Android Solution

User Guide

Issue **09**

Date **2016-05-26**

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.hisilicon.com

Email: support@hisilicon.com

About This Document

Purpose

This document describes the methods and precautions for using and debugging Android Solution.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3716C	V2XX
Hi3716M	V4XX
Hi3719C	V1XX
Hi3719M	V1XX
Hi3718C	V1XX
Hi3718M	V1XX
Hi3798C	V1XX
Hi3796C	V1XX
Hi3798M	V1XX
Hi3796M	V1XX
HiSTBAndroid	V500R001
HiSTBAndroid	V600R001

Intended Audience

This document is intended for:

- Technical support personnel

- Software development engineers

Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

Issue 09 (2016-05-26)

This issue is the ninth official release, which incorporates the following changes:

Chapter 7 Standby

In section 7.2.3, the precautions to be taken when the light standby function is used are added.

Chapter 21 Memory Optimization of the 1 GB Version

This chapter is added.

Issue 08 (2015-07-24)

This issue is the eighth official release, which incorporates the following changes:

Chapter 18, chapter 19, and chapter 20 are added.

Issue 07 (2015-03-27)

This issue is the seventh official release, which incorporates the following changes:

Chapter 7 Standby

Section 7.2.3 is added.

Issue 06 (2014-11-05)

This issue is the sixth official release, which incorporates the following changes:

Chapter 9 Quick Boot

Section 9.3.2 is modified.

Chapter 12 HiMiracast

Section 12.2.2.1 is modified.

Issue 05 (2014-09-30)

This issue is the fifth official release, which incorporates the following changes:

Chapters 8 and 9 are added.

Chapters 11 and 12 are modified.

Issue 04 (2014-08-14)

This issue is the fourth official release, which incorporates the following changes:

Section 12.4 is added, and other modifications are made to change the supported chips from the Hi371X series to Hi379X series.

Issue 03 (2014-04-10)

This issue is the third official release, which incorporates the following changes:

Chapter 9 HiMultiScreen

Section 9.2.4 is added.

Issue 02 (2013-12-20)

This issue is the second official release, which incorporates the following changes:

Chapters 6 and 7 are added. Chapter 10 is updated.

Some modifications are made to support Android 4.4 and Hi3716M V400.

Issue 01 (2013-11-25)

This issue is the first official release, which incorporates the following changes:

Chapter 4 Recovery

Section 4.2.5 is added, and section 4.2.6 is modified.

Issue 00B02 (2013-09-30)

This issue is the second draft release, which incorporates the following changes:

Descriptions of the Recovery, Network File System (NFS), SAMBA, HiMiracast, Procrank, and Dumpsys are added.

Issue 00B01 (2013-08-31)

This issue is the first draft release.

Contents

Figures

Tables

1 Introduction

This document describes module configurations of Android Solution, reference practice of HiSilicon, and usage and configuration of common tool commands.

2 IR Remote Control

NOTE

This chapter takes Hi3798M V100 as an example. The configurations are similar for other chips.

2.1 Function Description

This version supports two types of remote control drivers.

- ir_std

  Supports four protocols:

  - NEC with simple repeat code
  - NEC with full repeat code
  - SONY
  - TC9012

  Uses the IR module of the chip for parsing, providing a high speed but supporting a limited number of protocols.

- ir_s2

  Supports six protocols:

  - NEC with simple repeat code
  - NEC with full repeat code
  - SONY
  - TC9012
  - RC5
  - RC6

  Uses the IR module of the chip to receive IR waves and uses software for parsing, supporting a large number of protocols but providing a low speed.

This chapter describes how to use the IR remote control and provides guidance for you to install software on your remote control.

To install software on your remote control, perform the following steps:

Step 1 Set the IR driver type.

Step 2 Map the lower-layer key values.

----End

2.2 Usage

2.2.1 Setting the IR Driver Type

The ir_std driver is selected by default. You can access the driver type setting menu by running the following commands:

```
cd ./device/hisilicon/bigfish/sdk
make menuconfig SDK_CFGFILE=configs/hi3798mv100/XXXX
```

XXXX indicates the software development kit (SDK) configuration file.

To set the IR driver type, perform the following steps:

Step 1 Run the following commands:

```
cd ./device/hisilicon/bigfish/sdk
make menuconfig
SDK_CFGFILE=configs/hi3798mv100/hi3798mdmo_hi3798mv100_android_cfg.mak
```

Step 2 Choose **Msp** > **IR Config**.

Two types of IR drivers are available, as shown in Figure 2-1 and Figure 2-2.

Figure 2-1 Selecting the ir_std driver

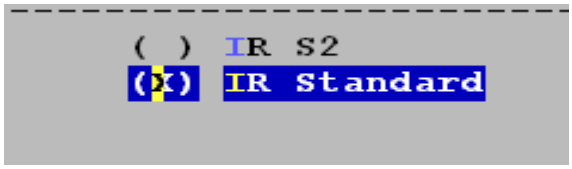
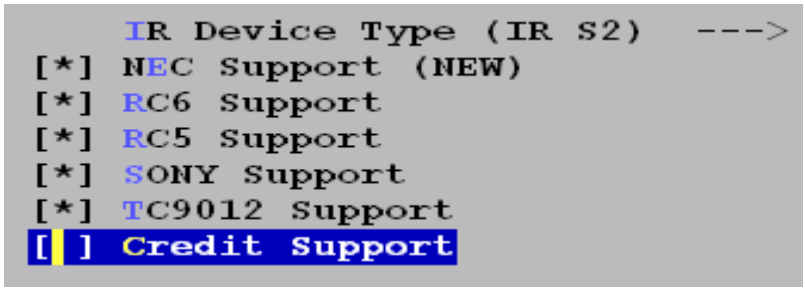


Figure 2-2 Selecting the IR_S2 driver



Step 3 Save the settings and exit. The modified configuration file is stored in the following directory:

```
./device/hisilicon/bigfish/sdk/configs/hi3798mv100
```

Step 4 Access the root directory and compile the kernel again.

----End

2.2.2 Mapping the Lower-Layer Key Values

The **key.xml** file describes the mapping between remote control key values and Linux key values. The **key.xml** file is stored in the following directory:

```
device/hisilicon/bigfish/system/ir_user/key_pars/key.xml
```

NOTE

The **key.xml** file complies with the standard Extensible Markup Language (XML) format.

To map the lower-layer key values, perform the following steps:

Step 1 View **hisi-key** in the **key.xml** file and understand the mapping format.

For example:

```
<key value="0x35caff00" name="KEY_UP"        />     <!--key up-->
```

where:

- **0x35caff00**: indicates a remote control key value.
- **KEY_UP**: indicates a Linux key value.
- **<!--key up-->**: indicates the comment.

Add or modify key values in the preceding format.

Step 2 Add the mapping of a new remote control key value to the **key.xml** file.

Insert the mapping between **<other-key>** and **</other-key>**.

NOTE

You can also modify the values in **hisi-key**.

For example:

```
<other-key>
<key value="0xed12f708" name="KEY_UP"        />
<key value="0xeb14f708" name="KEY_DOWN"     />
</other-key>
```

Step 3 Verify the mapping.

Push the modified **key.xml** file to the *board*/**etc**/ directory and run the following command:

```
android_ir_user -D
```

where:

- **android_ir_user**: indicates the IR user mode application that starts in **init.rc**.
- **–D**: indicates the output debugging information that is used only for debugging.

View the key value mappings of default HiSilicon remote controls and check the information below **remote controller name=other-key**. The information is displayed as follows:

```
remote controller name= other-key
```

```
    linux_keycode=0x0067, ir_keycode=0xed12f708, KEY_UP
    linux_keycode=0x006c, ir_keycode=0xeb14f708, KEY_DOWN
Create ir thread
```

NOTE

Check the value of **ir_keycode**. The information displayed must be consistent with the required mappings.

Step 4 Press a key on the remote control. The screen responds properly. (The prerequisite is that the remote control key values are mapped to Linux key values. The Android device directly responds to common operations with the Up, Down, Left, Right, and OK keys.)

CAUTION

At present, two remote controls are supported by default:

- HiSilicon default remote control (**hisi-key**)
- Customized remote control (**other-key**)

The number of remote controls is controlled by **#define KEY_ARRY_NUM 2** in the **ir_user.c** file. You cannot add remote controls in the **key.xml** file.

----End

3 UBI File System

3.1 Function Description

The UBI file system applies to the system, userdata, cache, and sdcard partitions in the SPI and NAND solution.

This chapter describes how to use the UBI file system, containing the following sections:

- Modifying the Partition List
- Modifying the bootargs
- Mounting UBI Partitions
- Generating UBI Images
- Burning UBI Images

Skip the first three operations if the default partition list and bootargs are used.

NOTE

Hi3798M V100 is used as an example in the following descriptions. The descriptions also apply to other chips.

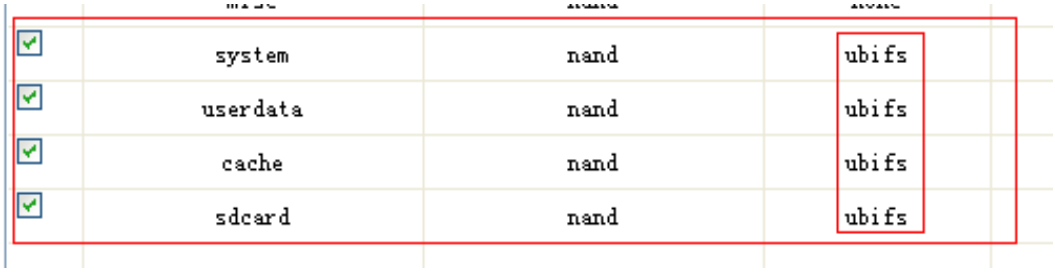
3.1.1 Modifying the Partition List

Select **ubifs** for partitions that require the UBI file system from the partition list, as shown in Figure 3-1. The default partition list (**Hi3798MV100-nand.xml**) is stored in the following directory:

```
device/hisilicon/Hi3798MV100/prebuilts/Hi3798MV100-nand.xml
```

ubifs is selected for the system, cache, userdata, and sdcard partitions by default, as shown in Figure 3-1. After you add a partition that requires the UBI file system, select **ubifs** for the partition.

Figure 3-1 UBI partitions



3.1.2 Modifying the bootargs

Add **ubi.mtd=***xxx* for UBI partitions in the bootargs. *xxx* indicates the name of a partition.

The bootargs file (**Hi3798MV100-nand.txt for example**) is stored in the following directory:

```
device/hisilicon/Hi3798MV100/etc/bootargs_ Hi3798MV100-nand.txt
```

The file contains the system, userdata, cache, and sdcard UBI partitions. The file content is as follows:

```
bootargs=mem=1G console=ttyAMA0,115200
mtdparts=hi_sfc:512k(fastboot),64k(bootargs),3456k(recovery),64k(deviceinfo);hinand:8M(baseparam),8M(pqparam),20M(logo),20M(logobak),40M(fastplay),40M(fastplayback),40M(kernel),20M(misc),500M(system),1024M(userdata),100M(cache),-(sdcard) ubi.mtd=system ubi.mtd=userdata ubi.mtd=cache ubi.mtd=sdcard
```

3.1.3 Mounting UBI Partitions

Modify the mounting position code:

```
device/hisilicon/bigfish/etc/init.rc
```

The system, userdata, and cache partitions are mounted by default:

```
mount ubifs ubi@system /system

mount ubifs ubi@userdata /data nosuid nodev
mount ubifs ubi@cache /cache nosuid nodev
```

To mount a new UBI partition, for example, systemapp (the sdcard partition is mounted to **vold** instead of **init.rc**), to **init.rc**, add the following code to **init.rc**:

```
mount ubifs ubi@systemapp /systemapp nosuid nodev
```

Precautions:

- Ensure that the partition name (*xxx*) in **ubi@***xxx* is the same as the partition name in the bootargs.
- Create a mount point (**/systemapp**) before the mounting. For details, see the method of creating a mount point for the userdata partition. Add the following code to **init.rc**:

```
mkdir /systemapp 0771 system system
```

- After the modification, compile the kernel again for the modified file to take effect.

3.1.4 Generating UBI Images

Generate UBI images for all partitions that require the UBI file system and burn the UBI images to the board. For example, to generate UBI images for the system, userdata, cache, and sdcard partitions, run the following command (for details, see **install_notes.txt**):

```
make ubifs
```

Or

```
make bigfish
```

UBI images with a file name extension of **.ubi** are generated in the **nand** of **out/target/product/Hi3798MV100** directory.

By default, UBI images are complied for the following partitions:

```
system_4k_1M.ubi
userdata_4k_1M.ubi
cache_4k_1M.ubi
sdcard_4k_1M.ubi
system_8k_2M.ubi
userdata_8k_2M.ubi
cache_8k_2M.ubi
sdcard_8k_2M.ubi


4K,8K: PageSize: Nand page size
1M,2M: BlockSize: Nand block size
```

The page size and block size of the NAND flash can be obtained by viewing the NAND flash user manuals or the information displayed when the fastboot starts. See the following example:

```
Check nand flash controller v610. found
Special NAND id table Version 1.36
Nand ID: 0x2C 0x68 0x04 0x4A 0xA9 0x00 0x00 0x00
Nand: Micron MT29F32G08CBACA
Nand(Soft-Auto): Block:1MB Page:4KB OOB:224B ECC:27bit Chip:4GB*1
```

3.1.4.1 Generating UBI Images with Varied Page Sizes and Block Sizes

Take the following precautions for generating UBI images with varied page sizes and block sizes:

- **device/hisilicon/Hi3798MV100/BoardConfig.mk** needs to be modified.
- The UBI images with the page size of 4 KB and block size of 1 MB and with the page size of 8 KB and the block size of 2 MB are compiled by default.

To generate UBI images with a page size of 2 KB and a block size of 128 KB, perform the following steps:

Step 1 Add **2k_128k** to the value of **PAGE_BLOCK_SIZE** in the **BoardConfig.mk** file.

```
# PAGE_BLOCK_SIZE will be used when make ubifs in AndroidBoard.mk
PAGE_BLOCK_SIZE := 8k_2M 4k_1M 2k_128k
```

NOTE

The underscore (_) between **2k** and **128k** is mandatory.

Step 2 Run the **make ubifs** or **make bigfish** command in the root directory to compile the UBI images.

----End

3.1.4.2 Generating a UBI Image for a New UBI Partition

Take the following precautions for generating a UBI image for a new UBI partition:

- Modify the **device/hisilicon/bigfish /build/ubi.mk** file.
- Refer to the method of generating a UBI image for the sdcard partition in the **AndroidBoard.mk** file.

Provided that the systemapp UBI partition is created, to generate a UBI image with a page size of 4 KB and a block size of 1 MB, perform the following steps:

Step 1 Add the following code: **TARGET_OUT_SYSTEMAPP:=$(PRODUCT_OUT)**/systemapp.

After the modification, the code is displayed as follows:

```
#just for rename for data to userdata
TARGET_OUT_USERDATA:=$(PRODUCT_OUT)/userdata #for systemapp
TARGET_OUT_SYSTEMAPP:=$(PRODUCT_OUT)/systemapp
```

Step 2 Add the following code to the bottom of the **define make_ubi_img** section:

```
$(hide) $(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_SYSTEMAPP) $(1)
$(2)
```

NOTE

Ensure that a semicolon (;) and a backslash (\) are added to the end of each row, except the bottom row.

After the modification, the code is displayed as follows:

```
define make_ubi_core
$(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT) $(1) $(2);\
cp -rf $(TARGET_OUT_DATA) $(TARGET_OUT_USERDATA);\
$(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_USERDATA) $(1) $(2);\
rm -rf $(TARGET_OUT_USERDATA);\
$(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_CACHE) $(1) $(2);\
$(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_SDCARD) $(1) $(2);\
$(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_SYSTEMAPP) $(1) $(2)
Endef
```

CAUTION

Verify the code in blue. If the code in blue is incorrect, the UBI image may fail to be generated.

----End

3.1.5 Burning UBI Images

CAUTION

UBI images must be generated for all partitions that require the UBI file system and burnt to the board. For example, UBI images of the default partitions (system, userdata, cache, and sdcard) and user-defined partitions must be burnt to the board.

Start the HiTool (released with the product) and load the partition list, bootargs, kernel, and UBI images. After the compilation, the partition list, bootargs, kernel, and UBI images are copied to the **out/target/product/Hi3798MV100/nand** directory.

3.2 Instance

Create the systemapp UBI partition and generate a UBI image with a page size of 2 KB and a block size of 128 KB.

NOTE

In this instance, the 500 MB systmapp partition is added to the right of the system partition. The systmapp partition needs to be mounted during system startup.

Provided that the original bootargs is as follows:

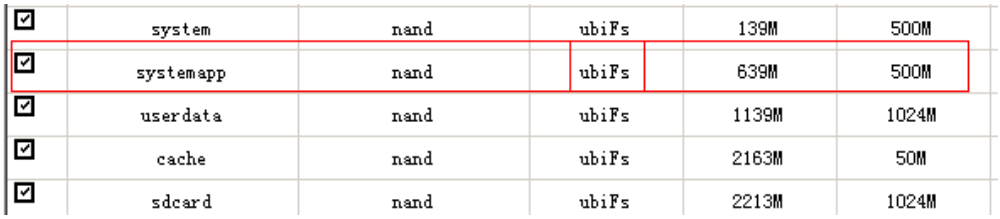
```
bootargs=mem=1G console=ttyAMA0,115200
mtdparts=hi_sfc:512k(fastboot),64k(bootargs),3456k(recovery),64k(deviceinfo);hinand:8M(baseparam),20M(logo),20M(logobak),40M(fastplay),40M(fastplaybak),40M(kernel),20M(misc),500M(system),1024M(userdata),100M(cache),2284M(sdcard) ubi.mtd=system ubi.mtd=userdata ubi.mtd=cache ubi.mtd=sdcard
```

The original bootargs contains the system, userdata, cache, and sdcard UBI partitions. After you add the systemapp partition to the bootargs, a total of five UBI partitions (system, systemapp, userdata, cache, and sdcard) are available.

To create the systemapp partition, perform the following steps:

Step 1 Add the systemapp partition to the partition list and select **ubiFs** for the systemapp partition, as shown in Figure 3-2.

Figure 3-2 Partition list instance



☑	system	nand	ubiFs	139M	500M
☑	systemapp	nand	ubiFs	639M	500M
☑	userdata	nand	ubiFs	1139M	1024M
☑	cache	nand	ubiFs	2163M	50M
☑	sdcard	nand	ubiFs	2213M	1024M

Step 2 Create the systemapp partition in the bootargs partition and add a value for **ubi.mtd**.

Based on section 3.1.2 "Modifying the bootargs", set **ubi.mtd** to **systemapp**, that is, **ubi.mtd=systemapp**.

The modified bootargs is as follows:

```
bootargs=mem=1G console=ttyAMA0,115200
mtdparts=hi_sfc:512k(fastboot),64k(bootargs),3456k(recovery),64k(deviceinfo);hinand:8M(baseparam),20M(logo),20M(logobak),40M(fastplay),40M(fastplaybak),40M(kernel),20M(misc),500M(system),200M(systemapp),1024M(userdata),100M(cache),2284M(sdcard) ubi.mtd=system ubi.mtd=systemapp ubi.mtd=userdata ubi.mtd=cache ubi.mtd=sdcard
```

Step 3 Mount the UBI file system.

Add the following code to **device/hisilicon/bigfish/etc/init.rc**:

```
mount ubifs ubi@systemapp /systemapp nosuid nodev
```

Precautions:

- Ensure that the partition name (*xxx*) in **ubi@***xxx* is the same as the partition name in the bootargs.
- Create a mount point (**/systemapp**) before the mounting. For details, see the method of creating a mount point for the userdata partition. Add the following code to **init.rc**:
  ```
  mkdir /systemapp 0771 system system
  ```
- After the modification, compile the kernel again for the modified file to take effect.

Step 4 Generate a UBI image.

Modify the **device/hisilicon/Hi3798MV100/build/ubi.mk** file.

1. Add the following code:
   ```
   ARGET_OUT_SYSTEMAPP:=$(PRODUCT_OUT)/systemapp
   ```
   After the modification, the code is displayed as follows:
   ```
   #just for rename for data to userdata
   TARGET_OUT_USERDATA:=$(PRODUCT_OUT)/userdata
   #for systemapp
   TARGET_OUT_SYSTEMAPP:=$(PRODUCT_OUT)/systemapp
   ```
2. Add the following code to the bottom of the **define make_ubi_img** section:
   ```
   $(hide) $(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_SYSTEMAPP)
   ```

```
$(1) $(2)
```

After the modification, the code is displayed as follows:

```
define make_ubi_img
$(hide) $(UBISH_DST) $(HOST_OUT_EXECUTABLES)  $(TARGET_OUT) $(1) $(2)
$(hide) cp -rf $(TARGET_OUT_DATA) $(TARGET_OUT_USERDATA)
$(hide) $(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_USERDATA)
$(1) $(2)
$(hide) rm -rf $(TARGET_OUT_USERDATA)
$(hide) $(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_CACHE) $(1)
$(2)
$(hide) $(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_SDCARD) $(1)
$(2)
$(hide) $(UBISH_DST) $(HOST_OUT_EXECUTABLES) $(TARGET_OUT_SYSTEMAPP)
$(1) $(2)
Endef
```

3. Add **2k_128k** to the value of **PAGE_BLOCK_SIZE** in the **BoardConfig.mk** file.

   ```
   PAGE_BLOCK_SIZE := 8k_2M 4k_1M 2k_128k
   ```

4. Run the **make ubifs** command in the root directory to compile the UBI image.

   ```
   make ubifs
   ```

----End

The following UBI images are generated in the **out/target/product/ Hi3798MV100/nand** directory:

```
System_2k_128k.ubi
systemapp_2k_128k.ubi
userdata_2k_128k.ubi
cache_2k_128k.ubi
sdcard_2k_128k.ubi
```

4 Recovery

4.1 Function Description

The Recovery mode is a critical boot mode in the Android system for enabling the command interpreter program (shell) and refreshing flash images. The Android system restores the factory settings and performs local upgrades or system recovery in the Recovery mode. The Recovery mode allows you to upgrade the specified partitions, such as kernel, system, and userdata, from different flashes, including the serial peripheral interface (SPI) flash, NAND flash, and eMMC.

