

Android™ User's Guide

Contents

1 Overview

This document describes how to build Android Pie 9.0 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.

1	Overview.....	1
2	Preparation.....	1
3	Building the Android platform for i.MX.....	2
4	Running the Android Platform with a Prebuilt Image.....	7
5	Programming Images.....	11
6	Bootting.....	14
7	Over-The-Air (OTA) Update.....	20
8	Customized Configuration.....	23
9	Revision History.....	26



2.1 Setting up your computer

To build the Android source files, use a computer running the Linux OS. The Ubuntu 16.04 64-bit version and openjdk-8-jdk is the most tested environment for the Android Pie 9.0 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 liblz2-2 liblz2-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
$ sudo apt-get install m4
$ sudo apt-get install libz-dev
```

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-p9.0.0_1.0.0-beta.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-p9.0.0_1.0.0-beta.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-p9.0.0_1.0.0-beta/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
Android 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 "[Getting i.MX Android release source code](#)").

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_8mm-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_8mm	i.MX 8M Mini EVK Board
evk_8mq	i.MX 8M Quad EVK Board
mek_8q	i.MX 8QuadMax/i.MX 8QuadXPlus MEK 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 8M Mini EVK Board/Platform device with userdebug type.

```
$ lunch evk_8mm-userdebug
```

4. Execute the **make** command to generate the image.

Building the Android platform for i.MX

```
$ 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_8mm:

- 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.
- dtbo-imx8mm.img: Board's device tree binary. It is used to support MIPI-to-HDMI output.
- dtbo-imx8mm-dsd.img: Board's device tree binary. It is used to support MIPI-to-HDMI output and Direct Stream Digital (DSD) playback.
- dtbo-imx8mm-m4.img: Board's device tree binary. It is used to support MIPI-to-HDMI output and audio playback based on Cortex-M4 freeRTOS.
- dtbo-imx8mm-mipi-panel: Board's device tree binary. It is used to support MIPI panel output.
- vbmeta-imx8mm.img: Android Verify boot metadata image for boot-imx8mm.img.
- vbmeta-imx8mm-dsd.img: Android Verify boot metadata image for boot-imx8mm-dsd.img.
- vbmeta-imx8mm-m4.img: Android Verify boot metadata image for boot-imx8mm-m4.img.
- vbmeta-imx8mm-mipi-panel.img: Android Verify boot metadata image for boot-imx8mm-mipi-panel.img.
- 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".
- 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-imx8mm.imx: U-Boot image without padding for i.MX 8M Mini EVK.
- imx8mm_m4_demo.img: Cortex-M4 FreeRTOS image to support audio playback on the Cortex-M4 side.
- vendor.img: vendor image, which holds platform binaries. Mounted at /vendor.
- boot.img: a composite image that includes the kernel Image, ramdisk, and boot parameters.

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](#).
- To build dtbo.img, see [Building dtbo.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 8M Mini EVK board	\$ lunch evk_8mm-userdebug
i.MX 8M Quad EVK board	\$ lunch evk_8mq-userdebug
i.MX 8QuadMax/i.MX 8QuadXPlus MEK board	\$ lunch mek_8q-userdebug

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 -j4 PRODUCT-XXX userdebug 2>&1 | tee build-log.txt #XXX depends on different boards.
See the table below.
```

Table 4. Android system image production build method 1

i.MX development tool	Description	Image build command
Evaluation Kit	i.MX 8M Mini EVK	\$ make -j4 PRODUCT-evk_8mm-userdebug
Evaluation Kit	i.MX 8M Quad EVK	\$ make -j4 PRODUCT-evk_8mq-userdebug
Evaluation Kit	i.MX 8QuadMax/8QuadXPlus MEK	\$ make -j4 PRODUCT-mek_8q-userdebug

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

```
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ make -j4
```

Table 5. Android system image production build method 2

i.MX development tool	Description	Lunch configuration
Evaluation Kit	i.MX 8M Mini EVK	evk_8mm-userdebug
Evaluation Kit	i.MX 8M Quad EVK	evk_8mq-userdebug
Evaluation Kit	i.MX 8QuadMax/8QuadXPlus MEK	mek_8q-userdebug

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

3.3 Building U-Boot images

Use the following command to generate `u-boot.imx` under the Android environment:

Building the Android platform for i.MX

