS, and 3 Mbps for recording video	Maximum to 720P, 30FPS, and 3 Mbps for recording video
media_profiles_480p.xml	Maximum to 480P, 30FPS, and 2 Mbps for recording video	Maximum to 480P, 30FPS, and 2 Mbps for recording video
media_profiles_qvga.xml	Maximum to QVGA, 15FPS, and 128 Kbps for recording video	Maximum to QVGA, 15FPS, and 128 Kbps for recording video

NOTE

Because not all UVC cameras can have 1080P, 30FPS resolution setting, it is recommended that `media_profiles_480p.xml` is used for any board's configuration, which defines the UVC as the rear camera or front camera.

9 Revision History

Table 13. Revision history

Revision number	Date	Substantive changes
N7.1.2_2.1.0_8MQ-EAR	09/2017	Initial release
O8.0.0_1.3.0_8M-PRC	01/2018	i.MX 8MQuad PRC (Beta) release
O8.1.0_1.3.0_8M	04/2018	i.MX 8MQuad RFP (GA) release
O8.1.0_1.3.1_8MQ-ear	07/2018	i.MX 8MQuad engineering release

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number AUG
Revision O8.1.0_1.3.1_8MQ-ear, 07/2018