This chapter describes how to create and customize an upgrade package for the Recovery mode.

4.2 Usage

4.2.1 Creating an Upgrade Package

On HiSTBAndroid V600R001, generate an upgrade package (**update.zip**) by running the following command in the root directory:

```
make updatezip
```

NOTE

This section describes how to create an upgrade package for Hi3798M V100. The procedure for creating an upgrade package for other chips is similar.

For details about how to create an upgrade package, see the **nand.mk and emmc.mk** file in the **device/hisilicon/bigfish/build** directory.

- **NAND_UPDATE_PACKAGE**: specifies the upgrade package for the NAND flash.
- **EMMC_UPDATE_PACKAGE:** specifies the upgrade package for the eMMC.

Creating an Upgrade Package for the NAND Flash

The following script is used to create an upgrade package for the NAND flash:

device/hisilicon/bigfish /build/nand.mk

```
NAND_UPDATE_PACKAGE :=$(NAND_PRODUCT_OUT)/update.zip
```

```
$(NAND_UPDATE_PACKAGE): $(INSTALLED_SYSTEMIMAGE)
$(INSTALLED_USERDATAIMAGE_TARGET) $(KERNEL_IMAGE) $(RECOVERY_IMAGE)
$(NAND_PREBUILT_IMG) $(NAND_HIBOOT_IMG) $(UPDATE
_TOOLS)
	@echo ----- Making update package ------
	$(hide) rm -rf $(NAND_PRODUCT_OUT)/update
	$(hide) mkdir -p $(NAND_PRODUCT_OUT)/update
	$(hide) mkdir -p $(NAND_PRODUCT_OUT)/update/file
	$(hide) mkdir -p $(NAND_PRODUCT_OUT)/update/file/META
$(hide) cp -af $(call include-path-for, recovery)/etc/recovery.fstab
$(NAND_PRODUCT_OUT)/update/file/META/recovery.fstab
	$(hide) cp -a $(PRODUCT_OUT)/bootargs_nand.txt
$(NAND_PRODUCT_OUT)/update/file/META/bootargs.txt
	$(hide) cp -a $(NAND_PRODUCT_OUT)/fastboot.bin
$(NAND_PRODUCT_OUT)/update/file/fastboot.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/recovery.img
$(NAND_PRODUCT_OUT)/update/file/recovery.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/baseparam.img
$(NAND_PRODUCT_OUT)/update/file/baseparam.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/logo.img
$(NAND_PRODUCT_OUT)/update/file/logo.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/fastplay.img
$(NAND_PRODUCT_OUT)/update/file/fastplay.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/bootargs.bin
$(NAND_PRODUCT_OUT)/update/file/bootargs.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/pq_param.bin
$(NAND_PRODUCT_OUT)/update/file/pq_param.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/kernel.img
$(NAND_PRODUCT_OUT)/update/file/boot.img
	$(hide) mkdir -p $(NAND_PRODUCT_OUT)/update/file/META-
INF/com/google/android
$(hide) cp -a $(PRODUCT_OUT)/system/bin/updater
$(NAND_PRODUCT_OUT)/update/file/META-INF/com/google/android/update-binary
	$(hide) cp -af $(PRODUCT_OUT)/system
$(NAND_PRODUCT_OUT)/update/file/system
	$(hide) cp -af $(PRODUCT_OUT)/data
$(NAND_PRODUCT_OUT)/update/file/userdata
	$(hide) cp -a $(call include-path-for, recovery)/etc/META-
INF/com/google/android/updater-script-by-file
$(NAND_PRODUCT_OUT)/update/file/META-INF/com/google/android/updater-
script
$(hide) echo
"recovery_api_version=$(RECOVERY_API_VERSION)" >$(NAND_PRODUCT_OUT)/updat
e/file/META/misc_info.txt
```

```
	$(hide) echo "fstab_version=$(RECOVERY_FSTAB_VERSION)" >>
$(NAND_PRODUCT_OUT)/update/file/META/misc_info.txt
	$(hide) (cd $(NAND_PRODUCT_OUT)/update/file && zip -
qry ../sor_update.zip .)
	zipinfo -1 $(NAND_PRODUCT_OUT)/update/sor_update.zip | awk
'/^system\// {print}' | $(HOST_OUT_EXECUTABLES)/fs_config >
$(NAND_PRODUCT_OUT)/update/file/META/filesystem_config.txt
	$(hide) (cd $(NAND_PRODUCT_OUT)/update/file && zip -
q ../sor_update.zip META/*)
	java -jar $(SIGNAPK_JAR) -w
$(DEFAULT_SYSTEM_DEV_CERTIFICATE).x509.pem
$(DEFAULT_SYSTEM_DEV_CERTIFICATE).pk8
$(NAND_PRODUCT_OUT)/update/sor_update.zip $(NAN
D_PRODUCT_OUT)/update/update.zip
	$(hide) cp -a $(NAND_PRODUCT_OUT)/update/update.zip
$(NAND_PRODUCT_OUT)/
	$(hide) rm -rf $(NAND_PRODUCT_OUT)/update
	@echo ----- Made update package: $@ --------
```

To create an upgrade package for the NAND flash, perform the following steps:

Step 1 Create the **update** directory.

```
$(hide) mkdir -p $(NAND_PRODUCT_OUT)/update
   $(hide) cp -a $(TOPDIR)out/host/linux-x86/framework/signapk.jar
$(NAND_PRODUCT_OUT)/update
```

Step 2 Copy the signature content to the **update/file** directory.

By default, fastboot, bootargs, recovery, logo, kernel, system, and data are copied to the **update/file** directory. You can delete or add the content to be copied as required.

```
$(hide) mkdir -p $(NAND_PRODUCT_OUT)/update/file
	$(hide) mkdir -p $(NAND_PRODUCT_OUT)/update/file/META
	$(hide) cp -af $(call include-path-for,
recovery)/etc/recovery.fstab
$(NAND_PRODUCT_OUT)/update/file/META/recovery.fstab
$(hide) cp -a $(PRODUCT_OUT)/bootargs_nand.txt
$(NAND_PRODUCT_OUT)/update/file/META/bootargs.txt
	$(hide) cp -a $(NAND_PRODUCT_OUT)/fastboot.bin
$(NAND_PRODUCT_OUT)/update/file/fastboot.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/recovery.img
$(NAND_PRODUCT_OUT)/update/file/recovery.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/baseparam.img
$(NAND_PRODUCT_OUT)/update/file/baseparam.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/logo.img
$(NAND_PRODUCT_OUT)/update/file/logo.img
	$(hide) cp -a $(NAND_PRODUCT_OUT)/fastplay.img
```

```
$(NAND_PRODUCT_OUT)/update/file/fastplay.img
        $(hide) cp -a $(NAND_PRODUCT_OUT)/bootargs.bin
$(NAND_PRODUCT_OUT)/update/file/bootargs.img
        $(hide) cp -a $(NAND_PRODUCT_OUT)/pq_param.bin
$(NAND_PRODUCT_OUT)/update/file/pq_param.img
        $(hide) cp -a $(NAND_PRODUCT_OUT)/kernel.img
$(NAND_PRODUCT_OUT)/update/file/boot.img
        $(hide) mkdir -p $(NAND_PRODUCT_OUT)/update/file/META-
INF/com/google/android
        $(hide) cp -a $(PRODUCT_OUT)/system/bin/updater
$(NAND_PRODUCT_OUT)/update/file/META-INF/com/google/android/update-binary
        $(hide) cp -af $(PRODUCT_OUT)/system
$(NAND_PRODUCT_OUT)/update/file/system
        $(hide) cp -af $(PRODUCT_OUT)/data
$(NAND_PRODUCT_OUT)/update/file/userdata
```

Step 3 Copy the upgrade script (**updater-script-by-file**) to the **update/file/META-INF/com/google/android/** directory and change the file name to **updater-script**.

The **updater-script** file is executed during the upgrade to control the upgrade process. To modify the **updater-script** file, see section 4.2.3 "Customizing an Upgrade Script."

```
$(hide) cp -a $(call include-path-for, recovery)/etc/META-
INF/com/google/android/updater-script-by-file
$(NAND_PRODUCT_OUT)/update/file/META-INF/com/google/android/updater-
script
```

Step 4 Access the **update/file** directory, compress all files into a .zip package, and name the .zip package **sor_update.zip**.

```
$(hide) (cd $(NAND_PRODUCT_OUT)/update/file && zip -
qry ../sor_update.zip .)
```

Step 5 Sign **sor_update.zip** to generate an upgrade package (**update.zip**).

```
java -jar $(SIGNAPK_JAR) -w  $(DEFAULT_SYSTEM_DEV_CERTIFICATE).x509.pem
$(DEFAULT_SYSTEM_DEV_CERTIFICATE).pk8
$(NAND_PRODUCT_OUT)/update/sor_update.zip $(NAN
D_PRODUCT_OUT)/update/update.zip
```

----End

Creating an Upgrade Package for the eMMC

EMMC_UPDATE_PACKAGE specifies the upgrade package for the eMMC. The upgrade package for the eMMC is different from that for the NAND flash.

- The copied signature content is different:
  - fastboot and bootargs are different.
  - kernel, recovery, system, and userdata are the same.

- The upgrade scripts are different:
  - Upgrade script for the NAND flash:
    bootable/recovery/etc/META-INF/com/google/android/updater-script-by-file
  - Upgrade script for the eMMC:
    bootable/recovery/etc/META-INF/com/google/android/updater-script-emmc

CAUTION

When a signature error occurs, the available memory capacity may be insufficient. You can specify the memory capacity.

For example, you can specify a memory capacity of 2048 MB. (**signapk.jar** uses only 1/3 of the specified memory capacity. For example, when the specified memory capacity is 2 GB, only 683 MB is used.)

java -jar –Xmx2048m signapk.jar -w testkey.x509.pem testkey.pk8 sor_update.zip update.zip

4.2.2 Customizing an Upgrade Package

You can modify the **AndroidBoard.mk** file in the **device/hisilicon/**Hi3798MV100/ directory to customize an upgrade file. The procedure for adding cache files is similar to that for adding images. To add cache files to the upgrade package, perform the following steps:

Step 1 Copy the cache folder that you want to upgrade to the **update/file** directory.

```
 $(hide) cp -af $(PRODUCT_OUT)/system
$(NAND_PRODUCT_OUT)/update/file/system
$(hide) cp -af $(PRODUCT_OUT)/data
$(NAND_PRODUCT_OUT)/update/file/userdata
$(hide) cp -af $(PRODUCT_OUT)/cache $(NAND_PRODUCT_OUT)/update/file/cache
```

Step 2 Compress the cache folder to the upgrade package. After modification, the following script is displayed in the **AndroidBoard.mk** file:

```
$(hide) cd $(NAND_PRODUCT_OUT)/update/file && zip -qry sor_update.zip
fastboot.img recovery.img baseparam.img logo.img bootargs.img boot.img
META-INF/ system userdata cache
```

----End

4.2.3 Customizing an Upgrade Script

The upgrade script is the core of an upgrade. It controls the upgrade content.

NOTE

Operation personnel shall understand the meaning of each line in the script.

- Reference upgrade script for the NAND flash:
  bootable/recovery/etc/META-INF/com/google/android/updater-script-by-file

- Reference upgrade script for the eMMC:
  bootable/recovery/etc/META-INF/com/google/android/updater-script-emmc

The upgrade script contains the following content:

- File system partitions
  - The NAND flash uses the UBI file system. For details about how to upgrade the file system partitions, see the procedure for upgrading the system partition in the reference upgrade script for the NAND flash.

    ```
    format("ubifs", "MTD", "system", "0", "/system");
    mount("ubifs", "MTD", "system", "/system");
    package_extract_dir("system", "/system");
    ```

  - The eMMC uses the EXT4 file system. For details about how to upgrade the file system partitions, see the procedure for upgrading the system partition in the reference upgrade script for the eMMC.

    ```
    format("ext4", "EMMC", "/dev/block/platform/hi_mci.1/by-
    name/system", "0", "/system");
    mount("ext4", "MTD", "/dev/block/platform/hi_mci.1/by-name/system",
    "/system");
    package_extract_dir("system", "/system");
    ```

CAUTION

- The third values of format and mount are different for the EXT4 file system and UBI file system. For details, see section 4.2.5 "Modifying Partitions During Upgrade."
- For details about how to repair the links and permission after the system partition is extracted from the upgrade package, see the updater-script-by-file script in the ootable/recovery/etc/META-IN**F/com/google/android/** directory.

- Raw partitions
  - The following sample describes how to upgrade the kernel partition for the NAND flash:

    ```
    ui_print("update boot......");
    format("raw", "MTD", "kernel", "0", "kernel");
    package_extract_file("boot.img", "/tmp/boot.img");
    write_raw_image("/tmp/boot.img", "kernel");
    ```

    In the preceding script, the first value of **format** is **raw**. However, in a UBI file system, the first value of **format** is **ubifs**.
  - The following sample describes how to upgrade the kernel partition for the eMMC:

    ```
    ui_print("update boot......");
    package_extract_file("boot.img", "/dev/block/platform/hi_mci.1/by-
    name/kernel");
    ```

CAUTION

The **format** parameter is unavailable for raw partitions on the eMMC. When the **format** parameter is executed, the EXT4 file system is configured for a raw partition.

4.2.4 Creating an OTA Upgrade Package

This section uses Hi3798M V100 as an example.

The example assumes that the **update-new.zip** and **update-old.zip** upgrade packages have been created by running **make bigfish** or make **updatezip**.

Run the following command to create an OTA upgrade package based on the **update-new.zip** and **update-old.zip** upgrade packages:

```
./device/hisilicon/bigfish/upgrade/ota/ota_from_target_files -i update-old.zip update-new.zip ota-update.zip
```

./device/hisilicon/bigfish/upgrade/ota is the directory where the script for creating the OTA upgrade package is located, and the directory can be changed.

4.2.5 Modifying Partitions During Upgrade

When the user system needs to be upgraded and the partition information is changed, you can modify the partitions during the upgrade process.

You are advised not to change the positions of the fastboot and bootargs partitions when modifying partitions.

The following takes the eMMC of Hi3798M V100 as an example. To add a 4 MB systemsign raw partition, perform the following steps:

Step 1 Change the position of the bootargs partition in fastboot in **device/hisilicon/Hi3798MV100/BoardConfig.mk**.

```
EMMC_BOOT_ENV_STARTADDR :=0x100000
```

Skip this step if the position of the bootargs partition is not changed.

Step 2 Modify **bootargs** (**device/hisilicon/Hi3798MV100/etc/bootargs_ Hi3798MV100-emmc.txt**).

```
bootcmd=mmc read 0 0x1FFFFC0 0x4D000 0x5000; bootm 0x1FFFFC0
bootargs=mem=1G console=ttyAMA0,115200
blkdevparts=mmcblk0:1M(fastboot),1M(bootargs),10M(recovery),2M(deviceinfo),8M(baseparam),8M(pqparam),20M(logo),20M(logobak),40M(fastplay),40M(fastplaybak),4M(systemsign),40M(kernel),20M(misc),500M(system),1024M(userdata),100M(cache),-(sdcard)
```

If the kernel position is changed after the bootargs partition is changed, update bootcmd.

Step 3 Modify **recovery.emmc.fstab**.

```
/systemsign   emmc   /dev/block/platform/hi_mci.1/by-name/systemsign
```

Step 4 Modify the upgrade script for partition modification in **bootable/recovery/etc/META-INF/com/google/android/updater-script-partchange-emmc**.

Modify the content in **partchange**. **/dev/block/mmcblk0** is the keyword.

Note that the partition information in **partchange** must be consistent with the information in **bootargs** described in step 2.

```
ifelse(is_mounted("/system"),unmount("/system"));
ifelse(is_mounted("/cache"),unmount("/cache"));
ifelse(is_mounted("/data"),unmount("/data"));
partchange("EMMC","/dev/block/mmcblk0:1M(fastboot),1M(bootargs),10M(recov
ery),2M(deviceinfo),8M(baseparam),8M(pqparam),20M(logo),20M(logobak),40M(
fastplay),40M(fastplaybak),4M(systemsign),40M(kernel),20M(misc),500M(syst
em),1024M(userdata),100M(cache),-(sdcard)");
setmisc("EMMC","/dev/block/platform/hi_mci.1/by-name/misc");
```

The preceding content is indispensable in the upgrade script, and you are advised not to delete it. When partition information is updated in partchange, all partitions cannot be occupied. Therefore, the system, userdata, and cache partitions need to be unmounted.

In the upgrade script, the bootargs partition must be updated, and other partitions can be updated based on the new partition information. If a partition position is changed, upgrade is required.

Step 5 Recompile the entire version by running **make bigfish –j32 2>&1 | tee log.txt**.

Step 6 Obtain the generated upgrade package after compilation.

----End

The restrictions are as follows:

- The upgrade package cannot be stored in the embedded sdcard partition. If the upgrade package is stored in the sdcard, the upgrade may fail but the original system is not damaged.
- The system cannot be powered off during partition modification, especially when the bootargs partition is upgraded.
- The memory, such as the USB flash drive, for storing the upgrade package cannot be removed during the upgrade process.
- The partchange function supports the NAND flash and eMMC flash but not the SPI flash. Therefore, if the SPI flash exists, do not change the position of partitions in the SPI flash.
- OTA is not supported by partition modification during upgrade. That is, partitions cannot be modified during OTA upgrade.
- In the upgrade script, the bootargs partition must be updated, and other partitions are updated based on the new partition information.

4.2.6 Functions in the updater-script File

The **updater-script** file can be modified as required. This section describes some functions in the **updater-script** file.

- ui_print(char *str)

  Function: Print information.

Parameter: The **str** parameter points to the address of the information to be printed.

- show_progress (char *sec, char *total)

  Function: Display the progress bar.

  Parameters:

  - **sec**: specifies the interval for updating the progress bar, generally set to **1**.
  - **total**: specifies the upgrade duration, determined by the size of the upgrade package.

- format(char *fs_type, char *partition_type, char *location,char * fs_size, char *mount_point)

  Function: Format partitions.

  Parameters:

  - **fs_type**: specifies the file system type (**ubifs** or **ext4** and **raw**).
    - NAND flash: **raw** indicates a raw partition, and **ubifs** indicates a file system partition.
    - eMMC: raw partitions are not supported, and **ext4** indicates a file system partition.
  - **partition_type**: specifies the memory type (**MTD** or **EMMC**).
  - **location**: specifies the partition name or device node corresponding to the partition
    - NAND flash: **location** is set to a partition name (**system**).
    - eMMC: **location** is set to the device node (**/dev/block/platform/hi_mci.1/by-name/system**) corresponding to the partition.
  - **fs_size**: specifies the file system size. The entire partition is erased when this parameter is set to **0**.
  - **mount_point**: specifies the mount point of the partition.

- package_extract_file(char *package_path, char *destination_path)

  Function: Extract a file from a .zip package.

  Parameters:

  - **package_path**: specifies the name of the file to be extracted from a .zip package
  - **destination_path**: specifies the directory to which the file is decompressed

- write_raw_image (char *file, char *partition)

  Function: Write a file to a partition.

  Parameters:

  - **file**: specifies the file to be written
  - **partition**: specifies the partition to which the file is written

- mount(char *fs_type, char *partition_type, char *location, char *mount_point)

  Function: Mount a specified partition to a directory.

  Parameters:

  - **fs_type**: specifies the file system type (**ubifs** or **ext4**).
  - **partition_type**: specifies the memory type (**MTD** or **EMMC**).
  - **location**: specifies the partition name or device node corresponding to the partition.
    - NAND flash: **location** is set to a partition name (**system**).
    - eMMC: **location** is set to the device node (**/dev/block/platform/hi_mci.1/by-name/system**) corresponding to the partition.
  - **mount_point**: specifies the mount point of the partition.

- unmount(char *mount_point)

  Function: Unmount a partition.

  Parameters:

  - **mount_point**: specifies the mount point of the partition.

- package_extract_dir(char *package_path, char *destination_path)

  Function: Extract a folder and decompress it to a specified directory.

  Parameters:

  - **package_path**: specifies the name of the folder to be extracted from a .zip package.
  - **destination_path**: specifies the directory to which the folder is decompressed.

- symlink(char *name, char *argv[])

  Function: Link all contents pointed by **argv*** to the **name** file.

  Parameters:

  - **name**: specifies the name of the file to which the contents are linked.
  - **argv***: specifies the file to be linked.

- set_perm_recursive (int uid, int gid, int dir_mode, int file_mode,char *path)

  Function: Modify the permission for a directory and files in the directory.

  Parameters:

  - **uid**: specifies the user ID.
  - **gid**: specifies the group ID.
  - **dir_mode**: specifies the permission for the directory.
  - **file_mode**: specifies the permission for files in the directory.
  - **path**: specifies the directory.

- partchange(char *partition_type, char *new_partition)

  Function: Create new partitions in the kernel based on the transferred partition information.

  Parameters:

  - **partition_type**: specifies the component type (**MTD** or **EMMC**).
  - **new_partition**: specifies the partition information.
  - **EMMC**: dev/block/mmcblk0
  - **MTD**: hinand

  The partchange function supports the NAND and eMMC flash but not the SPI flash.

- setmisc(char *partition_type, char *location)

  Function: Write the misc flag bit.

  - **partition_type**: specifies the component type (**MTD** or **EMMC**).
  - **location**: specifies the partition name or the device node corresponding to the partition.
    - For the NAND flash, the partition name is **misc**.
    - For the eMMC component, the device node corresponding to the partition is **/dev/block/platform/hi_mci.1/by-name/misc**.

5 NFS and SAMBA

5.1 Function Description

Functions of the NFS and SAMBA are described as follows:

- NFS stands for network file system.

  The NFS is compatible with Free Berkeley Software Distribution (FreeBSD), allowing you to share directories and files with other uses in the network and to access the files in remote systems.
- The SAMBA is a piece of free software for implementing the Server Message Block (SMB) protocol on Linux and UNIX. It consists of the server and client.