```
# U-Boot image for i.MX 8M Mini board
$ cd ${MY_ANDROID}
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ make bootloader -j4
```

For other platforms, use `lunch <buildName-buildType>` to set up the build configuration. For detailed build configuration, see Section 3.2 "Building Android images".

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 Image for i.MX 8M Mini, i.MX 8MQuad EVK, i.MX 8QuadMax/8QuadXPlus MEK
$ make android_defconfig
$ make KCFLAGS=-mno-android
```

The kernel images are found in `${MY_ANDROID}/out/target/product/evk_8mm/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 8M Mini EVK board
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ make bootimage -j4
```

For other platforms, use `lunch <buildName-buildType>` to set up the build configuration. For detailed build configuration, see Section 3.2 "Building Android images".

3.6 Building dtbo.img

Dtbo image holds the device tree binary of the board.

To generate dtbo.img under the Android environment, use the following commands:

```
# dtbo image for i.MX 8M Mini board
$ source build/envsetup.sh
$ lunch evk_8mm-userdebug
$ make dtboimage -j4
```

For other platforms, use `lunch <buildName-buildType>` to set up the build configuration. For detailed build configuration, see Section 3.2 "Building Android images".

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 "Programming Images" and "Boot".

Table 6. Image packages

Image package	Description
android_p9.0.0_1.0.0-beta_image_8mmevk.tar.gz	Prebuilt image and UUU script files for i.MX 8M Mini EVK board, which includes NXP extended features.
android_p9.0.0_1.0.0-beta_image_8mqevk.tar.gz	Prebuilt image and UUU script files for i.MX 8M Quad EVK board, which includes NXP extended features.
android_p9.0.0_1.0.0-beta_image_8qmek.tar.gz	Prebuilt image and UUU script files for i.MX 8QuadMax/8QuadXPlus MEK board, which includes NXP extended features.

The following tables list the detailed contents of `android_p9.0.0_1.0.0-beta_image_8mmevk.tar.gz` image package.

The table below shows the prebuilt images to support the system boot from SD on i.MX 8M Mini boards.

Table 7. Images for i.MX 8M Mini

i.MX 8M Mini SD image	Description
/u-boot-imx8mm.imx	Bootloader (with padding) for i.MX 8M Mini EVK board
/boot.img	Boot image for i.MX 8M Mini board
/system.img	System Boot image for i.MX 8M Mini board
/vendor.img	Vendor image for i.MX 8M Mini 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
/imx8mm_m4_demo.img	Cortex-M4 FreeRTOS image for i.MX 8M Mini board
/dtbo-imx8mm.img	Device Tree image for i.MX 8M Mini board to support MIPI-to-HDMI output.
/dtbo-imx8mm-dsd.img	Device Tree image for i.MX 8M Mini board to support MIPI-to-HDMI output and DSD playback.
/dtbo-imx8mm-m4.img	Device Tree image for i.MX 8M Mini board to support MIPI-to-HDMI output and audio playback based on Cortex-M4 FreeRTOS.
/dtbo-imx8mm-mipi-panel.img	Device Tree Image for i.MX 8M Mini board to support MIPI panel output.
/vbmata-imx8mm.img	Android Verify Boot metadata Image for i.MX 8M Mini EVK board to support MIPI-to-HDMI output

Table continues on the next page...

Table 7. Images for i.MX 8M Mini (continued)

/vbmata-imx8mm-dsd.img	Android Verify Boot metadata Image for i.MX 8M Mini board to support MIPI-to-HDMI output and DSD playback
/vbmata-imx8mm-m4.img	Android Verify Boot metadata image for i.MX 8M Mini board to support MIPI-to-HDMI output and Cortex-M4 playback.
/vbmata-imx8mm-mipi-panel.img	Android Verify Boot metadata image for i.MX 8M Mini board to support MIPI panel output.

The table below describes the UUU scripts in android_p9.0.0_1.0.0-beta_image_8mmevk.tar.gz. They are used with the UUU binary file to download the images above into the board.

Table 8. UUU scripts

UUU script name	Function
uuu-android-mx8mm-evk-emmc.lst	Used with the UUU binary file to download image files into eMMC. The m4_os partition is not flashed.
uuu-android-mx8mm-evk-sd.lst	Used with the UUU binary file to download image files into the SD card. The m4_os partition is not flashed.
uuu-android-mx8mm-evk-emmc-m4.lst	Used with the UUU binary file to download image files into eMMC. The m4_os partition is flashed.
uuu-android-mx8mm-evk-sd-m4.lst	Used with the UUU binary file to download image files into the SD card. The m4_os partition is flashed.

The following tables list the detailed contents of android_p9.0.0_1.0.0-beta_image_8mqevk.tar.gz image package.

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

Table 9. Images for i.MX 8M Quad EVK

i.MX 8MQuad EVK SD image	Description
u-boot-imx8mq.imx	Bootloader (with padding) for i.MX 8MQuad EVK board.
/boot.img	Boot image for i.MX 8M Quad EVK B3/B4 board
/system.img	System Boot image for i.MX 8M Quad EVK B3/B4 board
/vendor.img	Vendor image for i.MX 8M Quad 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.
/dtbo-imx8mq.img	Device Tree image for i.MX 8M Quad EVK B4 board to support HDMI output
/dtbo-imx8mq-dsd.img	Device Tree image for i.MX 8M Quad EVK B4 board to support HDMI output and DSD playback
/dtbo-imx8mq-mipi.img	Device Tree image for i.MX 8M Quad EVK B4 board to support MIPI-to-HDMI output
/dtbo-imx8mq-dual.img	Device Tree image for i.MX 8M Quad EVK B4 board to support HDMI and MIPI-to-HDMI dual output
/dtbo-imx8mq-mipi-panel.img	Device Tree image for i.MX 8M Quad EVK B4 board to support MIPI panel output

Table continues on the next page...

Table 9. Images for i.MX 8M Quad EVK (continued)

/dtbo-imx8mq-b3.img	Device Tree image for i.MX 8M Quad EVK B3 board to support HDMI output
/dtbo-imx8mq-mipi-b3.img	Device Tree image for i.MX 8M Quad EVK B3 board to support MIPI-to-HDMI output
/dtbo-imx8mq-mipi-panel-b3.img	Device Tree image for i.MX 8M Quad EVK B3 board to support MIPI panel output
vbmeta-imx8mq.img	Android Verify Boot metadata image for i.MX 8M Quad EVK board to support HDMI output.
vbmeta-imx8mq-dsd.img	Android Verify Boot metadata image for i.MX 8M Quad EVK board to support HDMI output and DSD playback.
vbmeta-imx8mq-mipi.img	Android Verify Boot metadata image for i.MX 8M Quad EVK board to support MIPI-to-HDMI output.
vbmeta-imx8mq-dual.img	Android Verify Boot metadata image for i.MX 8M Quad EVK board to support HDMI and MIPI-to-HDMI dual output.
vbmeta-imx8mq-mipi-panel.img	Android Verify Boot metadata image for i.MX 8M Quad EVK board to support MIPI panel output.
/vbmeta-imx8mq-b3.img	Android Verify Boot metadata image for i.MX 8M Quad EVK B3 board to support HDMI output
/vbmeta-imx8mq-mipi-b3.img	Android Verify Boot metadata image for i.MX 8M Quad EVK B3 board to support MIPI-to-HDMI output
/vbmeta-imx8mq-mipi-panel-b3.img	Android Verify Boot metadata image for i.MX 8M Quad EVK B3 board to support MIPI panel output

The table below describes the UUU scripts in android_p9.0.0_1.0.0-beta_image_8mqevk.tar.gz. They are used with the UUU binary file to download the images above into the board.

Table 10. UUU scripts

UUU script name	Function
uuu-android-mx8mq-evk-emmc.lst	Used with the UUU binary file to download image files into eMMC.
uuu-android-mx8mq-evk-sd.lst	Used with the UUU binary file to download image files into the SD card.

The following tables list the detailed contents of android_p9.0.0_1.0.0-beta_image_8mqmek.tar.gz image package.

The table below shows the prebuilt images to support the system boot from SD or eMMC on i.MX 8QuadMax MEK boards.

Table 11. Images for i.MX 8M Quad EVK

i.MX 8QuadMax MEK image	Description
/u-boot-imx8qm.imx	Bootloader (with padding) for i.MX 8QuadMax MEK board.
/u-boot-imx8qxp.imx	Bootloader (with padding) for i.MX 8QuadXPlus MEK board
/uuu-u-boot-imx8qm.imx	Bootloader used by UUU for the i.MX 8QuadMax MEK board. It is not flashed to MMC.
/uuu-u-boot-imx8qxp.imx	Bootloader used by UUU for the i.MX 8QuadXPlus MEK board. It is not flashed to MMC.

Table continues on the next page...

Table 11. Images for i.MX 8M Quad EVK (continued)

/boot.img	Boot image for i.MX 8QuadMax/8QuadXPlus MEK board
/system.img	System Boot image for i.MX 8QuadMax/8QuadXPlus MEK board
/vendor.img	Vendor image for i.MX 8QuadMax/8QuadXPlus MEK board
/partition-table.img	GPT table image for 16 GB boot storage.
/partition-table-7GB.img	GPT table image for 8 GB boot storage.
/partition-table-28GB.img	GPT table image for 32 GB boot storage.
/vbmeta-imx8qm.img	Android Verify Boot metadata image for i.MX 8QuadMax MEK board to support LVDS-to-HDMI/MIPI-to-HDMI display.
/vbmeta-imx8qm-hdmi.img	Android Verify Boot metadata image for i.MX 8QuadMax MEK board to support physical HDMI display.