  SMB is a communication protocol for sharing resources, such as files and printers, among computers in the local area network (LAN).

The Android system accesses the network over the SAMBA or NFS, allowing you to browse shared content and play video clips in the network. In addition, the Android system serves as the SAMBA server and uploads shared directories to the network.

- As a client, the Android system automatically searches for servers in the LAN over the SAMBA or NFS in the application media center (file manager). Alternatively, you can manually mount shared directories in the network. The shared content is listed. You can add, delete, or modify the shared content based on permission, or invoke application programs based on file types, such as pictures, music files, and video clips.
- As the SAMBA server, the Android system enables or disables the local SAMBA service on the application screen, shares the board **mnt** directory, determines whether a password is required for authentication, and controls the write permission for a client. This allows the clients in the network to access local shared content and perform operations based on permission.

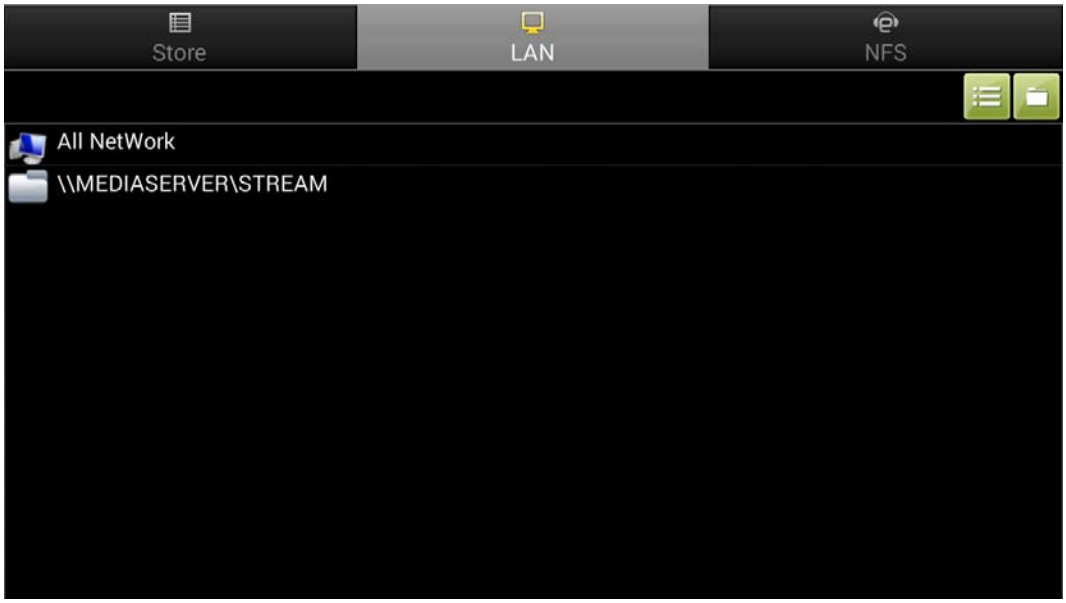
5.2 Usage

5.2.1 SAMBA Client

5.2.1.1 Automatically Searching for Servers

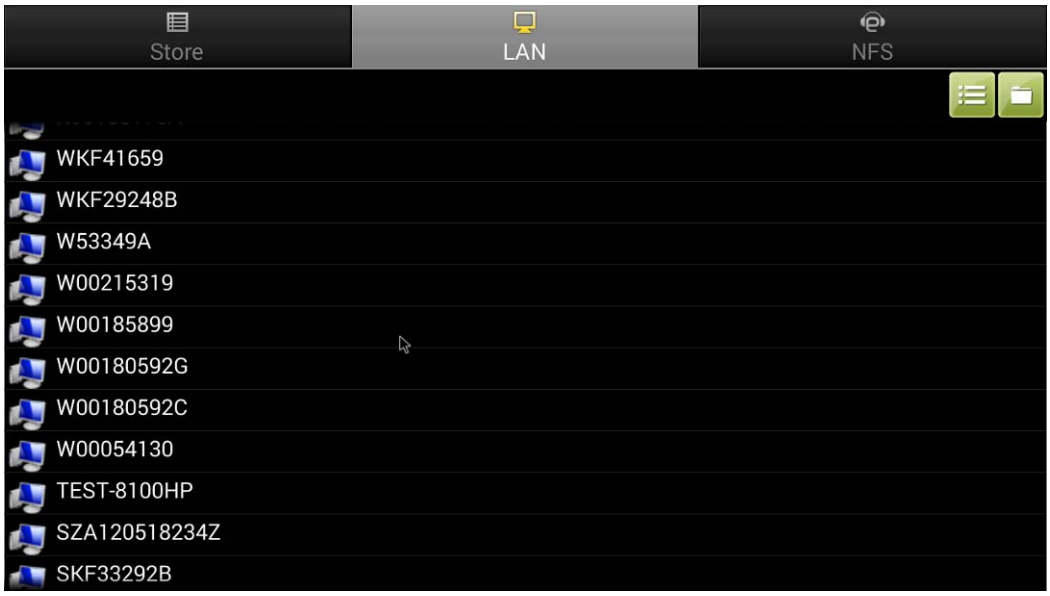
Figure 5-1 shows the SAMBA home screen.

Figure 5-1 SAMBA home screen



Step 1 Tap **All Network**. The system automatically searches for servers in the LAN. After the search is completed, search results are displayed on the screen, as shown in Figure 5-2.

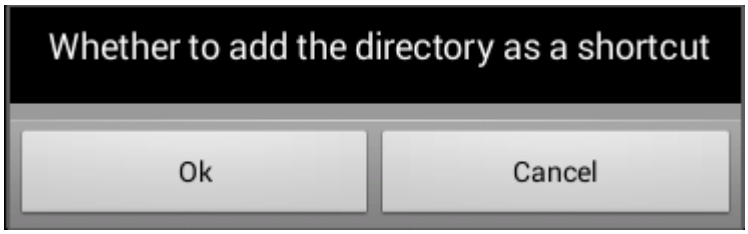
Figure 5-2 Searching for servers in the LAN



Step 2 Tap a server. If authentication is not required, the screen is refreshed, displaying shared directories on the server. If authentication is required, a dialog box is displayed, prompting you to enter the user name and password.

Step 3 Tap a shared directory. A dialog box is displayed, asking you whether to create a shortcut, as shown in Figure 5-3.

Figure 5-3 Dialog box for creating a shortcut



Step 4 Tap **OK**. A shortcut is created for the directory on the SAMBA home screen.

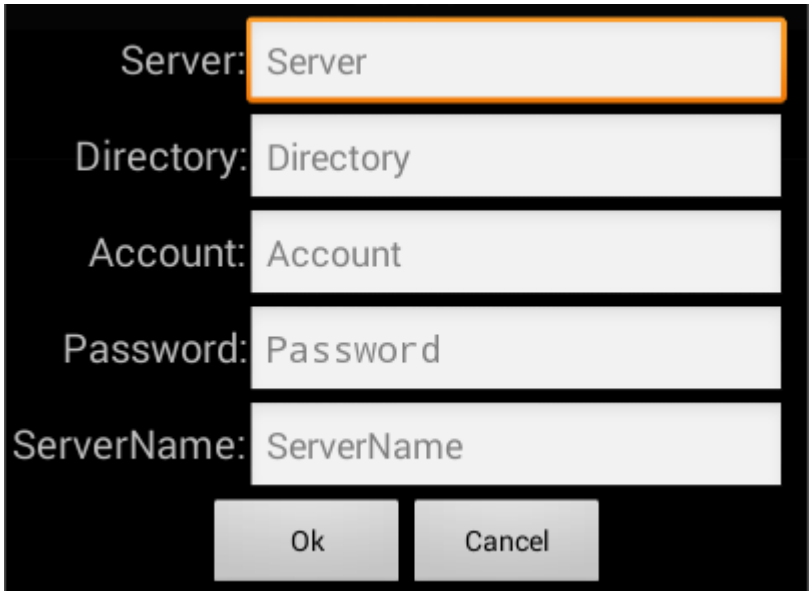
----End

5.2.1.2 Manually Mounting a Directory

To manually mount a specified directory, perform the following steps:

Step 1 Access the SAMBA home screen and press the Menu key on the remote control or press the mouse wheel. Choose **new** from the menu that is displayed. The dialog box shown in Figure 5-4 is displayed.

Figure 5-4 Manually mounting a directory



The parameters on the screen are described as follows:

- **Server**: specifies the IP address of the sharing server.

- **Directory**: specifies the shared directory (for example, **/share**) on the sharing server.
- **Account**: specifies the user name.
- **Password**: specifies the password.
- **ServerName**: specifies the server name. It is resolved by the SAMBA based on the IP address of the server.

Step 2 Tap **OK**. A shortcut is created for the directory on the SAMBA home screen.

Step 3 Tap the shortcut. Files in the directory are displayed. You can delete, modify, or add directories and open media pictures or files in the specified mode.

----End

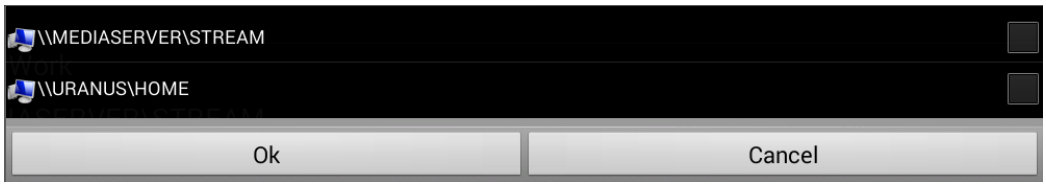
5.2.1.3 Deleting a Shortcut

To delete a shortcut from the SAMBA home screen, perform the following steps:

Step 1 Press the Menu key on the remote control or press the mouse wheel.

Step 2 Tap **Operation** and choose **Delete**. The deletion screen is displayed, as shown in Figure 5-5.

Figure 5-5 Deletion screen



Step 3 Select a shortcut that you want to delete and tap **OK**.

The shortcut is deleted from the SAMBA home screen.

----End

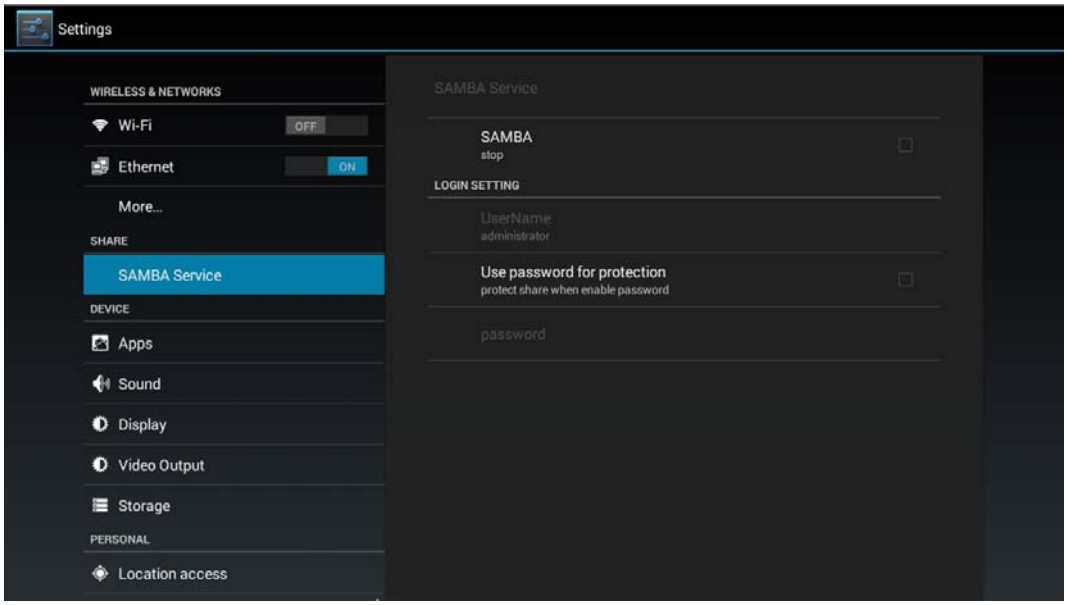
5.2.2 NFS Client

Operations on an NFS client are similar to those on a SAMBA client.

5.2.3 SAMBA Server

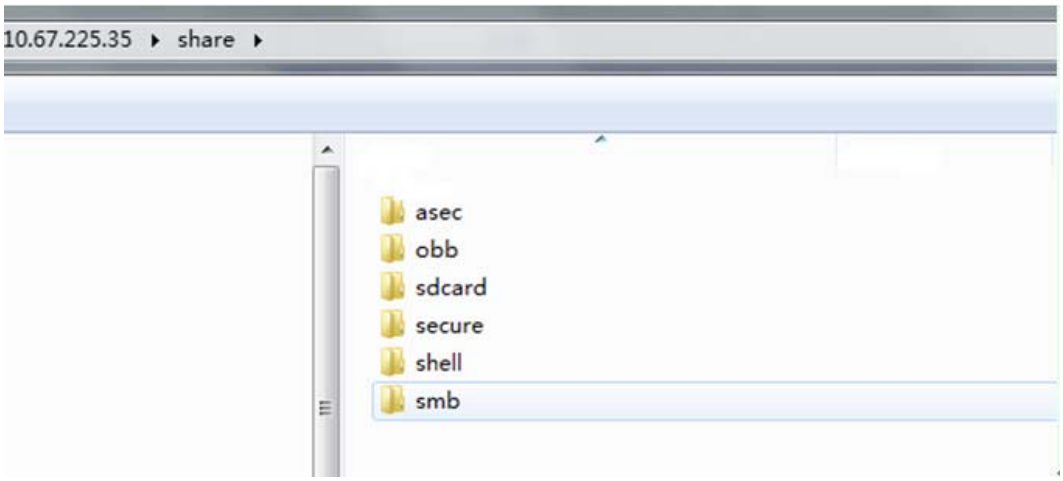
A set top box (STB) can serve as the SAMBA server, allowing the access from other devices in the LAN. Figure 5-6 shows the home screen of the SAMBA server.

Figure 5-6 SAMBA server home screen



As shown in Figure 5-6, the SAMBA service is off by default. To enable the SAMBA service, select **SAMBA**. When the "start successfully" message is displayed, the SAMBA service is enabled. You can access the STB on the computer, as shown in Figure 5-7.

Figure 5-7 Windows UI for the STB



If **Use password for protection** on the SAMBA server home screen shown in Figure 5-6 is selected, set a password, and enter the user name (**administrator**) and password when you access the STB the next time.

6 Loader

6.1 Function Description

The loader is used to download upgrade data by using the OTA/IP/USB medium and upgrade the local system, including the loader itself. For details about the loader, see the *Loader Development Guide*. For details about how to create upgrade streams, see the *HiLoader User Guide*. This chapter describes how to configure and use the loader in Android projects.

The recovery and loader programs on Android implement the upgrade function with respective restrictions. The recovery is used while the loader is disabled by default. If you enable the loader, the recovery is disabled automatically.

6.2 Usage

To start the loader, you need to modify the related configuration file. Take Hi3716C V200 as an example. The configuration file is **device/hisilicon/Hi3716CV200/BoardConfig.mk**.

```
#apploader config
APPLOADER_CFG := hi3716cdmo2b_hi3716cv200_apploader_cfg.mak
#loader ctrl macro
#false: system build with recovery
#true: system build with loader
TARGET_HAVE_APPLOADER := false
```

Where:

- **TARGET_HAVE_APPLOADER** indicates whether to enable the Apploader. If it is **true**, the loader is enabled.
- **APPLOADER_CFG** specifies the dedicated configuration file of the Apploader.

CAUTION

- The preceding configuration options are supported in HiSTBAndroid V500R001C00CP0005 or later.

- Besides the dedicated configuration file specified by **APPLOADER_CFG**, the Apploader also requires that the configuration items **CFG_HI_LOADER_SUPPORT** and **CFG_HI_APPLOADER_SUPPORT** in the Android dependent basic configuration file (for example, **device/hisilicon/bigfish/sdk/configs/hi3716cdmo2b_hi3716cv200_android_cfg.mak**) be enabled. This is automatically implemented by the Android architecture.
- Ensure that the SUID is set correctly by running **mknod**. Run **chmod a+s 'which mknod'** as the root user to add the SUID. Information similar to the following is displayed after the SUID is configured successfully:

```
-rwsr-sr-x 1 root root 43488 2010-09-22 02:32 /bin/mknod
```

After the Android project is created by running **make bigfish**, files related to the loader are generated in the **out/target/product/Hi3716CV200** directory.

```
Emmc
├── bootargs_Hi3716CV200-emmc-loader.bin
├── fastboot-burn-emmc.bin
├── apploader.bin
├── system.ext4
Nand
├── bootargs_Hi3716CV200-nand-loader.bin
├── fastboot-burn-nand.bin
├── apploader.bin
├── system.ubi
```

Where:

- In **bootargs_*-loader.bin**, five loader dependent partitions (stbinfo, loaderdb, loaderdbbak, loader, and loaderbak) are added to the bootargs partition table. For details about the usage of these partitions, see section 1.6 in the *Loader Development Guide*.
- In **fastboot-burn-*.bin**, the upgrade flag and the function of booting the Apploader image are added.
- **apploader.bin** is the core image of the loader and needs to be burnt to the loader and loaderbak partitions.
- **system.*** contains the sample_loader tool for simulating loader upgrade triggering and the dependent **libhiloader.so** library.

7 Standby

NOTE

This chapter takes Hi3798M V100 as an example. The configurations are similar for other chips.

7.1 Function Description

The current system supports four wakeup modes:

- The system restarts after wakeup (default mode).
- The Android Launcher UI is displayed after wakeup.
- The Application UI before standby is displayed after wakeup(Android native standby mode).
- The screen before the system enters the standby mode is displayed after wakeup (light standby mode).

The wakeup mode is configured by the system property which is defined in device/hisilicon/Hi37XXXVXXX/customer.mk.

```
# smart_suspend, deep_launcher, deep_restart, deep_resume;
PRODUCT_PROPERTY_OVERRIDES += \
    persist.suspend.mode=deep_restart
```

This property has four optional configurations:

- deep_restart: The system restarts after wakeup (default mode)
- deep_launcher: The Android Launcher UI is displayed after wakeup
- deep_resume: The Application UI before standby is displayed after wakeup(Android native standby mode)
- smart_suspend: The screen before the system enters the standby mode is displayed after wakeup (light standby mode)

Please choose the wakeup mode according to actual requirements.

Except the Android native standby mode, other three wakeup modes are designed by HiSilicon, and they are different from the native one.

By using the broadcasting mechanism and wakelock mechanism, the Android system ensures that the applications and services complete related data processing and save or restore the running status before standby or after wakeup.

The HiSilicon Android standby solution is designed based on the television scenario. In this scenario, you need only to turn off the screen to enable the television to enter the low-power state so that it can be started quickly, and you do not need to enable the television to frequently enter and exit the standby mode as you do in the mobile phone scenario.

The following describes the core of the HiSilicon Android standby solution:

- The inherent standby lock is masked, and a special dedicated lock is used.
- Core services are created and the special lock is obtained before standby. All non-core Java processes are stopped and then the system enters the standby mode. After wakeup, the launcher UI is displayed.

If you do not want to stop the created services during standby, add tags in the .xml file (see section 7.3.1 "Preventing Background Services from Being Stopped").

CAUTION

In the HiSilicon solution, all non-core Java processes are stopped after the system enter the standby mode. Therefore, the standby lock cannot be used for created applications if they are not core services (services that are not stopped in standby mode).

In a word, do not use the standby lock or use it with caution.

7.2 Usage

7.2.1 Dedicated Standby Lock

In the Android inherent standby solution, after the standby broadcasting message is received, applications or services apply for the PARTIAL_WAKE_LOCK lock by using the power management service PowerManagerService to suspend the standby process. After some necessary clear and save operations, the applications or services release the lock and the system enters the standby mode.

PARTIAL_WAKE_LOCK is defined as follows in **PowerManager.java**:

```
public static final int PARTIAL_WAKE_LOCK = 0x00000001;
```

In the HiSilicon Android standby solution, there is no response if applications apply for the PARTIAL_WAKE_LOCK lock in powerManagerService. The new standby lock SUSPEND_WAKE_LOCK is created. This new lock implements the functions of the inherent lock but is not owned by the inherent programs. You need to use this lock with caution.

SUSPEND_WAKE_LOCK is defined as follows in **PowerManager.java**:

```
public static final int SUSPEND_WAKE_LOCK = 0x00001000;
```

7.2.2 Core Service HiRMService

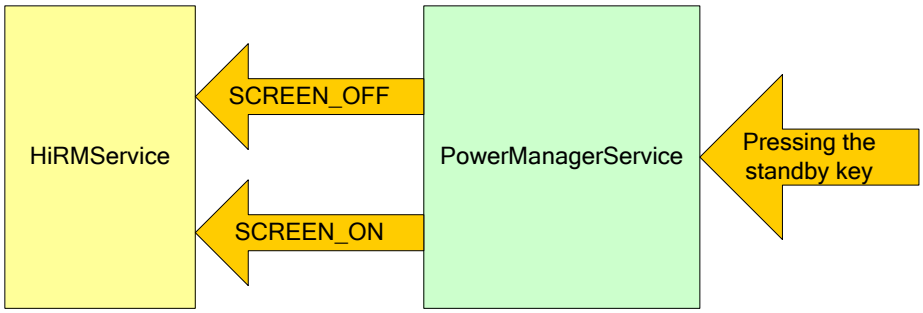
The inherent Android system requires that applications hold the standby lock before standby to clear or maintain application status so that applications run properly after wakeup. In the HiSilicon Android standby solution, because the launcher UI is displayed after system wakeup, some clear operations need to be performed before standby. HiRMService is used to implement this function.

HiRMService is defined as an application-level core service which starts after the system boots and resides in the system. This service listens to standby and wakeup broadcasting messages. After receiving the standby broadcasting message, this service applies for the standby lock, stops non-core services, disables services such as the network and Bluetooth, and then releases the standby lock. After receiving the wakeup broadcasting message, it restores services such as the network and Bluetooth.

After you press the standby key, the system responds to the standby key and sends the standby broadcasting message (SCREEN_OFF) or wakeup broadcasting message (SCREEN_ON) in PowerManagerService. In the inherent system, applications apply for the standby lock if necessary after they receive the standby broadcasting message, and release the lock after clearing is complete. Then the system enters the standby mode. In the HiSilicon solution, HiRMService listens to the standby and wakeup broadcasting messages of PowerManagerService, then applies for and releases new valid standby locks to complete the clearing operations before the system enters the standby mode, or restores services such as the network and Bluetooth during wakeup.

Figure 7-1 shows the simplified interaction between HiRMService and PowerManagerService.

Figure 7-1 Interaction between HiRMService and PowerManagerService



7.2.3 Light Standby

CAUTION

The light standby is not real standby, and the standby power is not significantly decreased. The light standby mode requires synchronous modifications of the middleware and the development application. You are advised not to use this standby mode.

You can press the standby button to enter the standby mode or wake up the system from the standby mode. After entering the light standby mode, the system only disables the audio and video outputs. No other operations are performed.

- After entering the light standby mode, the system gives a light standby broadcast: **Action="smart_suspend_broadcast_enter"**. After audio and video outputs are disabled, the system gives another broadcast: **Action=Intent.ACTION_SCREEN_OFF**.
- After being woken up from the light standby mode, the system gives the broadcast: **Action="smart_suspend_broadcast_enter"**. After audio and video outputs are enabled, the system gives another broadcast: **Action=Intent.ACTION_SCREEN_ON**.

The middleware or third-party applications can handle their own services if necessary after receiving the preceding broadcasts.

7.2.4 Configuration of the Wakeup Key

The code is stored in **device/hisilicon/bigfish/system/ir_user/ir_user.c**.

In Android 4.4, the key value corresponding to KEY_POWER in **device/hisilicon/bigfish/system/ir_user/key_pars/key.xml** is set for the C51 driver by default as the wakeup key by using the PowerResumSet function in **ir_user.c**.

7.3 Instances

7.3.1 Preventing Background Services from Being Stopped

In the **manifest** file of applications or service, add **android:persistent="true"** in the **application** tag.

```
<application android:persistent="true" android:label="@string/app_name">


       <receiver android:name=".HiRMBroadcastReceiver">
           <intent-filter>
              <action
android:name="android.intent.action.BOOT_COMPLETED"/>
          </intent-filter>
        </receiver>


        <service android:name=".HiRMService"/>


</application>
```

7.3.2 Obtaining the New Standby Lock

The new standby lock is created to deactivate the inherent standby lock and is dedicated for HiRMService. It can be ignored. If you need to apply for the lock for your own services, ensure that the services are not stopped during standby. For details about how to prevent services from being stopped, see section 7.3.2 "Obtaining the New Standby Lock." Note that you need to use the standby with caution.

The method for obtaining the new standby lock is the same as that for obtaining the inherent lock. You need only to replace the inherent lock with the new lock in the definition. See the following instance:

```
wakeLock=pm.newWakeLock(PowerManager.SUSPEND_WAKE_LOCK,this.getClass().ge
```

```
tCanonicalName());
wakeLock.acquire();
wakeLock.release();
```

8 3G Card

8.1 Overview

CAUTION

Currently only the dual-core Android 4.4 version HiSTBAndroid V500R001C01CP0003 supports the 3G card by default.

The 3G card enables the Android STB to access the mobile network, extending the STB network access capability. The 3G card provides two access modes:

- Automatic mode

  In this mode, you do not need to specify any parameter. The parameters are automatically obtained based on the SIM card for accessing the network. The operation is simple.
- Manual mode

  In this mode, you need to specify parameters for accessing the networking, including the PIN code, dialing code, access point, user name, and password. This mode can be used when the automatic mode does not work. This mode is more complex. It provides four access modes including China Mobile, China Unicom, China Telecom, and customization. For the first three access modes, default parameters for the specific operator are defined to facilitate operations. For the last access mode, you can customize the parameters.

8.2 Usage

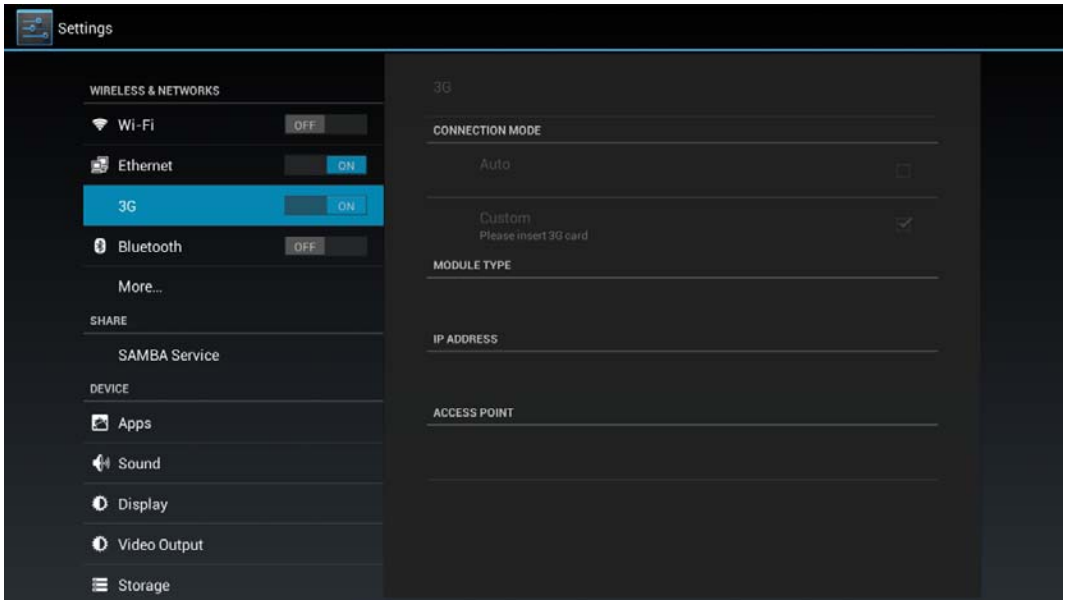
CAUTION

Before using the 3G card, you need to activate the card. For details about the charge, consult the local operator.

8.2.1 Automatic Mode

Figure 8-1 shows the connection mode configuration of the 3G card.

Figure 8-1 Connection mode configuration



To use the automatic mode, perform the following steps:

Step 1 Choose **Settings** > **3G**, insert the 3G dongle device, and set **3G** to **ON**.

Step 2 Select **Auto**. If **Auto** has been selected before, the network is connected automatically when **3G** is set to **ON**. During the connection process, the **ON** switch, **Auto**, and **Custom** options are unavailable.

After the network is successfully connected, the controls are available and the IP address is displayed. The 3G dongle supports the USB 2.0 and USB 3.0 ports. After the connection mode is configured and enabled, the previous configuration is restored each time the STB is restarted.

----End

8.2.2 Manual Mode

To use the manual mode, perform the following steps:

Step 1 Choose **Settings** > **3G**, insert the 3G dongle device, and set **3G** to **ON**.

Step 2 Select **Custom**. **ACCESS POINT** is now available. If the access point is China Unicom, China Mobile, or China Telecom, the system starts dialing. If dialing is successful, the IP address is displayed; otherwise, information is displayed indicating a connection failure. The selected access point must match the SIM card. For example, for the China Unicom SIM card, you need to select a China Unicom access point.

Step 3 If you select the customization mode, the parameters for the PIN code, dialing code, access point, user name, and password are displayed. Click any parameter (PIN code, dialing code, user name, or password), specify the parameters in the displayed dialog box, and click **OK**.

The system starts to connect to the network. If all the parameter values are correct, the connection will be successful. If you are not sure about the PIN code and dialing code, consult the operator.

When the network is connected, you can switch the connection mode from automatic mode to manual mode, or change the access point. During the switchover, the current connection is disabled and a new connection is created.

----End

8.3 GUI

- The 3G GUI can display information about the 3G dongle vendor. After the 3G card is inserted, the device information is automatically read and displayed, and the information is cleared after the 3G card is removed.
- The IP address can be obtained only after the network is successfully connected, and it is cleared when the network is disconnected.
- In manual mode, the parameter list is displayed only when you select the customized access point. When the customized access point is switched to other access point, the parameter list is automatically hidden.
- When **3G** is set to **OFF**, all controls are dimmed and unavailable. When the 3G card is removed, the controls are also dimmed.

8.4 FAQs

8.4.1 What Do I Do If Data Streams Cannot Be Transmitted over the 3G Card After the Network Cable Is Inserted?

Problem Description

When the 3G card is used to connect to the network, if the network cable is inserted and also connected to the network, data streams cannot be transmitted by using the 3G card.

Cause Analysis

This issue occurs because the Ethernet takes priority over the 3G network. If the Ethernet is connected, data streams are transmitted over the Ethernet instead of the 3G network.

Solution

Remove the network cable and reconnect the 3G card.

8.4.2 How Do I Change the Access Point?

Problem Description

How do I change the access point?

Solution

When the 3G card is used for the first time, if you choose the manual mode, the customized access point is used by default. After you change the access point, the configuration is stored into the database and loaded when the STB is started next time.

9 Quick Boot

9.1 Overview

9.1.1 Introduction

Quick boot is a feature that accelerates the boot of the Android system. The memory image is created for the Android system and then restored during each boot to significantly reduce the boot time.

When integrating the quick boot solution, you need to pay attention to the quick boot adaptation, compilation version, and mass production details.

9.1.2 Requirements

Quick boot is supported when the requirements described in Table 9-1 are met.

Table 9-1 Requirements of quick boot

Item	Requirement
Chip	Hi3798M V100
Flash	eMMC
Android version	Android 4.4.2 user mode
Memory	1 GB
Logo/Fastpaly	Logo

9.2 Instructions

To use the quick boot function, perform the following steps:

Step 1 Enable quick boot compilation.

Change **BOARD_QBSUPPORT: = false** to **BOARD_QBSUPPORT: = true** in **device/hisilicon/Hi3798MV100/customer.mk**.

If the file system partitions with read and write properties (such as data and cache) exist, for example, the fc partition, you need to add the unmount and mount operations in **device/hisilicon/bigfish/etc/andsnap**.

```
umount /fc
stop keystore
stop  hidns-sd
/sbin/snapshot -i3$*
start keystore
start  hidns-sd
mount -o nosuid,nodev -t ext4 /dev/block/platform/hi_mci.1/by-name/fc /fc
```

Step 2 Compile the version.

```
source build/envsetup.sh
lunch Hi3798MV100-user
make bigfish -j16 2>&1 | tee log.txt
```

CAUTION

You must compile the user version because the eng version does not support quick boot.

Step 3 Burn images after the version is compiled.

NOTE

- If the quick boot function is used, two additional images **userapi** and **hibdrv** need to be burnt.
- The quick boot compilation option is disabled (**BOARD_QBSUPPORT: = false**) in the SDK.
- After enabling the quick boot compilation option, clear up the last compilation result and recompile the version.

----End

9.3 Notes

9.3.1 Mass Production

The indicator is used to show the state when images are being burnt to the board.

- Blinking red and green: Images are being burnt or the system boots normally for the first time.
- Red: The memory image is being created.
- Steady green: The boot is complete.

The entire process takes about three minutes. During factory production, do not turn off the power if the indicator is not steady green to avoid exception.

NOTE

You need to modify the configuration as required based on the LED.

9.3.2 Upgrade

When compiling the Android system upgrade script, ensure that the following two flags in the **data** directory are removed:

- /data/property/persist.sys.firstboot.flag
- /data/property/persist.sys.qb.flag

Sample code:

```
bootable/recovery/etc/META-INF/com/google/android/updater-script-emmc
mount("ext4", "EMMC", "/dev/block/platform/hi_mci.1/by-name/userdata",
"/data");
package_extract_dir("userdata", "/data");
delete("/data/property/persist.sys.firstboot.flag");
delete("/data/property/persist.sys.qb.flag");
unmount("/data");
```

To replace the quick boot version with the non-quick boot version, or replace the non-quick boot version with the quick boot version, perform the upgrade by using the entire package but not the OTA mode. That is, format the data partition in the upgrade script.

Sample code:

```
bootable/recovery/etc/META-INF/com/google/android/updater-script-emmc
format("ext4", "EMMC", "/dev/block/platform/hi_mci.1/by-name/userdata",
"0", "/data");
mount("ext4", "EMMC", "/dev/block/platform/hi_mci.1/by-name/userdata",
"/data");
package_extract_dir("userdata", "/data");
delete("/data/property/persist.sys.firstboot.flag");
delete("/data/property/persist.sys.qb.flag");
unmount("/data");
```

9.3.3 Boot Time

The boot time is affected by the factors described in Table 9-2.

Table 9-2 Factors that affect the boot time

Factor	Influence	Recommendation	Prolonged Boot Time
eMMC	The quick boot time varies according to the eMMC model.	Use the eMMC listed in the compatibility list.	Determined by the eMMC model
Setting the default launcher	When there are multiple launchers, reloading the default launcher prolongs the quick boot time. The launcher configuration is not saved in the current version.	Pre-configure a self-developed launcher.	0.5−1 second
Setting the	Reloading the configured language and font	Set the system language	About 1 second

Factor	Influence	Recommendation	Prolonged Boot Time
language and font size	size prolongs the quick boot time. The last configured system language and font size are not saved in the current version.	to English and font size to large by default.	
Setting the wallpaper	Reloading the configured wallpaper prolongs the quick boot time. The last configured wallpaper is not saved in the current version.	-	About 0.5 second
Adding a partition	Adding a new partition with read and write properties prolongs the quick boot time.	-	About 0.5 second
Burning the fastplay partition	The quick boot feature does not support fastplay. If the fastplay partition is burnt, the quick boot time is prolonged.	You are advised not to burn the fastplay partition.	2 seconds

NOTE

Setting the default launcher and saving the last language, font size, and wallpaper affect the quick boot time. Therefore, these functions are removed for the quick boot feature.

9.4 API

The quick boot feature provides an API to choose the quick boot mode. This API is unavailable for the non-quick boot version (when **BOARD_QBSUPPORT** is **false**)

To use the quick boot API, perform the following steps:

Step 1 Import the following package.

```
import android.app.QbAndroidManager;
```

Step 2 Define the variables.

```
QbAndroidManager mQbAndroidManager;
private QbAndroidManager getQbAndroidManager()
{
if (mQbAndroidManager == null)
{
mQbAndroidManager=(QbAndroidManager)mContext.getSystemService(Context.QB_
ANDROID_MANAGER_SERVICE);
if (mQbAndroidManager == null)
{
Log.e("Qb", "mQbAndroidManager is Null.");
}
}
return mQbAndroidManager;
}
```

Step 3 Disable the quick boot mode by calling mQbAndroidManager.setQbEnable (false).

NOTE

Calling this API restarts the system. This API is invalid in non-quick boot mode.

Step 4 Enable the quick boot mode by calling mQbAndroidManager.setQbEnable (true).

NOTE

Calling this API restarts the system. Calling this API in quick boot mode automatically re-creates the image.

The sample source code is in **\device\hisilicon\bigfish\development\apps\HiFastboot**.

----End

10 HiDLNA

10.1 Function Description

10.1.1 DLNA Overview

The Digital Living Network Alliance (DLNA) is an organization established by Sony, Intel, and Microsoft. It aims to implement interconnection between wired and wireless networks and multimedia devices including PCs, mobile devices, and electrical appliances, enabling unlimited sharing and development of digital media and content services. Its objective is to enjoy your music, photos, and videos anywhere anytime. It now has over 280 member companies.

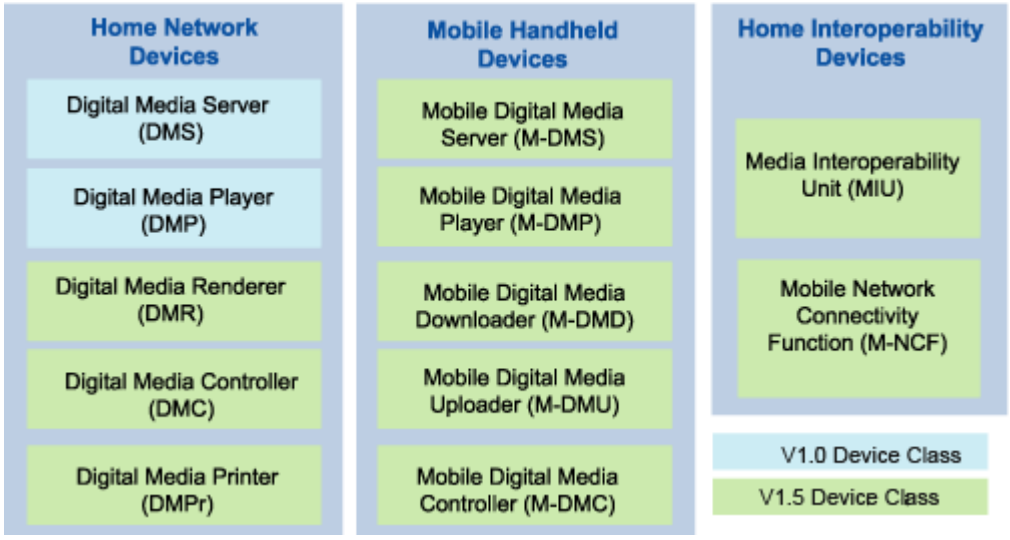
The DLNA does not create technologies. It formulates solutions or guidelines that the world follows. Therefore, the technologies and protocols upon which these guidelines are built are those widely used currently.

The DLNA divides its application into the following five functional components (from top to bottom):

- Network interconnection
- Network protocol
- Media transmission
- Device control and management
- Media format

Figure 10-1 shows the device types defined by the DLNA.

Figure 10-1 DLNA device types



10.1.2 HiDLNA Functions

The HiDLNA is one of the multi-screen components, which consists of the following three device types defined by the DLNA:

- DMS

  The digital media server (DMS) implements storage and sharing of media resources, providing access to stored media contents for the connected digital media players (DMPs) and digital media renderers (DMRs).

- DMP

  The DMP finds and obtains contents on the DMS or mobile DMS (M-DMS) and provides playback, rendering, and playback control functions.

- DMR

  The DMR plays contents obtained from the DMS or M-DMS through the configuration of other devices. The DMR is different from the DMP as the DMR requires the digital media controller (DMC) to obtain contents on the DMS.

10.2 Usage

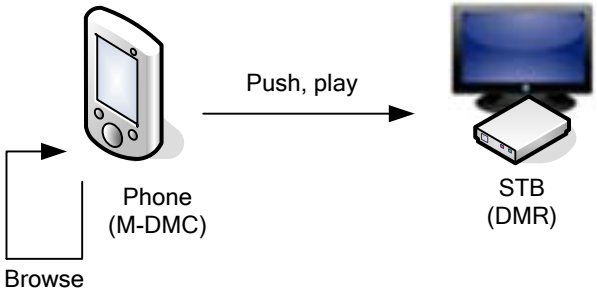
10.2.1 HiDLNA Application Scenarios

10.2.1.1 DMR Application Scenarios

DMR 2-Box Push Model

When you push photos, videos, or music on a mobile phone to an STB DMR for playing, as shown in Figure 10-2, two devices are involved in the interoperability. The DLNA defines this kind of scenario as the 2-Box Push model.

Figure 10-2 DMR 2-Box Push model

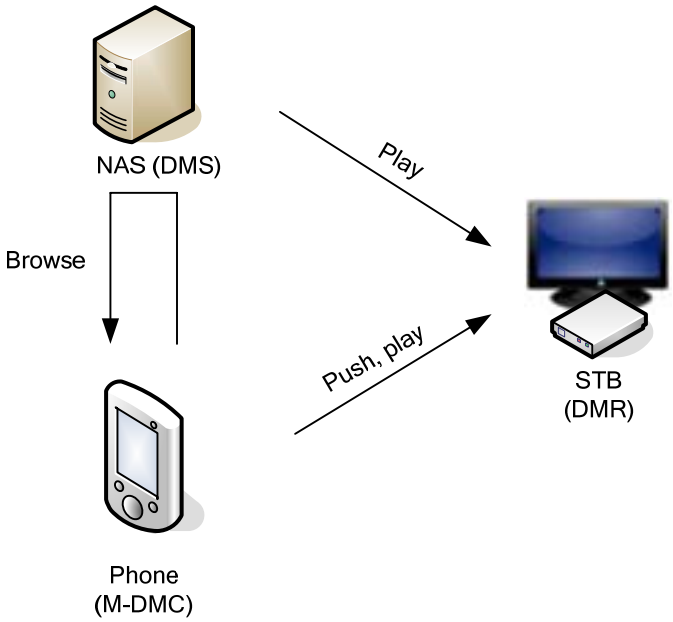


Recommended DMC clients: aVia for Android and Windows Media Player for Windows 7 can push local pictures, videos, and music to DMR devices for playing.

DMR 3-Box Model

When you use a mobile phone to access media resources on a DMS device (or an online media) and push the media resources on the DMS device to an STB DMR for playing, as shown in Figure 10-3, three devices are involved in the interoperability. The DLNA defines this kind of scenario as the 3-Box model.

Figure 10-3 DMR 3-Box model

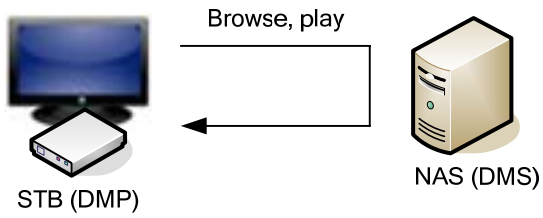


Recommended DMC clients: Twonky Mobile, aVia, Sohu Video, and Tencent Video for Android; 8Player for iOS; Windows Media Player for Windows 7

10.2.1.2 DMP Application Scenario

An STB directly accesses and plays media resources on a DMS device as a DMP device, as shown in Figure 10-4.

Figure 10-4 DMP application scenario



In this scenario, you can open the DMP application on an STB, find a media server on the network, and browse and play media files on the media server (DMS).