/vbmeta-imx8qxp.img	Android Verify Boot metadata image for i.MX 8QuadXPlus MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display with dual camera support.
/vbmeta-imx8qxp-ov5640mipi.img	Android Verify Boot metadata image for i.MX 8QuadXPlus MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display with single MIPI camera support.
/dtbo-imx8qm.img	Device Tree image for i.MX 8QuadMax MEK board to support LVDS-to-HDMI/MIPI-to-HDMI display.
/dtbo-imx8qm-hdmi.img	Device Tree image for i.MX 8QuadMax MEK board to support physical HDMI display.
/dtbo-imx8qxp.img	Device Tree image for i.MX 8QuadXPlus MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display with dual camera support.
/dtbo-imx8qxp-ov5640mipi.img	Device Tree image for i.MX 8QuadXPlus MEK board to support single LVDS-to-HDMI/MIPI-to-HDMI or dual LVDS-to-HDMI display with single MIPI camera support.

The table below describes the UUU scripts in android_p9.0.0_1.0.0-beta_image_8qmek.tar.gz. They are used with the UUU binary file to download the images above into the board.

Table 12. UUU scripts

UUU script name	Function
uuu-android-mx8qm-mek-emmc.lst	Used for i.MX 8QuadMax MEK board with UUU binary file to download image files into eMMC.
uuu-android-mx8qm-mek-sd.lst	Used for i.MX 8QuadMax MEK board with UUU binary file to download image files into the SD card
uuu-android-mx8qxp-mek-emmc.lst	Used for i.MX 8QuadXPlus MEK board with UUU binary file to download image files into eMMC.
uuu-android-mx8qxp-mek-sd.lst	Used for i.MX 8QuadXPlus MEK board with UUU binary file to download image files into the SD card.

NOTE

boot.img is an Android image that stores zImage and ramdisk together. It can also store other information such as the kernel boot command line and machine name. This information can be configured in android.mk. It can avoid touching 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 minimum, 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 [Bootimg](#).

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

- UUU and UUU script file to download all images to the eMMC/SD card.
- fsl-sdcard-partition.sh to download all images to the SD card.
- fastboot_imx_flashall script to download all images to the eMMC/SD storage.

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
- GPT table image: partition-table.img
- Android dtbo image: dtbo.img
- Android boot image: boot.img
- Android system image: system.img
- Android verify boot metadata image: vbmeta.img
- Android vendor image: vendor.img

5.1.1 Storage partitions

The layout of the eMMC 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 partition 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 13. Storage partitions

Partition type/index	Name	Start offset	Size	File system	Content
N/A	bootloader	0 KB (i.MX 8QuadMax eMMC) or 32 KB (i.MX 8QuadXPlus, i.MX 8QuadMax SD card) or 33 KB (i.MX 8M Quad, i.MX 8M Mini)	4 MB	N/A	bootloader
1	dtbo_a	8 MB	4 MB	N/A	dtbo.img
2	dtbo_b	Follow dtbo_a	4 MB	N/A	dtbo.img
3	boot_a	Follow dtbo_b	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
4	boot_b	Follow boot_a	48 MB	boot.img format, a kernel + recovery ramdisk	boot.img
5	system_a	Follow boot_b	1536 MB	EXT4. Mount as / system	Android system files under / system/dir
6	system_b	Follow system_a	1536 MB	EXT4. Mount as / system	Android system files under / system/dir
7	misc	Follow system_b	4 MB	N/A	For recovery store bootloader message, reserve
8	metadata	Follow datafootor	2 MB	N/A	For system slide show
9	persistdata	Follow metadata	1 MB	N/A	Option to operate unlock \unlock
10	vendor_a	Follow persistdata	112 MB	EXT4. Mount at / vendor	vendor.img
11	vendor_b	Follow vendor_a	112 MB	EXT4. Mount at / vendor	vendor.img
12	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.
13	fbmisc	Follow userdata	1 MB	N/A	For storing the state of lock \unlock
14	vbmeta_a	Follow fbmisc	1 MB	N/A	For storing the verify boot's metadata
15	vbmeta_b	Follow vbmeta_a	1 MB	N/A	For storing the verify boot's metadata

To create these partitions, use UUU 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 and download images to them 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 imx8mm, imx8mq, imx8qm, imx8qxp.
```