The frequently used DMS devices include PCs and network storage devices (NAS).

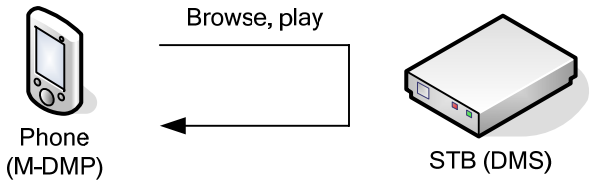
Recommended DMS client: Windows Media Player for Windows 7

10.2.1.3 DMS Application Scenarios

DMS 2-Box Push Model

When you use a DMP device, such as a mobile phone, to directly access and play shared media files on an STB DMS, as shown in Figure 10-5, two devices are involved in the interoperability. The DLNA defines this kind of scenario as the 2-Box Pull model.

Figure 10-5 DMS 2-Box Push model



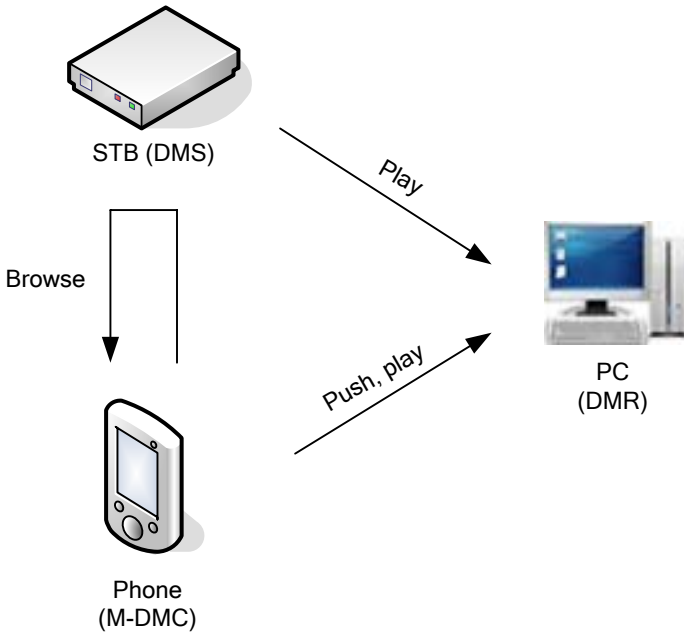
The frequently used DMP devices include intelligent TV sets, PCs, intelligent mobile phones, and game machines, which can directly access and play shared media files on an STB DMS.

Recommended DMP clients: Twonky Mobile and aVia for Android; 8Player for iOS; Windows Media Player for Windows 7.

DMS 3-Box Model

When you use a mobile phone with a DMC client to access an STB DMS, and push the shared media resources on the STB DMS to a PC or TV set that supports the DMR function for playing, as shown in Figure 10-6, three devices are involved in the interoperability. The DLNA defines this kind of scenario as the 3-Box model.

Figure 10-6 DMS 3-Box model



Recommended DMC clients: Twonky Mobile and aVia for Android; 8Player for iOS; Windows Media Player for Windows 7.

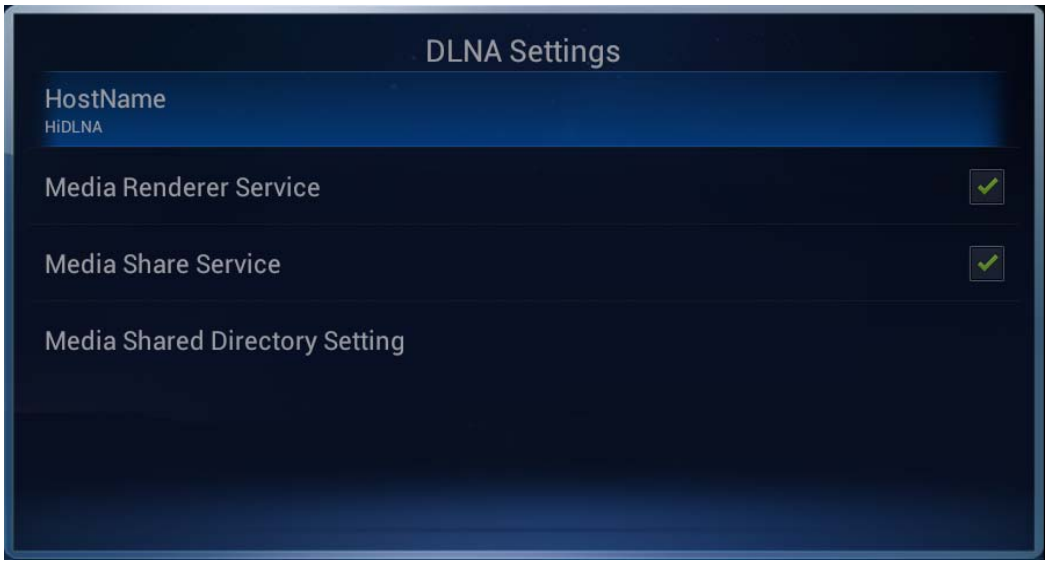
Recommended DMR client: Windows Media Player for Windows 7

10.2.2 HiDLNA Usage Guide

10.2.2.1 HiDLNA Configuration

Open **My App**, select **HiDLNA Settings**, and press **OK** on the remote control. The **DLNA Settings** screen is displayed, as shown in Figure 10-7.

Figure 10-7 HiDLNA settings



As shown in Figure 10-7, the following settings are included:

- **HostName**

  Modifies the displayed DMS and DMR name.

- **Media Renderer Service**

  Enables or disables the DMR service. You can select this option to enable the DMR service or deselect this option to disable the DMR service.

- **Media Share Service**

  Enables or disables the DMS service. You can select this option to enable the DMS service or deselect this option to disable the DMS service.

- **Media Shared Directory Setting**

  Specifies the shared directory on the DMS. You can share media resources in a specified directory by using the DMS service.

DMR Configuration

The DMR service is used to push media resources. When the DMR service is enabled, media resources can be pushed to STBs for playing through other DMC devices.

To start the DMR service, select **Media Renderer Service** on the **DLNA Settings** screen, as shown in Figure 10-7.

DMS Configuration

The DMS is used to share media resources for other devices to access and play.

To configure the DMS, perform the following steps:

Step 1 Set the media shared directory.

1. Select **Media Shared Directory Setting** on the **DLNA Settings** screen, and press **OK** on the remote control.
2. Select **Add a directory,** and press **OK** on the remote control.
3. Select the directory to be shared, and press **OK** on the remote control. The directory is added to the shared directory list.

Step 2 Enable the DMS service.

Select **Media Share Service** on the **DLNA Settings** screen, and press **OK** on the remote control to enable the DMS service. The DMS then scans media resources in the shared directories. After that, you can use other devices to access and play the media resources on the DMS.

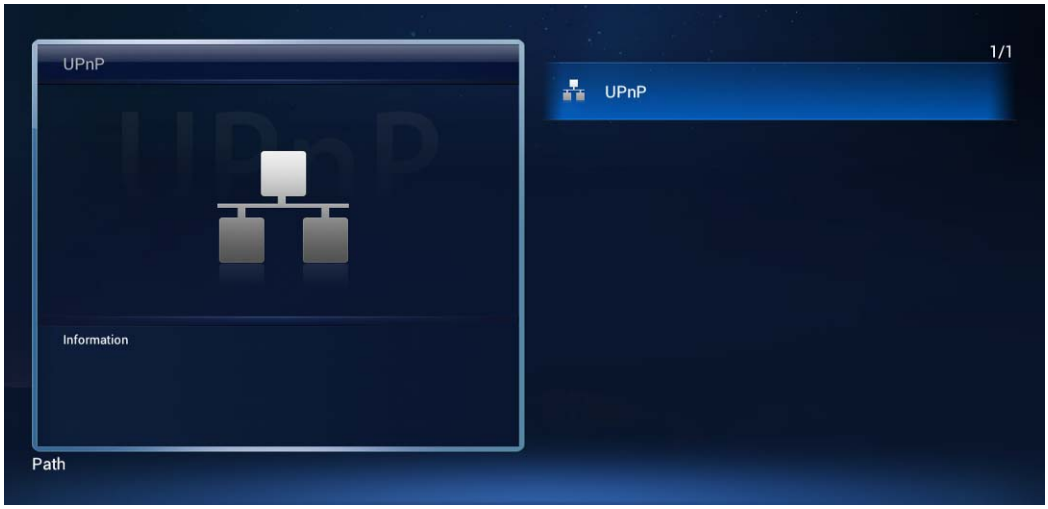
----End

10.2.2.2 DMP Usage

The DMP finds and obtains shared contents on the DMS or M-DMS and provides the media playback function.

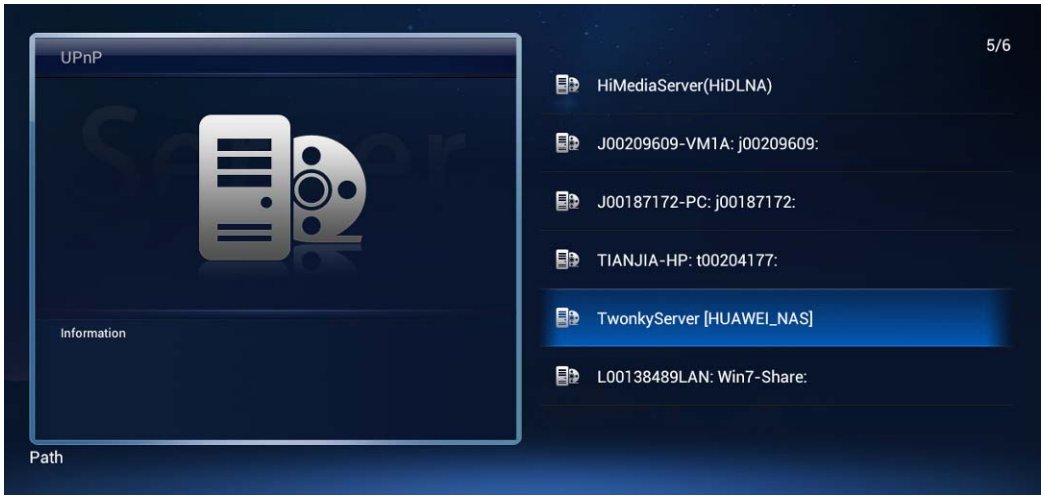
Step 1 Open **My App**, select **HiMediaCenter**, and press **OK** on the remote control. The DMP application screen is displayed, as shown in Figure 10-8.

Figure 10-8 DMP application screen



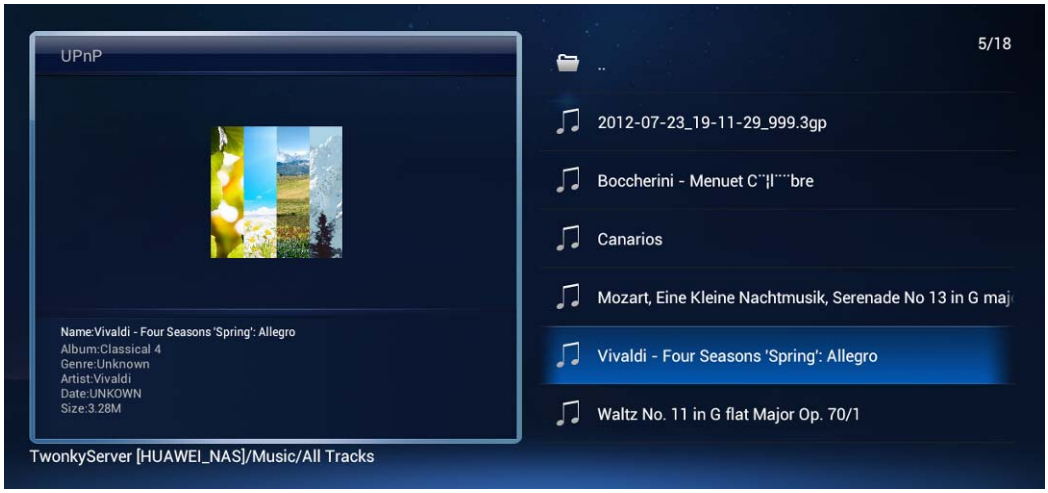
Step 2 Select **UPnP**, and press **OK** on the remote control. The list of all DMSs in the LAN is displayed, as shown in Figure 10-9.

Figure 10-9 DMS list



Step 3 Select the DMS to be accessed, and press **OK** on the remote control to go to the shared directory. You can move the cursor over a media file for a while, and the preview information of the file is displayed, as shown in Figure 10-10.

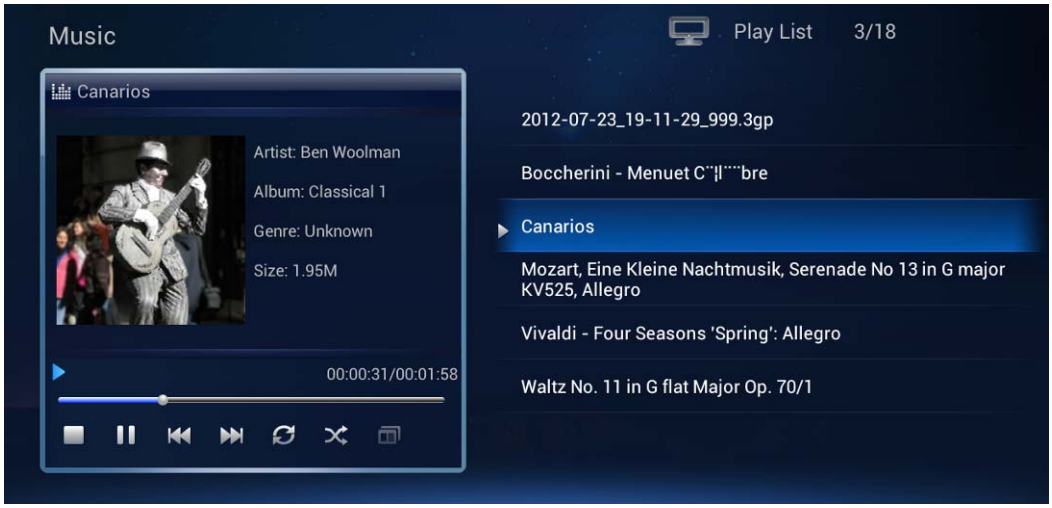
Figure 10-10 Resource preview screen



Step 4 Select the media file to be played, and press **OK** on the remote control. You can switch to the full-screen mode or perform some play-related operations.

For example, if you play a music file, you can perform operations such as play, pause, play the previous one, play the next one, or play in loop mode or random mode, as shown in Figure 10-11.

Figure 10-11 Music playback screen



Step 5 Press **Back** on the remote control to return to the DMP screen. You can browse, select, preview, or play other media files.

----End

10.2.3 Recommended DLNA Clients

10.2.3.1 DLNA Clients for Mobile Phones

Android

Figure 10-12 shows the recommended Android mobile phone clients.

Figure 10-12 Android mobile phone clients

	
Twonkey Mobile	aVia
Version: 2.3.2	Version: 4.0.18634
M-DMP/M-DMC/M-DMS	M-DMP/M-DMC

- Twonky Mobile:

  This client has comprehensive functions. It can be used to perform some basic play control and volume adjusting operations on the STB after media resources are pushed.
- aVia:

  This client has a neat GUI and is easy to use. It can push media resources but cannot perform play control operations on the STB.

You can download and install the clients by using the Play store application on Android mobile phones, or by using software on PCs such as the Peasecod Mobile Phone Eidolon.

iOS

Figure 10-13 shows the recommended iOS mobile phone client 8player lite.

Figure 10-13 iOS mobile phone client


8player lite
Version: 2.3.3
M-DMP/M-DMC

This client has a neat GUI and is easy to use. It can push media resources on the DMS to the DMR for playing and perform play control operations on the STB.

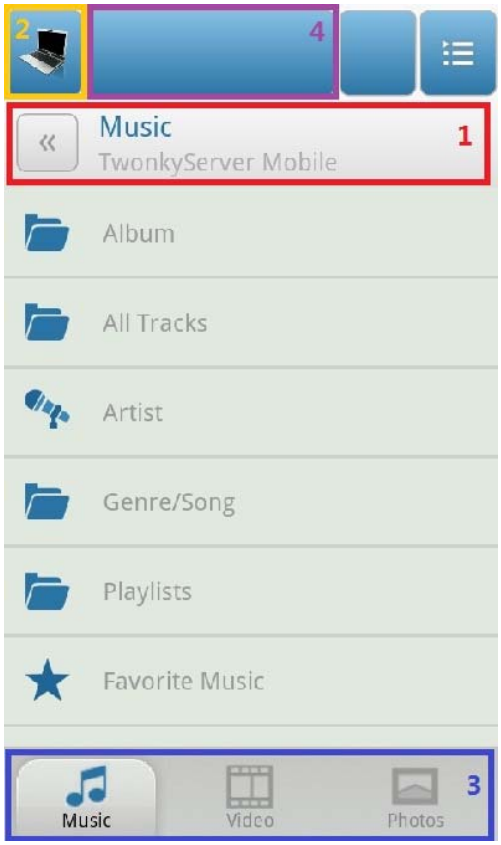
You can download and install the client from the APP store on iOS mobile phones. The trial version can only push the first five media files in the list. The official version must be paid for.

10.2.3.2 Usage of DMC Clients for Mobile Phones

Twonky Mobile

Twonky Mobile supports the M-DMP, M-DMS, and M-DMC functions. This section uses the implementation of the M-DMC function as an example to describe how this client accesses DMS resources and pushes the resources to the DMR for playing. Figure 10-14 shows the GUI of Twonky Mobile.

Figure 10-14 Twonky Mobile GUI



Step 1 Tap the red rectangle shown in Figure 10-14, the DMS list is displayed. Select the DMS device to be accessed.

Step 2 Tap the yellow rectangle shown in Figure 10-14, the DMR list is displayed. Select the DMR device to which the media resource is pushed.

Step 3 Select a media type by tapping corresponding icon in the blue rectangle shown in Figure 10-14, then select a media file, and push the selected media file to the DMR device selected in step 2 for playing.

Step 4 When the playback begins, tap the yellow rectangle shown in Figure 10-14 to switch to the corresponding playback screen. You can perform related play control operations. Use the music player as an example. You can pause or play the music that the STB is playing, as shown in Figure 10-15.

Figure 10-15 Twonky music playback screen

----End

aVia

aVia has a neat GUI and is easy to use. It can push media resources but cannot perform play control operations on the STB. Figure 10-16 shows the aVia GUI.

Figure 10-16 aVia GUI

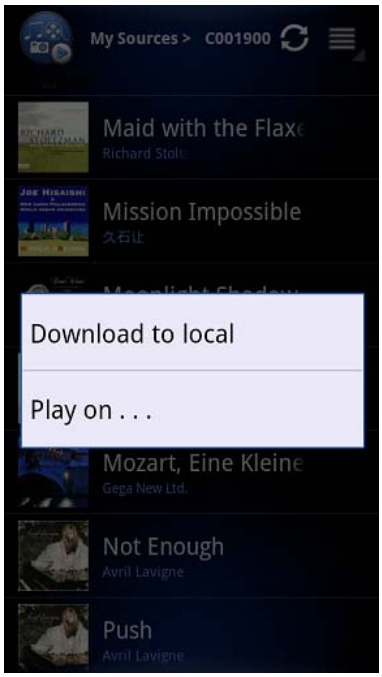
You can open **My Media** to browse local media resources, or open **My Sources** to browse media resources on DMS devices on the network. The following is an example of pushing media resources on a DMS.

Step 1 Tap **My Sources** shown in Figure 10-16. The DMS list is displayed, as shown in Figure 10-17. Select a DMS device to browse its shared media resources.

Figure 10-17 aVia DMS list screen

Step 2 Tap the media file to be pushed for a while. A menu is displayed, as shown in Figure 10-18.

Figure 10-18 aVia pop-up menu



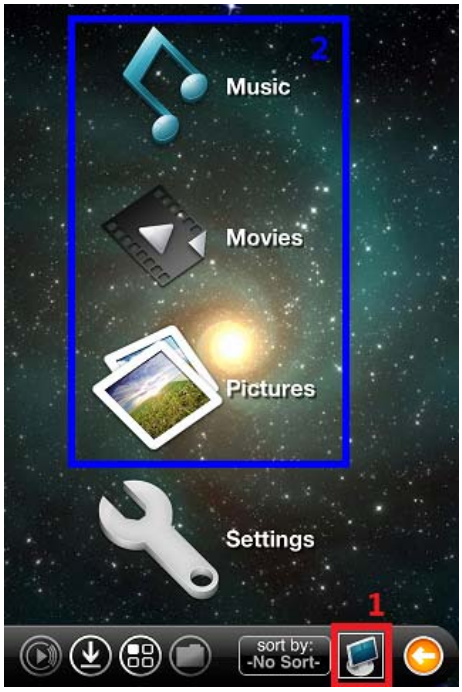
Step 3 Tap **Play on** and select a DMR device from the displayed DMR list. The media file is pushed to the selected DMR device for playing.

----End

8player

8player is a DLNA client that implements relatively comprehensive functions for iSO devices. It has a neat GUI and is easy to use. Figure 10-19 shows the 8player GUI.

Figure 10-19 8player GUI



Step 1 Tap the red rectangle shown in Figure 10-19 and select a DMR device.

Step 2 Select the type of the media to be browsed by tapping the corresponding icon in the blue rectangle shown in Figure 10-19. The DMS list is displayed, as shown in Figure 10-20.

Figure 10-20 8player DMS list screen

Step 3 Select a DMS, and select a media file to be pushed from the DMS. The selected media file is pushed to the DMR device selected in step 1 for playing. Take the playback of a music file as an example. You can pause or play the media file that is being played by the STB on the screen shown in Figure 10-21.

Figure 10-21 8player music playback screen

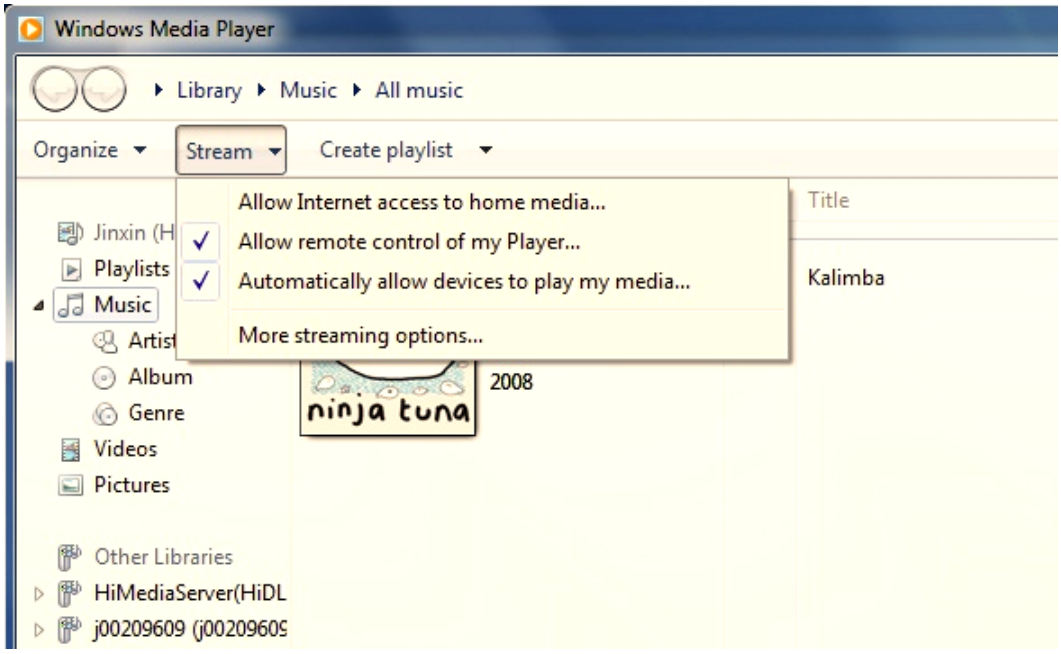
----End

10.2.3.3 Configuration of Windows Media Player 12

Windows Media Player inherent in Windows 7 supports the DMS, DMP, and DMR functions, among which the DMR and DMS functions need to be configured.

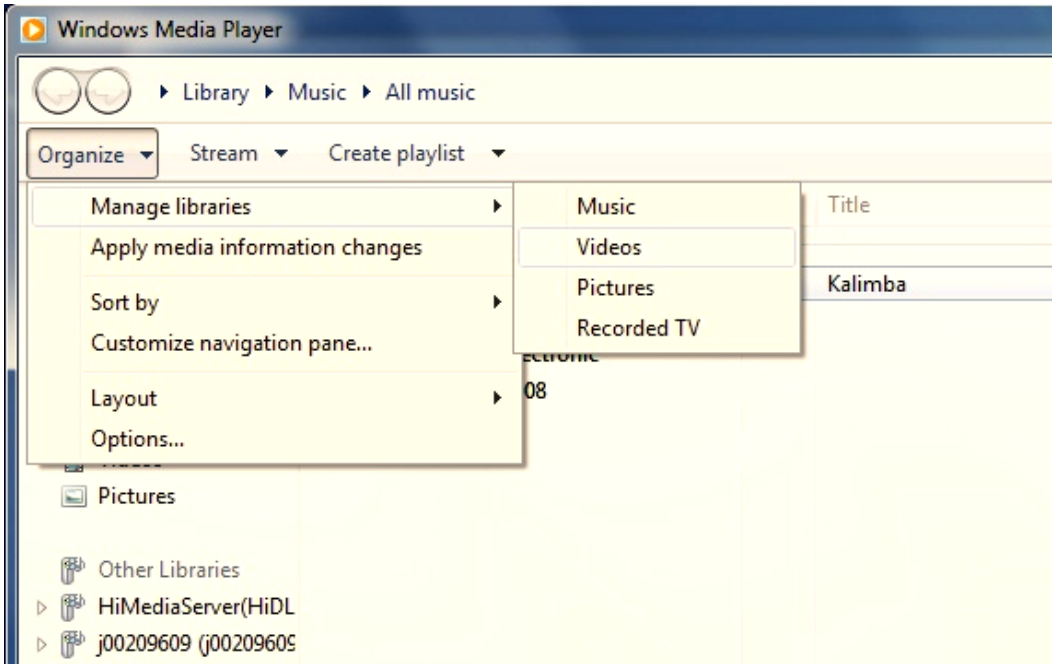
To enable the DMR and DMS functions, choose **Stream** on the GUI of Windows Media Player 12 and select **Allow remote control of my Player** and **Automatically allow devices to play my media**, as shown in Figure 10-22.

Figure 10-22 Configuring Windows Media Player 12



To manage media libraries, choose **Organize** > **Manage libraries**, and choose a media type to add corresponding media files, as shown in Figure 10-23.

Figure 10-23 Managing media libraries

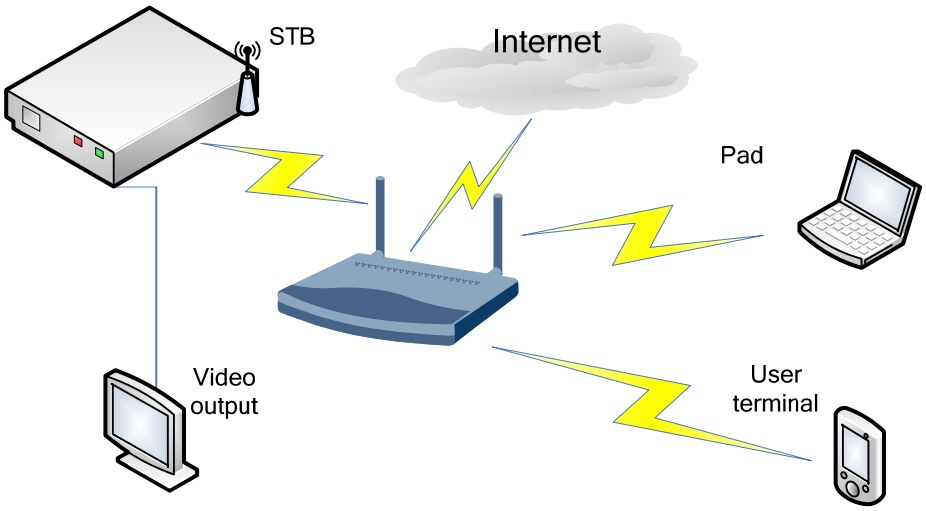


11 HiMultiScreen

11.1 Function Description

HiMultiScreen applies to LANs. It displays the STB screen outputs on a handheld device and enables the handheld device to control the STB in a remote manner. HiMultiScreen software includes the client software and server software. The client software runs on the handheld device, and the server software runs on the STB. Figure 11-1 shows the networking.

Figure 11-1 HiMultiScreen networking



HiMultiScreen consists of the device discovery module, Mirror module, RemoteControl module, VIME module, and Speech module. The modules provide the following functions:

- Device discovery

  This module discovers STBs in a LAN by sending multicast messages and describes a controlled STB in the *device name+IP address* format.

- Mirror

  This module displays the same graphics-layer information as that displayed on a controlled STB and sends operation commands to the Mirror server of the STB, allowing you to simulate touchscreen operations.

- RemoteControl

  This module simulates most common keys that are used on the IR remote control for the HiSilicon demo board, creating similar operation experience to that created by a remote control.

- VIME

  The VIME module on a client receives user inputs and transmits the user inputs to the VIME module on the STB. Then the VIME module on the STB transmits the user inputs to the current text box.

- Speech

  This module uses the client MIC as the voice input source, obtains the speech recognition result through interaction between the Internet and iFly voice cloud, and submits the recognition result to the STB server end. The Speech module can implement various voice functions such as intelligent conversation on the STB server end. It also allows you to start STB applications and search films or websites by using the voice.

11.2 Usage

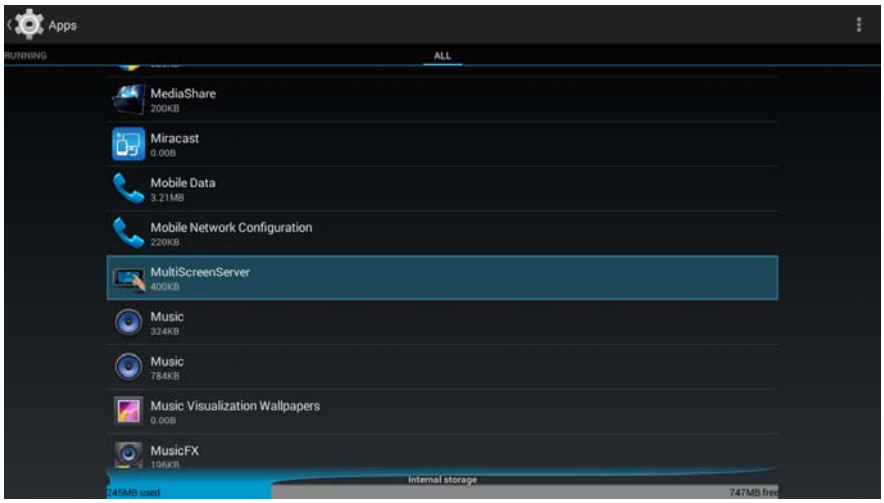
11.2.1 Network Environment

Before multi-screen networking, ensure that the STB and handheld device are in the same LAN. The Speech module requires access to the Internet. Network environment affects user experience.

11.2.2 STB

Ensure that the multi-screen application (MultiScreenServer) has been installed on the STB. You can choose **Settings** > **Apps** > **ALL** and check whether MultiScreenServer is in the list. See Figure 11-2.

Figure 11-2 Icon of the multi-screen application on the STB



Choose **Settings** > **Device info**. You can then customize the device name so that devices can be distinguished if there are multiple STB servers.

Figure 11-3 Selecting Device name

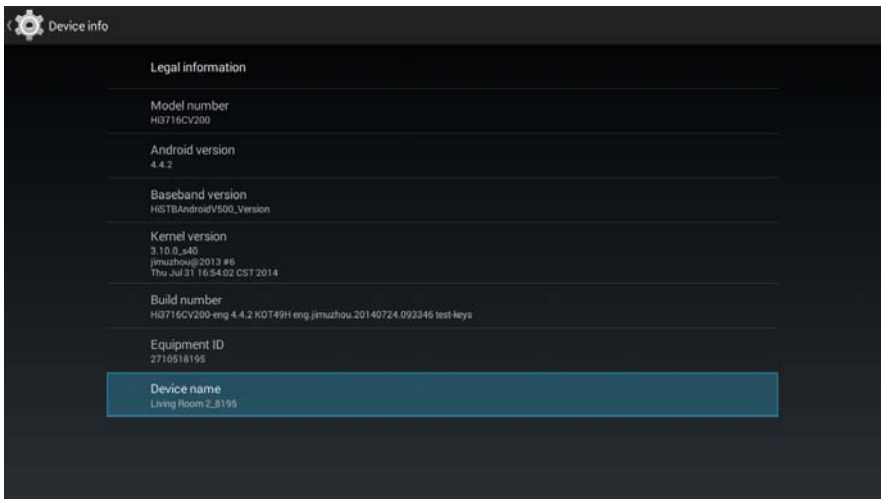
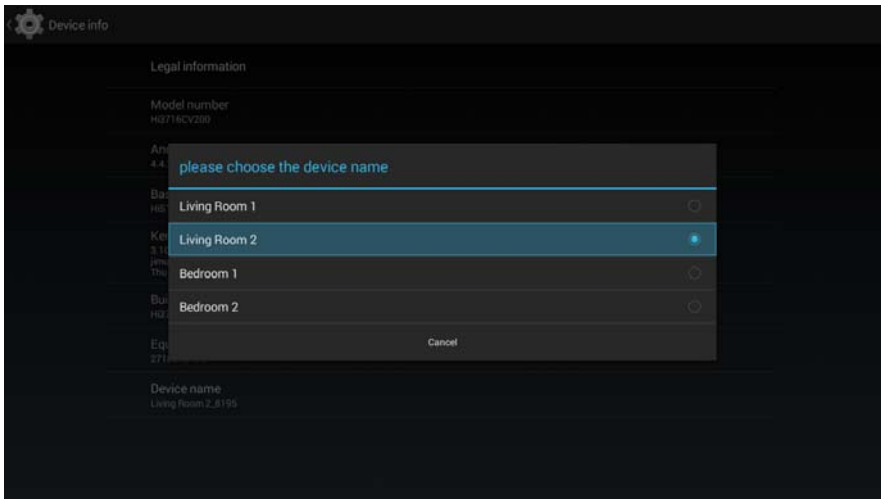


Figure 11-4 Modifying the device name



11.2.3 Handheld Device

At present, only Android (2.3 or later) devices are supported.

Enable Wi-Fi for the handheld device and connect to the LAN.

Install the multi-screen application (MoreSee). Figure 11-5 shows the application icon. Tap the icon to enable the multi-screen application.

Figure 11-5 Icon of the multi-screen application on the client

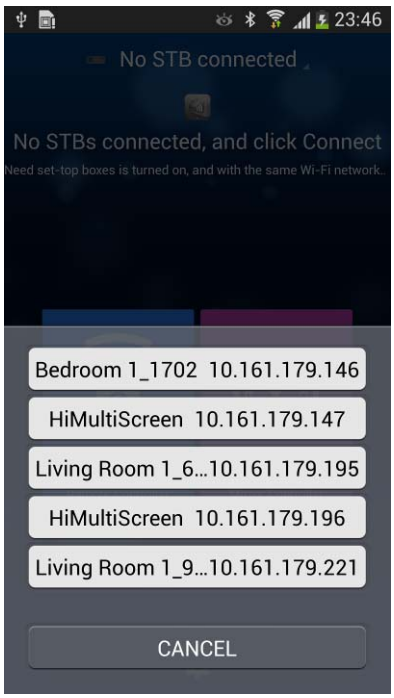
11.2.3.1 Device Discovery

Tap the application icon and access the main GUI. The application automatically searches for and displays available STBs, as shown in Figure 11-6.

Figure 11-6 Main GUI

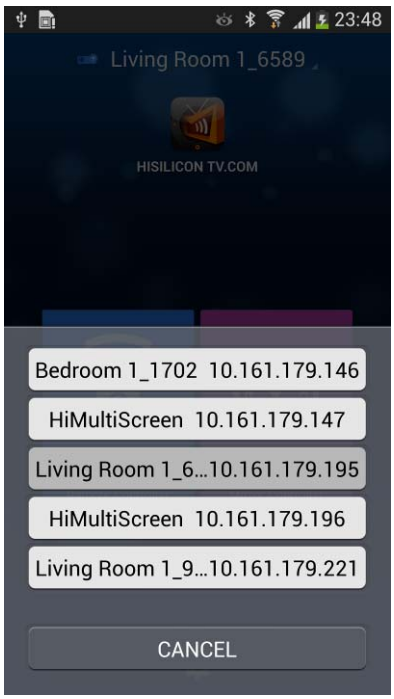
Tap **No STB connected** to manually refresh the STBs that can be connected in the LAN. You can also view information about the searched STBs in the LAN, see Figure 11-7.

Figure 11-7 Device list



Tap the STB (device name+IP address) to be connected. The tapped device in the list is selected, and information about the connected device is displayed at the top of the GUI. See Figure 11-8.

Figure 11-8 GUI after a device is connected

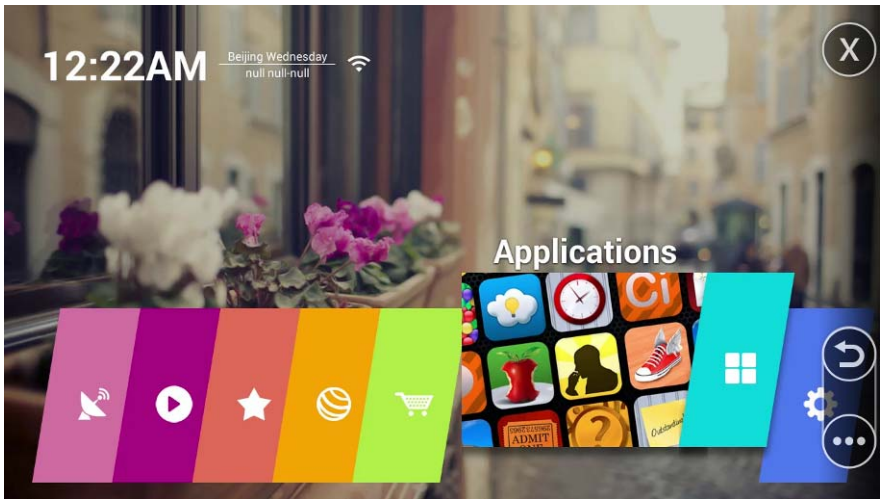


11.2.3.2 Mirroring

After an STB is connected properly, tap **Mirror Controller**, as shown in Figure 11-9. The mirroring UI of the handheld device is displayed, which displays the graphics layer of the STB end, as shown in Figure 11-10.

Figure 11-9 Mirror UI entrance

Figure 11-10 Mirror UI



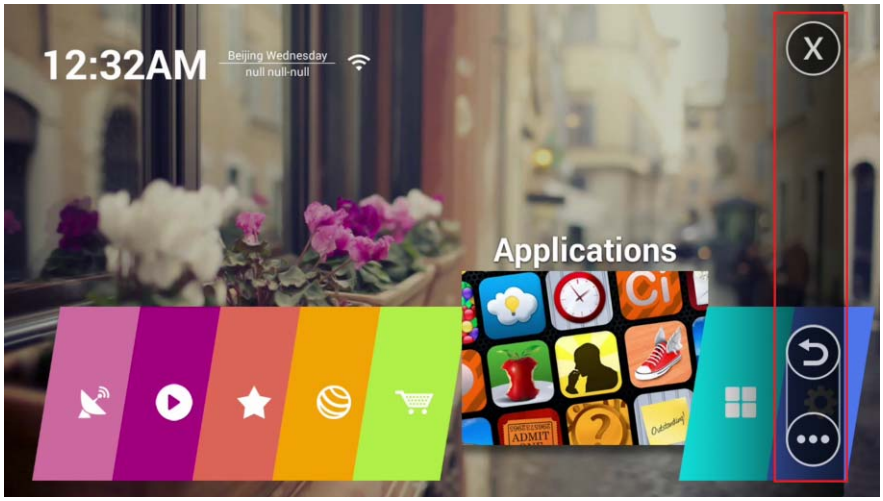
You can control the STB on the handheld device.

The Mirror screen allows you to control one STB on one handheld device. Provided that handheld device 2 attempts to connect to STB 1 when you control STB 1 on the Mirror screen

of handheld device 1, an occupation message is displayed on handheld device 1, handheld device 1 returns to the main GUI, and handheld device 2 successfully connects to STB 1.

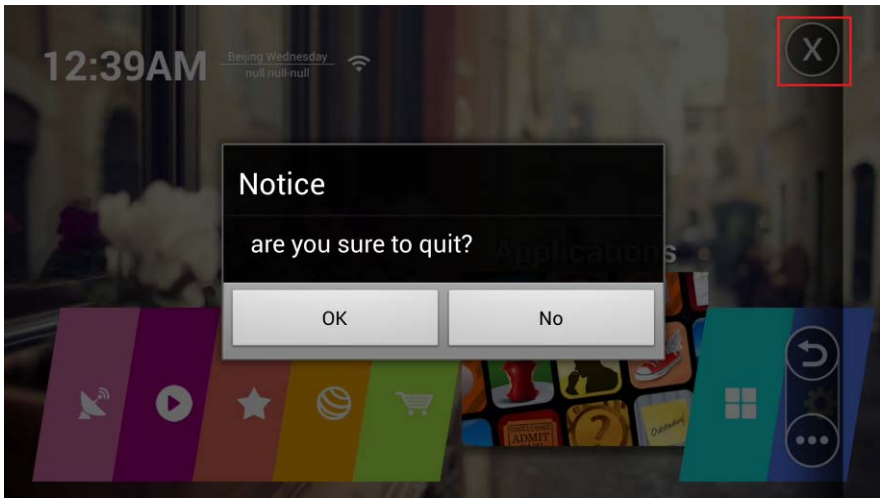
The vertical button region on the mirror UI includes some frequently used functional buttons, as shown in the red rectangle in Figure 11-11. You can drag the region to any position on the UI. The buttons are the exit, return (corresponding to the Return button on the remote control), and function buttons from top to bottom.

Figure 11-11 Vertical button region



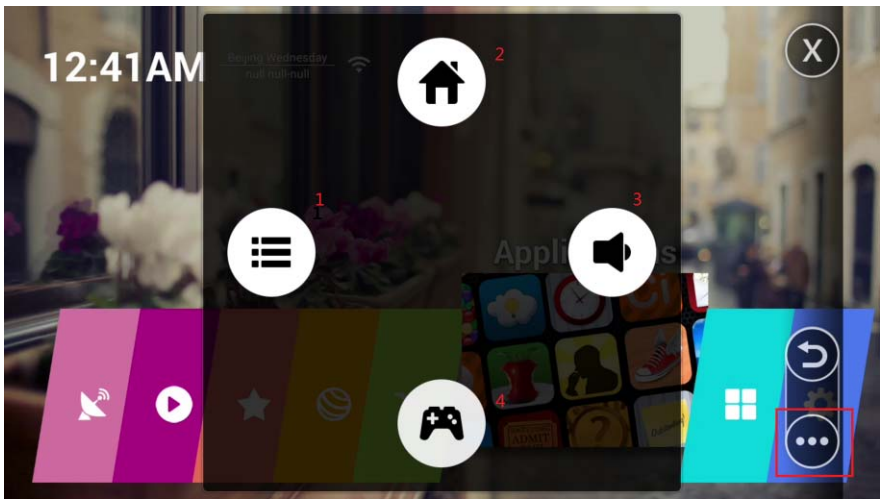
Tapping the exit button (as shown in the red rectangle in Figure 11-12) displays a message, asking whether you want to exit the mirror UI.