NOTE

- The minimum size of the SD card is 8 GB bytes.
- If the SD card is 8 GB, use `sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> -c 7 /dev/sdX` to flash images.
- If the SD card is 16 GB, use `sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> /dev/sdX` to flash images.
- If the SD card is 32 GB, use `sudo ./device/fsl/common/tools/fsl-sdcard-partition.sh -f <soc_name> -c 28 /dev/sdX` to flash images.
- /dev/sdX, the X is the disk index from 'a' to 'z', which may be different on each Linux PC.
- Unmount all the SD card partitions before running the script.
- Put related bootloader, boot image, system image, and vbmeta image in your current directory.
- This script needs `simg2img` tool to be installed on your PC. The `simg2img` is a tool that converts sparse system image to raw system image on the host PC running Linux OS. The `android-tools-fsutils` package includes the `simg2img` command for Ubuntu Linux.

5.1.2 Downloading images with UUU

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

5.1.3 Downloading images with fastboot_imx_flashall script

UUU can be used to flash the Android system image into the board, but it needs to make the board enter serial down mode first, and make the board enter boot mode once flashing is finished.

A new `fastboot_imx_flashall` script is supported to use fastboot to flash the Android system image into the board. It is more flexible. To use the new script, the board must be able to enter fastboot mode and the device must be unlocked. The table below lists the `fastboot_imx_flashall` scripts.

Table 14. fastboot_imx_flashall script

Name	Host system to execute the script
<code>fastboot_imx_flashall.sh</code>	Linux OS
<code>fastboot_imx_flashall.bat</code>	Windows OS

With the help of `fastboot_imx_flashall` scripts, you do not need to use fastboot to flash Android images one-by-one manually. These scripts will automatically flash all images with only one command.

The way to use these scripts is follows:

- Linux shell script usage: `sudo fastboot_imx_flashall.sh <option>`
- Windows batch script usage: `fastboot_imx_flashall.bat <option>`

Options:

```
-h          Displays this help message
-f soc_name Flashes the Android image file with soc_name
-a          Only flashes the image to slot_a
```

Booting

```
-b          Only flashes the image to slot_b
-c card_size Optional setting: 7 / 14 / 28
             If it is not set, use partition-table.img (default).
             If it is set to 7, use partition-table-7GB.img for 8 GB SD card.
             If it is set to 14, use partition-table-14GB.img for 16 GB SD card.
             If it is set to 28, use partition-table-28GB.img for 32 GB SD card.
             Make sure that the corresponding file exists on your platform.

-m          Flashes the Cortex-M4 image.
-d dev      Flash dtbo, vbmeta, and recovery image file with dev.
             If it is not set, use default dtbo, vbmeta, and recovery image.
-e          Erases user data after all image files are flashed.
-l          Locks the device after all image files are flashed.
-D directory Directory of images.
             If this script is execute in the directory of the images, it does not
need to use this option.
-s ser_num  Serial number of the board.
             If only one board connected to computer, it does not need to use this
option
```

NOTE

- -f option is mandatory. SoC name can be imx8qm or imx8qxp.
- Boot the device to U-Boot fastboot mode, and then execute these scripts. The device should be unlocked first.

Example:

```
sudo ./fastboot_imx_flashall.sh -f imx8qm -a -e -D /imx_pi9.0/mek_8q_car/
```

Options explanation:

- -f imx8qm: Flashes images for i.MX 8QuadMax MEK Board.
- -a: Only flashes slot a.
- -e: Erases user data after all image files are flashed.
- -D /imx_pi9.0/mek_8q_car/: Images to be flashed are in the directory of /imx_pi9.0/mek_8q_car/.

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 8M Mini EVK board

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

Table 15. Boot device switch settings

Boot device switch	SW1101 (1-8 bit)	SW1102 (1-8 bit)
SD boot	01000100	00110100
eMMC boot	01110010	00001010

Table 16. Boot mode switch settings

Boot mode switch	SW1101 (1-2 bit)
Download mode	10

To test booting from SD, change the board Boot_Mode switch to SW1101 01000100 (1-8 bit) and SW1102 00110100 (1-8 bit).