Figure 11-12 Displayed information



Tapping the function button (as shown in the red rectangle in Figure 11-13) displays the function button panel. Buttons 1, 2, and 3 can function as the Menu, Home and Volume buttons of the STB, and button 4 is the gaming sensing switch.

Figure 11-13 Function buttons



11.2.3.3 RemoteControl

Tapping **Remote Controller** on the main GUI enters the remote control UI. See Figure 11-14.

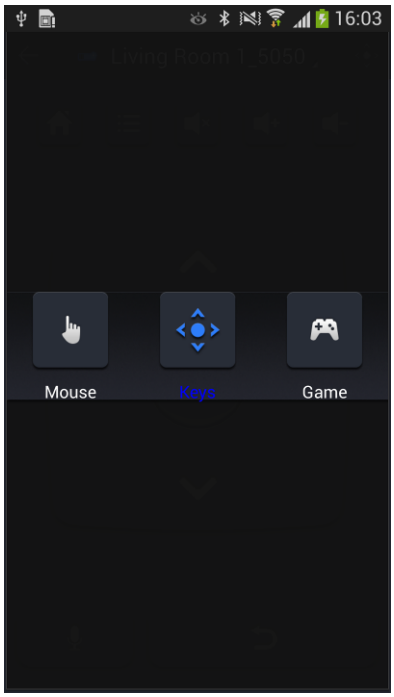
Figure 11-14 RemoteControl entrance

Tapping a button on the remote control UI sends the corresponding key value to the STB, which has the same effect as using the IR remote. As shown in Figure 11-15, the buttons at the bottom are the voice recording button and return button. The button in the red rectangle is used to switch the remote control mode. Other buttons on the UI correspond to buttons on the remote control.

Figure 11-15 Buttons on the remote control UI

Tapping the mode switch button allows you to select a remote control mode, as shown in Figure 11-16.

Figure 11-16 Selecting a remote control mode



Tap the mode switch button, and tap **Mouse**. The mouse mode UI is displayed, as shown in Figure 11-17. This UI can be used as the touchpad. Shake the mobile phone as instructed. The cursor is displayed on the STB server end, as shown in Figure 11-17. The client can be used as the mouse to control the STB server end.

Figure 11-17 Mouse mode UI

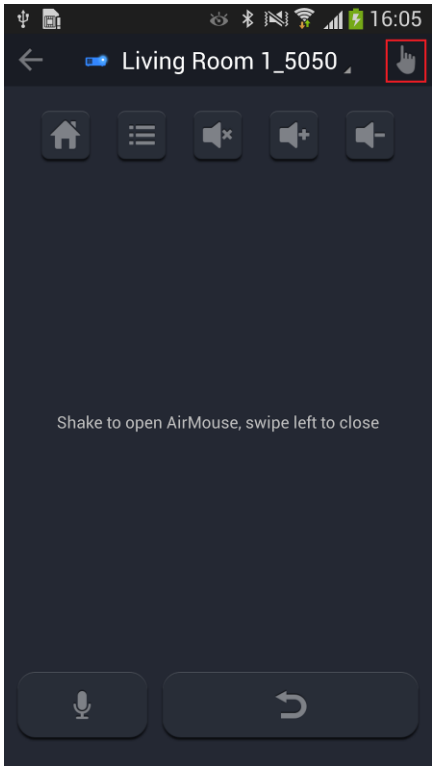
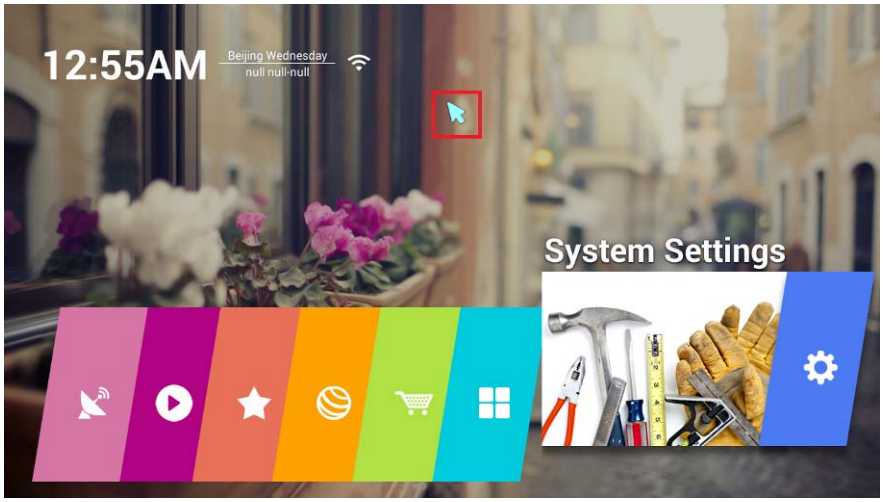
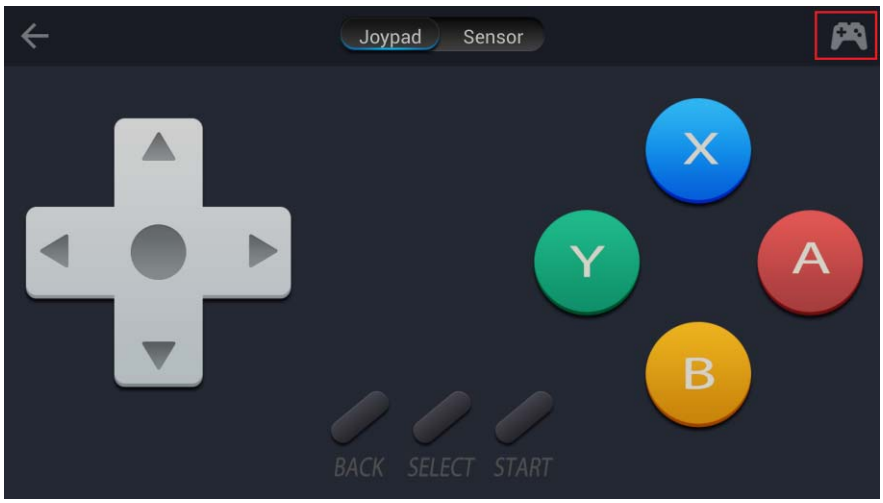


Figure 11-18 Cursor on the STB server end



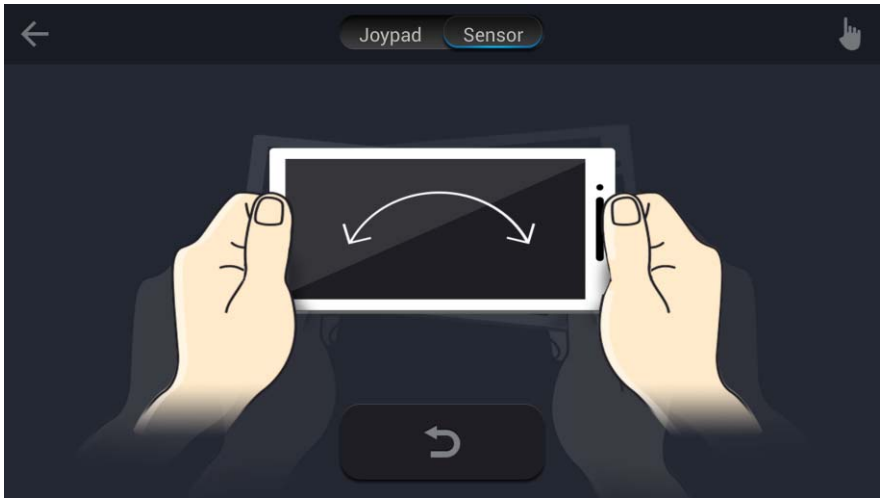
Tap the mode switch button, and tap **Game**. The gaming UI is displayed, as shown in Figure 11-19.

Figure 11-19 Gaming UI



Tap **Sensor**. The gaming sensor UI is displayed, as shown in Figure 11-20.

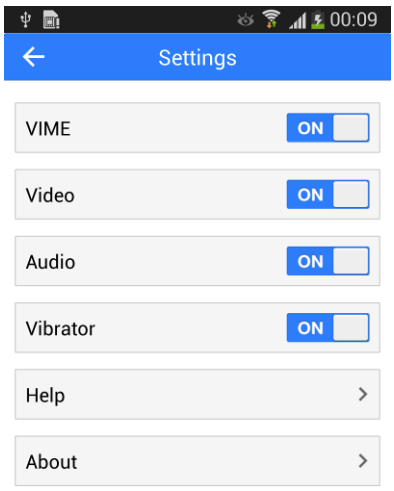
Figure 11-20 Gaming sensor UI



11.2.3.4 Settings

Tap the settings button on the main UI. The **Settings** UI shown in Figure 11-21 is displayed. You can set the virtual input method, video mirroring, audio mirroring, and remote control vibration. Tapping **Help** enters the guide UI, and tapping **About** displays the software version information.

Figure 11-21 Settings

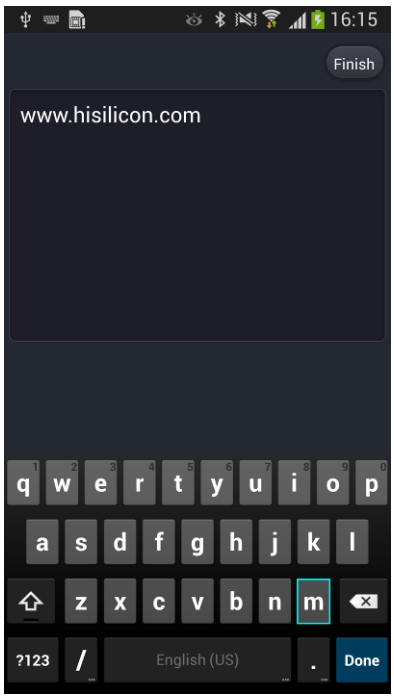


11.2.3.5 Virtual Input Method

The virtual input method allows you to enter texts on the STB by using the handheld device.

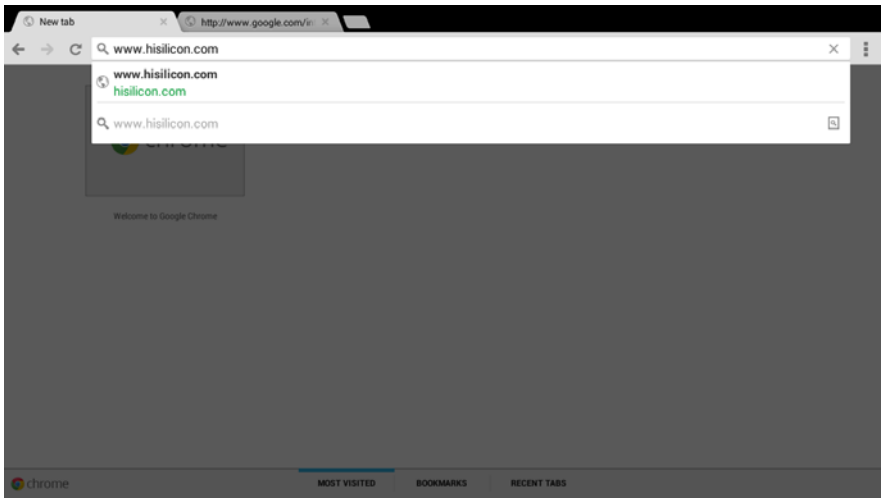
Ensure that **VIME** is set to **ON** on the Settings UI (ON by default). On the remote control or mirror UI, tap any textbox on the STB server end. The input method is displayed on the handheld device end, as shown in Figure 11-22. The UI in Figure 11-22 is displayed after the address bar of the Chrome browser is tapped.

Figure 11-22 Virtual input method



After entering the texts, tap **Done**. The content in the textbox of the STB server end is updated, as shown in Figure 11-23.

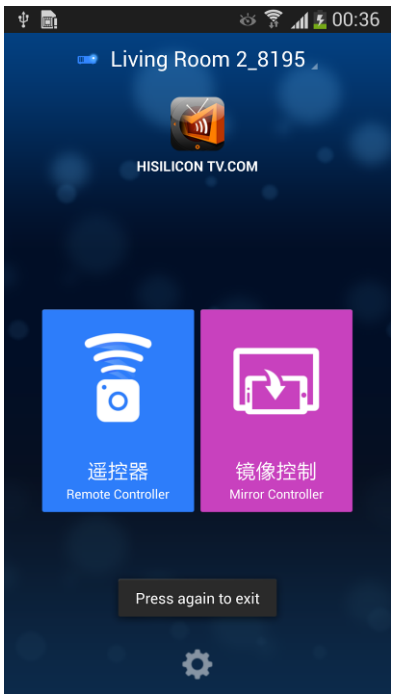
Figure 11-23 Content in the textbox of the STB server end



11.2.3.6 Exiting MoreSee

To exit MoreSee, tap the Back key of the mobile phone for two consecutive times. If you tap the Back key for only once, "Press again to exit" is displayed, as shown in Figure 11-24.

Figure 11-24 "Press again to exit"



11.2.4 Proc Guide

11.2.4.1 Proc Description

The proc information helps development personnel check the current running status and related running information of the HiMultiScreen rapidly. Running the **cat** command displays the HiMultiScreen running information, and running the **echo** command allows you to change the related parameters of the current device. The mirror module supports the **echo** command currently.

11.2.4.2 Using the Cat Command

To view the HiMultiScreen running information by using the **cat** command, perform the following steps:

Step 1 Enable the multiscreen service at the STB end.

Step 2 Enable the serial port and go to the **Proc** directory of the HiMultiScreen by running the following command:

```
root@android:/ # cd proc/hisi/himultiscreen/
```

Step 3 View the entries by running the following command:

```
root@android:/proc/hisi/himultiscreen # ls
```

Figure 11-25 Entries before the client is connected

```
root@android:/proc/hisi/himultiscreen # ls
gsensors
upnp_multiscreen_device
upnp_multiscreen_device_action
```

Figure 11-26 Entries after the client is connected

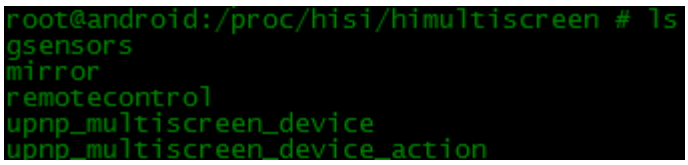


Table 11-1 Entries in the HiMultiScreen

Entry	Description
gsensors	Status information of the G-sensor module
mirror	Status information of the audio/video mirror module
remotecontrol	Status information of the remote control (touch, mouse, and keyboard) event processing module
upnp_multiscreen_device	UPnP status
upnp_multiscreen_device_action	Action information related to the UPnP service

Step 4 View the related entry information as required.

- After the client is connected, view the upnp_multiscreen_device entry (including the device information, service information, and related status variable information) by running the following command:

  ```
  root@android:/proc/hisi/himultiscreen # cat upnp_multiscreen_device
  ```

Figure 11-27 Entry upnp_multiscreen_device instance

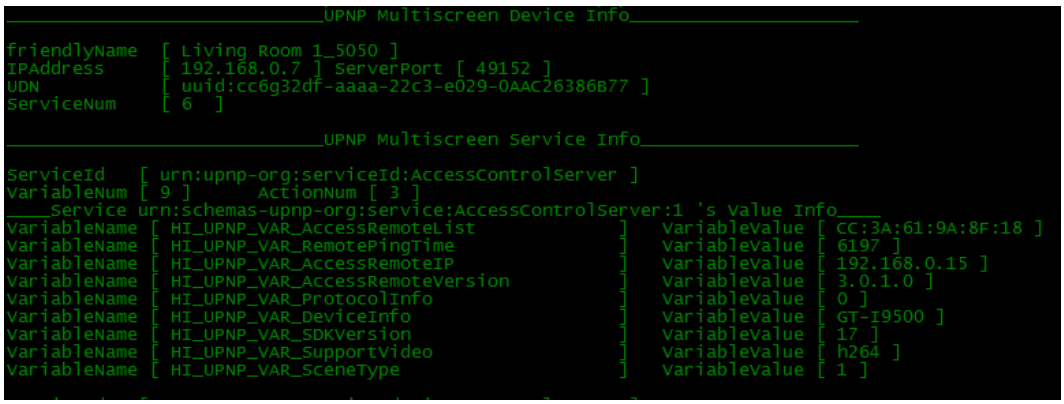


Table 11-2 Items in upnp_multiscreen_device

Items	Description
friendlyName	Host name

Items	Description
IPAddress	IP address for the device
ServerPort	Service port ID of the device
UDN	Unique device name
ServiceNum	Number of services provided by the device
ServiceId	Service name
VariableName	Name of the status variable related to the service
VariableValue	Value corresponding to the variable name

- After the client is connected, view the action information provided by the UPnP service (including the name of each service and the action information provided by each service) by running the following command:

  ```
  root@android:/proc/hisi/himultiscreen # cat
  upnp_multiscreen_device_action
  ```

Figure 11-28 Entry upnp_multiscreen_device_action instance

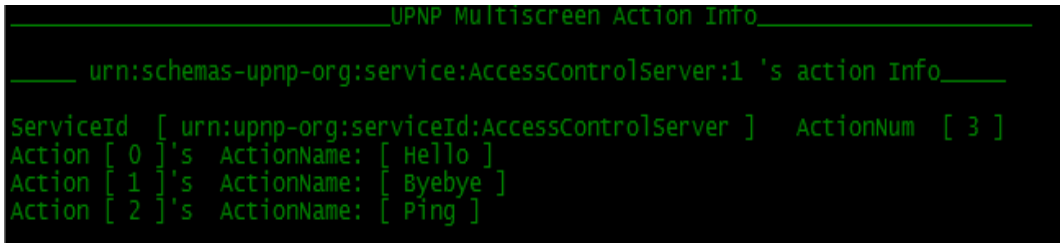


Table 11-3 Items in upnp_multiscreen_device

Item	Description
ServiceId	Service name
ActionNum	Number of actions provided by the service
ActionName	Action name

- After the client is connected, view the mirror-related information by running the following command:

  ```
  root@android:/proc/hisi/himultiscreen # cat mirror
  ```

Figure 11-29 Entry mirror H.264 transmission instance

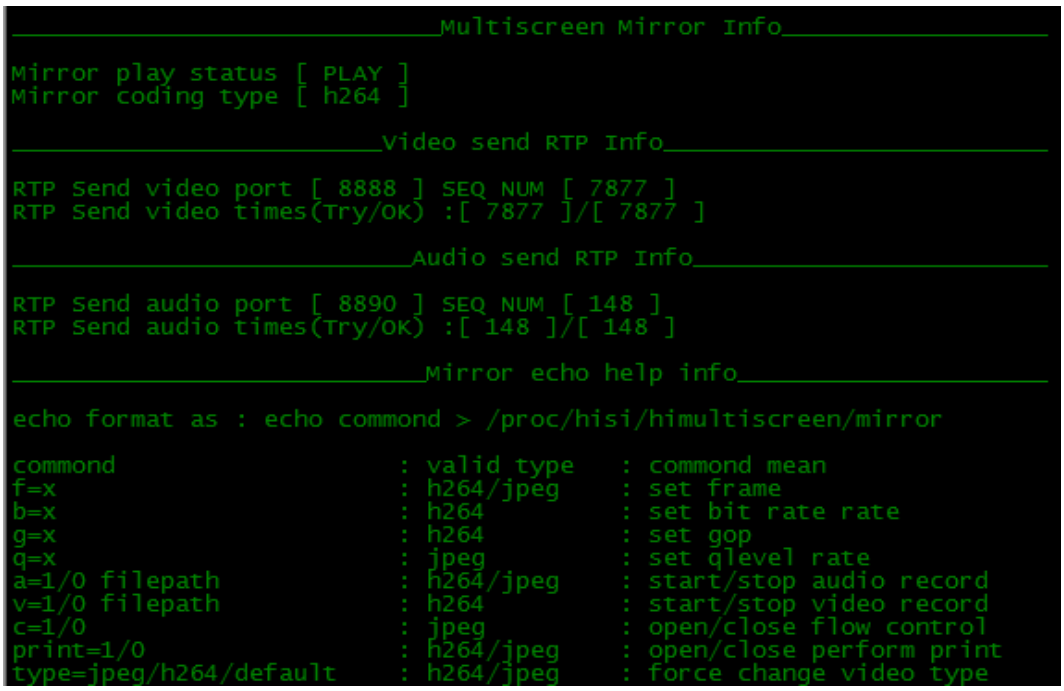


Table 11-4 Items in mirror h264 transmission

Item	Description
Multiscreen Mirror Info	Multiscreen mirror information
Mirror play status	Mirror video playback status (PLAY/PAUSE)
Mirror coding type	Video encoding type
Video send RTP Info	Video RTP TX information
RTP Send video port	Video TX port
SEQ NUM	TX packet sequence number
RTP Send video times(Try/OK)	Number of video transmission attempts/successes
Audio send RTP Info	Audio TX information
RTP Send audio port	Audio TX port
RTP Send audio times(Try/OK)	Number of audio transmission attempts/successes
Mirror echo help info	Help information of the echo command
Echo format as	Format of the echo command
command	Echo command
valid type	Valid transmission type for the related command
command mean	Function of the related command

Figure 11-30 Entry mirror JPEG transmission instance

Table 11-5 Items in mirror H.264 transmission

Item	Description
Multiscreen Mirror Info	Multiscreen mirror information
Mirror play status	Mirror video playback status (PLAY/PAUSE)
Mirror coding type	Video encoding type
Send jpeg height/weight	Width/Height of the transmitted JPEG image
encode Qlevel	Picture quality parameter
send whole / part frame rate	Frame rate for transmitting entire frames/part of the frames