To test booting from eMMC, change the board Boot_Mode switch to SW1101 01110010 (1-8 bit) and SW1102 00001010 (1-8 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 bootarg console=ttymxcl,115200 earlycon=ec_imx6q,0x30890000,115200 init=/
init androidboot.console=ttymxcl consoleblank=0 androidboot.hardware=freescale cma=800M
androidboot.primary_display=imx-drm firmware_class.path=/vendor/firmware
transparent_hugepage=never [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.

6.1.2 Booting from SD/eMMC on the i.MX 8M Quad EVK board

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

Table 17. Boot device switch settings

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

Table 18. Boot mode switch settings

Boot mode switch	Download Mode (MfgTool mode)	Boot mode
SW802 (1-2 bit)	01	10

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

To test booting from eMMC, change the board Boot_Mode switch to 10 (1-2 bit) and SW801 0010 (1-4 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=ttymxcl,115200 earlycon=imxuart,0x30860000,115200 init=/
init androidboot.gui_resolution=1080p androidboot.console=ttymxcl consoleblank=0
```

Booting

```
androidboot.hardware=freescale cma=1280M androidboot.primary_display=imx-drm
firmware_class.path=/vendor/firmware androidboot.fbTileSupport=enable [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).

6.1.3 Booting from SD/eMMC on the i.MX 8QuadMax MEK board

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

Table 19. Boot device switch settings

i.MX 8QuadMax boot switch	Download mode (UUU mode)	SD boot	eMMC boot
SW2 Boot_Mode (1-6 bit)	001000	001100	001000

To test booting from SD, change the board Boot_Mode switch to 001100 (1-6 bit).

To test booting from eMMC, change the board Boot_Mode switch to 001000 (1-6 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=ttyLP0,115200 earlycon=lpuart32,0x5a060000,115200,115200
init=/init androidboot.console=ttyLP0 consoleblank=0 androidboot.hardware=freescale
androidboot.fbTileSupport=enable cma=800M@0x960M-0xe00M 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.

6.1.4 Booting from SD/eMMC on the i.MX 8QuadXPlus MEK board

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

Table 20. Boot device switch settings

i.MX 8QuadXPlus boot switch	Download mode (UUU mode)	SD boot	eMMC boot
SW2 Boot_Mode (1-4 bit)	1000	1100	0100

To test booting from SD, change the board Boot_Mode switch to 1100 (1-4 bit).

To test booting from eMMC, change the board Boot_Mode switch to 0100 (1-4 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=ttyLP0,115200 earlycon=lpuart32,0x5a060000,115200,115200
init=/init androidboot.console=ttyLP0 consoleblank=0 androidboot.hardware=freescale
androidboot.fbTileSupport=enable cma=800M@0x960M-0xe00M 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.

6.2 Boot-up configurations

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

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
```

- 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 21. Kernel boot parameters

Kernel parameter	Description	Typical value	Used when
console	Where to output kernel log by printk.	console=ttymx0	i.MX 8M Mini use console=ttymx0.
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".
androidboot.console	The Android shell console. It should be the same as console=.	androidboot.console=ttymx0	To use the default shell job control, such as Ctrl+C to terminate a running process, set this for the kernel.
cma	CMA memory size for GPU/VPU physical memory allocation.	cma=800M or cma=1280M or cma=800M@0x960M-0xe00M <ul style="list-style-type: none"> For i.MX 8M Mini and i.MX 8QuadMax, it is 800 MB by default. For i.MX 8M Quad, it is 1280 MB by default. For i.MX 8QuadXPlus and 8QuadMax, it is 800 MB by default. 	Start address is 0x96000000 and end address is 0xDFFFFFFF. The CMA size can be configured to other value, but cannot exceed 1184 MB, because the Cortex-M4 core will also allocate memory from CMA and Cortex-M4 cannot use the memory larger than 0xDFFFFFFF.
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 Pie 9.0 CTS requirement: serial input should be disabled by default. Setting this argument enables console serial input, which will violate the CTS requirement. Setting this argument will also bypass all the selinux rules defined in Android system. It is recommended to set this argument for internal developer.
androidboot.primary_display	It is used to choose and fix primary display.	androidboot.primary_display=imx-drm	androidboot.primary_display=mxsfb-drm is only used for MIPI display.
androidboot.lcd_density	It is used to set the display density and overwrite ro.sf.lcd_density in init.rc for MIPI-to-HDMI display.	androidboot.lcd_density=160	-
androidboot.displaymode	It is used to configure the kernel/driver work mode/fps.	<ul style="list-style-type: none"> 4k display should be configured as: androidboot.displaymode=4k. The default fps is 60fps. To configure fps, change this value to 4kp60/4kp50/4kp30. 	The system will find out and work at the best display mode, and display mode can be changed through this bootargs.

Table continues on the next page...

Table 21. Kernel boot parameters (continued)

Kernel parameter	Description	Typical value	Used when
		<ul style="list-style-type: none"> 1080p display should be configured as: androidboot.displaymode=1080p. The default fps is 60fps. To configure fps, change this value to 1080p60/1080p50/1080p30. 720p display should be configured as: androidboot.displaymode=720p. The default fps is 60fps. To configure fps, change this value to 720p60/720p50/720p30. 480p display should be configured as: androidboot.displaymode=480p. The default fps is 60fps. To configure fps, change this value to 480p60/480p50/480p30. 	
androidboot.fbTileSupport	It is used to enable framebuffer super tile output on i.MX 8MQuad EVK.	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	-
transparent_hugepage	It is used to change the sysfs boot time defaults of Transparent Hugepage support.	transparent_hugepage=never/always/madvise	i.MX 8M Mini sets transparent_hugepage=never to have only 2GB memory.

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 convenience 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
```

Over-The-Air (OTA) Update

- 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 Over-The-Air (OTA) 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_8mm-userdebug
$ make target-files-package -j4
```

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

```
${MY_ANDROID}/out/target/product/evk_8mm/obj/PACKAGING/target_files_intermediates/evk_8mm-
target_files-${date}.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_8mm-userdebug
$ make otapackage -j4
```

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

```
${MY_ANDROID}/out/target/product/evk_8mm/evk_8mm-ota-${date}.zip
```

evk_8mm-ota-\${date}.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.

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 Using 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=0fSBbXonyTjaAzMpwTBgM9AVt1BeyOigpCCgkoOfHKY=
 - FILE_SIZE=379074366
 - METADATA_HASH=Icrs3NqoglyzppyCZouWKbo5f08IPokhlUfHDmz77WQ=
 - 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=0fSBbXonyTjaAzMpwTBgM9AVt1BeyOigpCCgkoOfHKY=
FILE_SIZE=379074366
METADATA_HASH=Icrs3NqoglyzppyCZouWKbo5f08IPokhlUfHDmz77WQ/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 under `${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_8mm/system/build.prop ${server_ota_folder}
cp ${MY_ANDROID}/out/target/product/evk_8mm/evk_8mm-ota-${date}.zip ${server_ota_folder}
cd ${server_ota_folder}
unzip evk_8mm-ota-${date}.zip
```

- 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_8mm/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
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_8mm_pie_9$ ls
build.prop build_diff.prop payload.bin payload_diff.bin payload_properties.txt
payload_properties_diff.txt
```

NOTE

- `server_ota_folder`: `${http_root}/evk_8mm-${ota_folder_suffix}-${version}`.
- `${old_build.prop}` is the old image's `build.prop`.
- `evk_8mm-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 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=pie
```

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 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

The OTA package includes the DTBO image, which stores the board's DTB. There may be many DTS for one board. For example, in \${MY_ANDROID}/device/fsl/imx8q/mek_8q/BoardConfig.mk:

```
TARGET_BOARD_DTS_CONFIG ?= imx8mm:fsl-imx8mm-evk.dtb imx8mm-mipi-panel:fsl-imx8mm-evk-rm67191.dtb imx8mm-dsd:fsl-imx8mm-evk-ak4497.dtb imx8mm-m4:fsl-imx8mm-evk-m4.dtb
```

The OTA package only includes the first DTS_CONFIG definition DTS: fsl-imx8mm-evk.dtb, so the default OTA package can only be applied for evk_8mm with single HDMI display. To generate an OTA package for evk_8mm with a MIPI panel display, modify the TARGET_BOARD_DTS_CONFIG as follows:

```
TARGET_BOARD_DTS_CONFIG ?= imx8mm-mipi-panel:fsl-imx8mm-evk-rm67191.dtb imx8mm:fsl-imx8mm-evk.dtb imx8mm-dsd:fsl-imx8mm-evk-ak4497.dtb imx8mm-m4:fsl-imx8mm-evk-m4.dtb
```

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

8 Customized Configuration

8.1 How to change boot command line in boot.img

After boot.img is used, the default kernel boot command line is stored inside the image. It packages together during android build.

You can change this by changing BOARD_KERNEL_CMDLINE's definition in \${MY_ANDROID}/device/fsl/{product}/BoardConfig.mk.

NOTE

Replace {product} with your product, eg., evk_8mm.

8.2 How to configure 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_8mm/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 22. Media profile parameters

Profile file name	Rear camera	Front camera
<code>media_profiles_1080p.xml</code>	Maximum to 1080P, 30FPS and 8 Mbps for recording video	Maximum to 720P, 30FPS, and 3 Mbps for recording video
<code>media_profiles_720p.xml</code>	Maximum to 720P, 30FPS, and 3 Mbps for recording video	Maximum to 720P, 30FPS, and 3 Mbps for recording video
<code>media_profiles_480p.xml</code>	Maximum to 480P, 30FPS, and 2 Mbps for recording video	Maximum to 480P, 30FPS, and 2 Mbps for recording video
<code>media_profiles_qvga.xml</code>	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.

8.3 How to configure the logical display density

The Android UI framework defines a set of standard logical densities to help application developers target application resources.

Device implementations must report one of the following logical Android framework densities:

- 120 dpi, known as 'ldpi'
- 160 dpi, known as 'mdpi'
- 213 dpi, known as 'tvdpi'
- 240 dpi, known as 'hdpi'
- 320 dpi, known as 'xhdpi'
- 480 dpi, known as 'xxhdpi'

Device implementations should define the standard Android framework density that is numerically closest to the physical density of the screen, unless that logical density pushes the reported screen size below the minimum supported.

To configure the logical display density for framework, you must define the following line in `${MY_ANDROID}/device/fsl/{product}/init.rc`:

```
setprop ro.sf.lcd_density <density>
```

NOTE

Replace `{product}` with your product, eg., `evk_8mm`.

8.4 How to enable low-power audio

The "DirectAudioPlayer" application is provided to support audio playback from DirectOutputThread. The source code is in `${MY_ANDROID}/vendor/nxp-opensource/fsl_imx_demo/DirectAudioPlayer`. After the "vendor.audio.lpa.enable" property is set to 1, low-power audio can be enabled. In this situation, audio can keep playing even if the system enters suspending mode.

By default, the music stream plays from MixedThread. To make stream play from DirectOutputThread, add the `AUDIO_OUTPUT_FLAG_DIRECT` flag to the related tracks. On the Android Application layer, there is no `AUDIO_OUTPUT_FLAG_DIRECT` flag to specify DirectOutputThread explicitly. Instead, use `FLAG_HW_AV_SYNC` when there is "new AudioTrack" in the application. Then the Android audio framework will add `AUDIO_OUTPUT_FLAG_DIRECT` for this track, and this stream will play from DirectOutputThread.

In low-power audio mode, the default audio period time is 1 second, and the whole buffer can hold 60 seconds data. These two parameters can be configured by the `vendor.audio.lpa.period_ms` and `vendor.audio.lpa.hold_second` properties as follows:

```
> setprop vendor.audio.lpa.hold_second 60
> setprop vendor.audio.lpa.period_ms 1000
```

To enable low-power audio, perform the following steps:

1. Flash `boot-imx8mm-m4.img`, `imx8mm_m4_demo.img`, and `vbmeta-imx8mm-m4.img` to support audio playback based on Cortex-M4 FreeRTOS.
2. Add `bootmcu` to `bootcmd` in U-Boot command line, see Section 3.4.2 "Bootting with Single MIPI-to-HDMI display and audio playback based on Cortex-M4 FreeRTOS" in the *Android™ Quick Start Guide (AQSUG)*.
3. Run the following command to enable low-power audio mode:

```
> su
> setprop vendor.audio.lpa.enable 1
> pkill audioserver
```

4. Push the .wav audio files to /sdcard/. It is better to use a long duration audio file.
5. Disable the following system sounds:

```
Settings -> Sound -> Touch sounds
Settings -> Sound -> Screen locking sounds
Settings -> Sound -> Charging sounds
```

6. Open the DirectAudioPlayer application, and select a file from the spinner. The file selected is listed under the spinner.
7. Click the **Play** button to play audio.
8. Press the ON/OFF button on the board. The system then enters suspend mode, and the audio can keep playing.

NOTE

Only i.MX 8M Mini EVK board supports this feature.

9 Revision History

Table 23. Revision history

Revision number	Date	Substantive changes
P9.0.0_1.0.0-beta	11/2018	Initial release

How to Reach Us:**Home Page:**nxp.com**Web Support:**nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μ Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.

Document Number AUG
Revision P9.0.0_1.0.0-beta, 11/2018