Item	Description
Mirror send whole frame try/success	Number of attempts/successes for transmitting entire frames
Mirror send part frame try/success	Number of attempts/successes for transmitting part of the frames
VO Info	Video output information
VO refresh status	VO refresh status
VO Cast_Frame_Index	Sequence number of the frame obtained by the VO
VO original height/weight/PTS	Height/Width/PTS of original frames of the VO
VO AcquireCastFrame try/success	Number of attempts/successes for obtaining frames
TDE API CALL Info	Information about calling the graphics encoding API
TDE DOWNSCALE try /success	Number of attempts/successes of TDE down-sampling
TDE QuickCopy try /success	Number of attempts/successes of TDE quick copy
TDE JpegEncode try /success	Number of attempts/successes of TDE JPEG encoding
Flow Control Info	Flow control information
flow control status	Flow control enable/disable
client rtp lost num /rtp interval / afford frame rate	Number of lost packets returned by the client/Interval for counting lost packets/Allowed frame rate
Video send RTP Info	Video RTP TX information
Audio send RTP Info	Audio TX information
Mirror echo help info	Help information of the echo command

- After the client is connected, view the remote control information by running the following command:

```
root@android:/proc/hisi/himultiscreen # cat remotecontrol
```

Figure 11-31 Entry remotecontrol instance

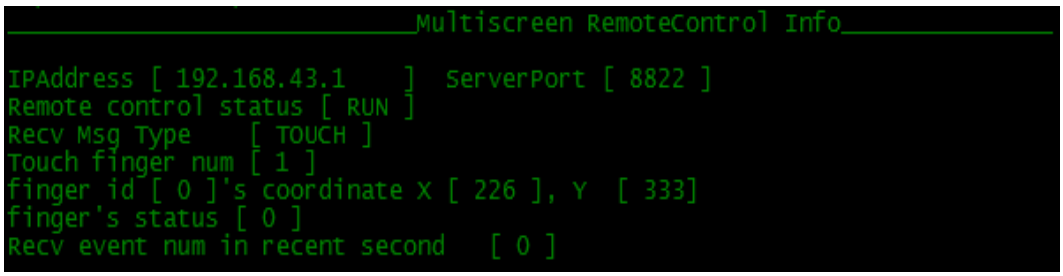


Table 11-6 Items in the remote control entry

Item	Description
IPAddress	Multiscreen mirror information
ServerPort	Mirror video playback status (PLAY/PAUSE)
Remote control status	Video encoding type
Recv Msg Type	Type of the last received message (TOUCH/KEYBOARD)
Touch finger num	Number of fingers in the latest touch operation
finger id	ID of the last touched mobile phone
finger's status	Finger press status: **0** indicates released and **1** indicates pressed.
coordinate X/Y	X/Y coordinates of the client touchscreen
key value	Key value
key status	Key status: **0** indicates released and **1** indicates pressed.
Recv event num in recent second	Number of events received from the client in the last second

- After the client is connected, enable the sensor of the client, and view the gsensors related information by running the following command:

  ```
  root@android:/proc/hisi/himultiscreen # cat gsensors
  ```

Figure 11-32 Entry gsensors instance

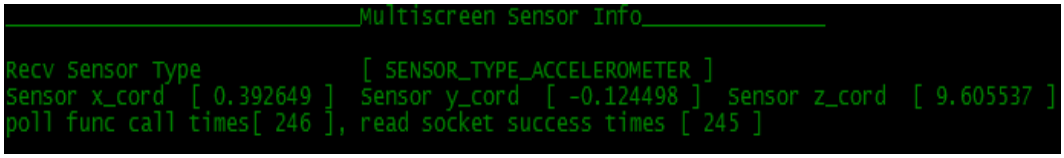


Table 11-7 Items in the gsensors entry

Item	Description
Recv Sensors Type	Type of the sensors received from the client
Sensor x_cord/y_cord/z_cord	X/Y/Z coordinates of the last received sensor
poll func call times	Number of times that the API for reading the sensor hardware data is called
read socket success times	Number of times that the network port is read successfully

----End

11.2.4.3 Using the Echo Command

To run the **echo** command, perform the following steps:

Step 1 Enable the multiscreen service on the STB, and connect the client to the multiscreen service.

Step 2 Enable the serial port, and view the current video transmission format by running the **cat** command. Run the **echo** command. Table 11-8 describes the supported **echo** commands. For example, to set the frame rate, run the following command:

```
echo f=20 > proc/hisi/himultiscreen/mirror
```

----End

Table 11-8 Echo commands

Echo Command	Description
echo print=**x** > /proc/hisi/himultiscreen/mirror	• This command is used to display the time and performance logs. • If **x** is **1**, the logs are displayed; if **x** is **0**, the logs are not displayed. • This command is valid when the transmission format is JPEG or H.264. • You can check the command execution by using ddms or running **logcat** over the serial port.

Echo Command	Description
echo type=**x** > /proc/hisi/himultiscreen/mirror	• This command is used to set the mirror transmission mode forcibly. • If **x** is **h264**, the transmission format is H.264; if **x** is **jpeg**, the transmission format is JPEG; if **x** is **default**, the transmission format is determined by the compatibility list. • This command is valid when the transmission format is JPEG or H.264. After configuration, the command takes effect when the client is exited and reconnected. • After this command is used, you need to restore to the default transmission format. • You can view the transmission type by running **cat mirror**.
echo a=**x** **filepath** > /proc/hisi/himultiscreen/mirror	• This command is used to record audio. • If **x** is **1**, the audio recording function is enabled; if **x** is **0**, the audio recording function is disabled. **filepath** indicates the path and name of the recorded file, for example, **/mnt/sdcard/audiorec.aac**. The extension name of the audio file is .aac. • The recorded file needs to be deleted manually. • This command is valid when the transmission format is JPEG or H.264. • You can view the recorded files in the corresponding path to verify the command execution result.
echo v=**x** **filepath** > /proc/hisi/himultiscreen/mirror	• This command is used to record videos. • If **x** is **1**, the video recording function is enabled; if **x** is **0**, the video recording function is disabled. **filepath** indicates the path and name of the recorded file, for example, **/mnt/sdcard/videorec.264**. The extension name of the video file is .264. • The recorded file needs to be deleted manually. • This command is valid when the transmission format is H.264. • You can view the recorded files in the corresponding path to verify the command execution result.

Echo Command	Description
echo f=**x** > /proc/hisi/himultiscreen/mirror	• This command is used to set the frame rate (frame rate for transmitting entire frames during the JPEG format transmission or output frame rate during the H.264 format transmission). • x ranges from 0 to 30. • This command is valid when the transmission format is JPEG or H.264. If the transmission format is JPEG, the flow control needs to be disabled for the configuration to take effect. • If the transmission format is JPEG, you can verify the command execution result by running **cat mirror**. If the transmission format is H.264, you can check whether **FrmRate** is configured successfully by running **cat venc**.
echo b=**x** > /proc/hisi/himultiscreen/mirror	• This command is used to set the bit rate. • The reference values of x as follows: greater than 5 MB (5242880) for 1080p, greater than 3 MB (3145728) for 720p, and about 2 MB (2097152) for D1. • This command is valid when the transmission format is H.264. • Check whether **FrmRat**e is configured successfully by viewing the VENC parameter **TargetBitRate**. If the value of **TargetBitRate** is around the reference values, the configuration is successful.
echo g=**x** > /proc/hisi/himultiscreen/mirror	• This command is used to set the GOP value (I frame interval). • x ranges from o to 2147483647 (0x7fffffff). • This command is valid when the transmission format is H.264. • You can check whether the configuration is successfully by viewing the **Gop** item in VENC.
echo c=**x** > /proc/hisi/himultiscreen/mirror	• This command is used to enable/disable flow control. • If **x** is **1**, flow control is enabled; if **x** is **0**, flow control is disabled. • This command is valid when the transmission format is JPEG. • You can check whether flow control is enabled or disabled by running **cat mirror**.

Echo Command	Description
echo q=**x** > /proc/hisi/himultiscreen/mirror	• This command is used to set the picture quality parameter **Qlevel**. • x ranges from 0 to 100. • This command is valid when the transmission format is JPEG. This command takes effect after the flow control is disabled. • You can check whether **Qlevel** is configured successfully by running **cat mirror**.

12 HiMiracast

12.1 Function Description

The Wi-Fi Alliance launched the Wi-Fi CERTIFIED Miracast program, a certification program, in 2012. Miracast-certified devices provide simplified discovery and settings, allowing you to quickly transfer audio and video clips among devices. Miracast is customized by mobile and consumer electronics manufacturers and chip vendors in the Wi-Fi Alliance.

12.1.1 Basic Concepts

Wi-Fi Direct

Wi-Fi Direct is a wireless network interconnection protocol launched by the Wi-Fi Alliance in 2010. It allows devices in a wireless network to interconnect with each other without wireless routers. Wi-Fi Direct supports interconnection between two devices or among more devices. It is compatible with all Wi-Fi devices from 11a/b/g to 11n, enabling direct interconnection among Wi-Fi devices that comply with different standards. In Miracast services, Wi-Fi Direct is used to discover and connect devices.

Wi-Fi Station

Wi-Fi Station is a network member and structural station of Wi-Fi. It is the basic component of the network.

Concurrent

In concurrent mode, Wi-Fi Direct and Wi-Fi Station are working simultaneously.

Wi-Fi Display

Wi-Fi Display is a standard protocol created by the Wi-Fi Alliance. It combines the Wi-Fi standard and H.264 video coding technology, allowing you to mirror audio and video files from a mobile device to a large screen in real time and providing support for reliable transmission and playing among devices anytime anywhere.

Relationship Between Miracast and Wi-Fi Display

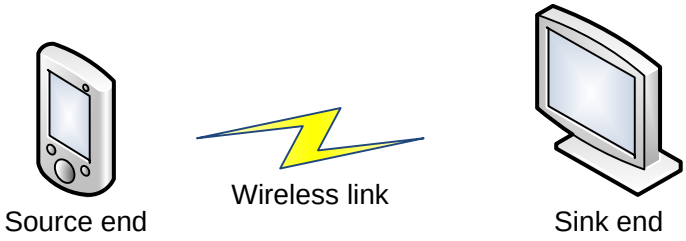
Miracast is launched by the Wi-Fi Alliance to certify devices that provide the Wi-Fi Display function. The Miracast label is attached to certified devices.

12.1.2 Features

12.1.2.1 Point-to-Point Connection

As shown in Figure 12-1, the source end serves as the transmission end to encode and transmit audio and video data over Miracast. The sink end serves as the reception end to receive and decode audio and video streams sent from the source end and send the decoded data to the display.

Figure 12-1 Point-to-point connection over Miracast



12.1.2.2 Coding and Transmission of High-Definition Video Clips

By using the H.264 video coding technology, Miracast provides maximum 1920 x 1080 resolution and 60 fps frame rate, which are specified in the Consumer Electronics Association (CEA) standard in the Wi-Fi display protocol. Devices in the current market support maximum 1280 x 720 resolution and 60 fps frame rate, providing perfect visual experience.

12.1.2.3 Synchronous Transmission of Audio and Video Clips

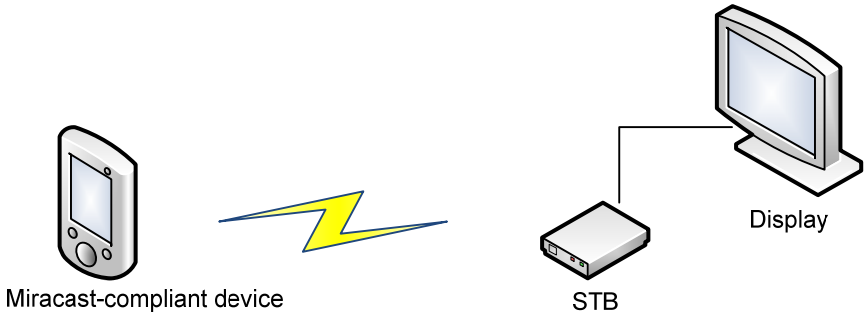
Miracast supports synchronous transmission of audio and video clips from the source end to the sink end. The supported audio formats specified in the Wi-Fi display protocol include LPCM, AAC, and AC3. At present, HiSilicon Miracast supports only AAC.

12.2 Usage

12.2.1 HiMiracast Application Scenario

Figure 12-2 shows the HiMiracast application scenario.

Figure 12-2 HiMiracast application scenario



As shown in Figure 12-2, the topology consists of the mobile phone or pad, STB, and display. The mobile phone or pad serves as the source end to generate MPEG-TS streams in Miracast services. (MPEG stands for Moving Picture Experts Group, and TS stands for Transport Stream.) At present, mobile phones, such as LG Nexus4, M2, M2A, M2S, ASUS ME173X, and Huawei G700 provide Miracast services. The STB decodes and receives TS streams and sends the decoded data to the display over a data line. The STB integrates the wireless network adapter for Wi-Fi Display to connect to the mobile phone or pad and transmit data.

12.2.2 HiMiracast Operation Guide

12.2.2.1 Procedure

Step 1 Compile the Android SDK.

Take HiSTBAndroid V600 as an example. Compile the HiSTBAndroid V600R001C00SPCXXX (Android) SDK based on the *Android Solution Development Guide* in the SDK. Burn images to the board as required after successful compilation.

Step 2 Install a Wi-Fi dongle.

If the board does not integrate the Wi-Fi hardware, a Wi-Fi dongle with the USB port is required. At present, Wi-Fi chips, such as RTL8188EUS, RTL8188ETV, and Atheros 9374, provide Miracast services. After you connect a Wi-Fi dongle to the USB port on the board, verify the availability of Wi-Fi. For details, see Step 3. Installation of the Wi-Fi dongle affects the Miracast performance. Take the following precautions when you install the Wi-Fi dongle:

CAUTION

If the distance between the USB port and high definition multimedia interface (HDMI) on the demo board is less than 10 cm, the HDMI interferes with Wi-Fi, which may cause Miracast connection failures or frequent connection interruption. This problem also occurs when the antenna on the Wi-Fi dongle is close to the HDMI.

You can resolve the preceding problem by using either of the following methods:

- Connect the Wi-Fi dongle by using a USB extension line to keep it far away from the HDMI.

- Select a USB port without a USB slot on the demo board and ask hardware personnel to weld a USB slot for the USB port.

Step 3 Check whether Wi-Fi is available.

Enable and disable Wi-Fi for multiple times, as shown in Figure 12-3 and Figure 12-4. If the Wi-Fi can be enabled and disabled properly, it is available.

Figure 12-3 Disabling Wi-Fi

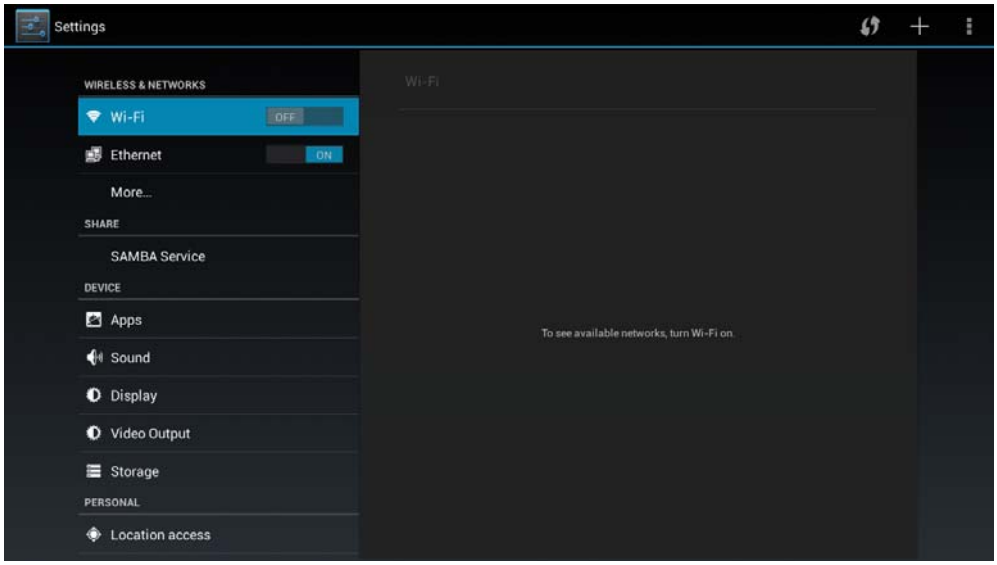
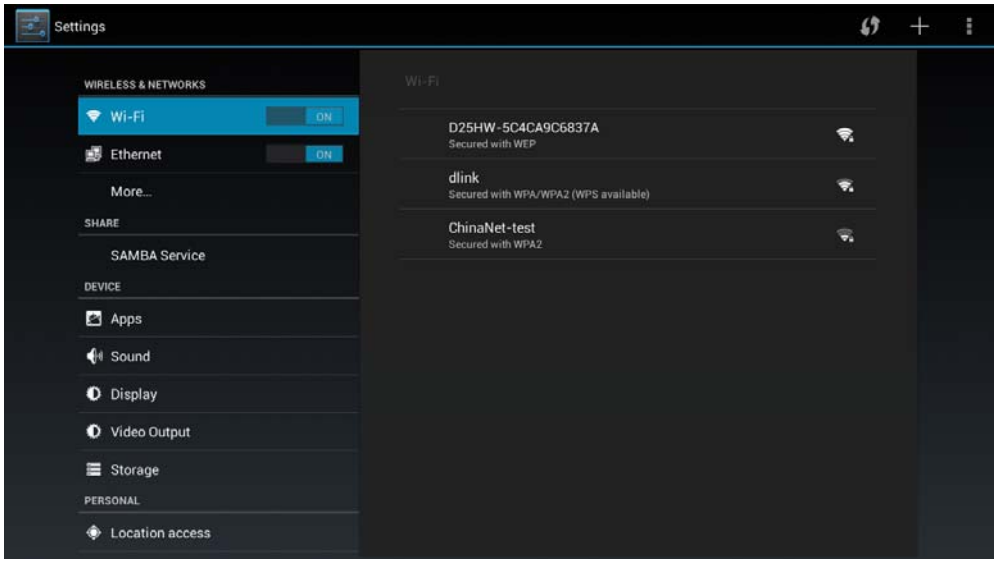


Figure 12-4 Enabling Wi-Fi



Step 4 Configure the Wi-Fi P2P frequency for the board.

Configure the 5 GHz environment for the board:

1. Disable Wi-Fi on the board.
2. Connect to the board over the serial port. Go to the **/data/misc/wifi** directory, and delete **p2p_supplicant.conf** by running the following commands:

   ```
   #cd /data/misc/wifi
   #rm p2p_supplicant.conf
   ```

3. Go to the **/etc/wifi** directory and configure **p2p_supplicant.conf** by running the following commands:

   ```
   #cd /etc/wifi
   #adb remount
   #busybox vi p2p_supplicant.conf
   ```

   The modifications are as follows:

   ```
   # Listen channel
   p2p_listen_reg_class=124
   p2p_listen_channel=149

   # Operation channel
   p2p_oper_reg_class=124
   p2p_oper_channel=149
   ```

CAUTION

p2p_listen_reg_class and **p2p_oper_reg_class** are country codes, and **p2p_listen_channel** and **p2p_oper_channel** are channel frequencies (153, 157, and 161 are also supported besides 149).

4. Save the file and exit. Restart the board.

Configure the 2.4 GHz environment for the board:

1. Disable Wi-Fi on the board.
2. Connect to the board over the serial port, go to the **/data/misc/wifi** directory, and delete **p2p_supplicant.conf** by running the following commands:

   ```
   #cd /data/misc/wifi
   #rm p2p_supplicant.conf
   ```

3. Go to the **/etc/wifi** directory and configure **p2p_supplicant.conf** by running the following commands:

   ```
   #cd /etc/wifi
   #adb remount
   #busybox vi p2p_supplicant.conf
   ```

   The modifications are as follows:

   ```
   # Listen channel
   p2p_listen_reg_class=81
   p2p_listen_channel=1
   ```

```
# Operation channel
p2p_oper_reg_class=81
p2p_oper_channel=1
```

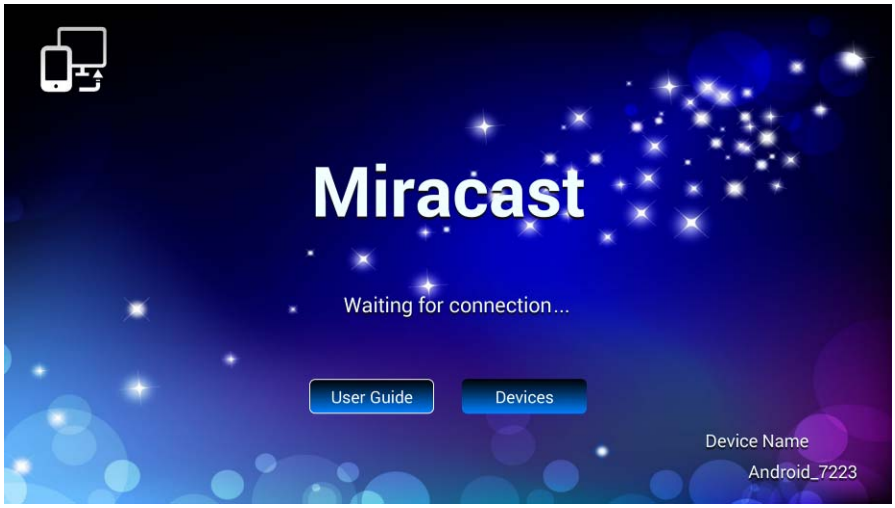
4. Save the file and exit. Restart the board.

Step 5 Start the Miracast application.

After you verify the availability of Wi-Fi, return to the application list and start the Miracast application.

- If the board has been connected to a wireless router, as the Wi-Fi operates in concurrent mode, the Wi-Fi Station affects Miracast services, causing artifacts or mosaics. To resolve this problem, the Miracast application directly disconnects the board from the AP, and the current wireless services (for example, pushing pictures by using the DLNA through the AP) of the board are interrupted.
- If the board is not connected to a wireless router, the Miracast screen is displayed. The Miracast sink end is in the **Waiting for connection** state, and the name of the current device is displayed in the lower right corner for searching and discovery, as shown in Figure 12-5.

Figure 12-5 Miracast screen



- There are two buttons on the preceding Miracast screen: **User Guide** and **Devices**. Selecting **User Guide** enters the user guide screen, which describes how to configure the mobile phone. There are four pages, and you can turn the page by using the left/right arrow keys. See Figure 12-6. Selecting **Devices** displays the mobile devices supported by the Miracast. See Figure 12-7.

Figure 12-6 User Guide screen

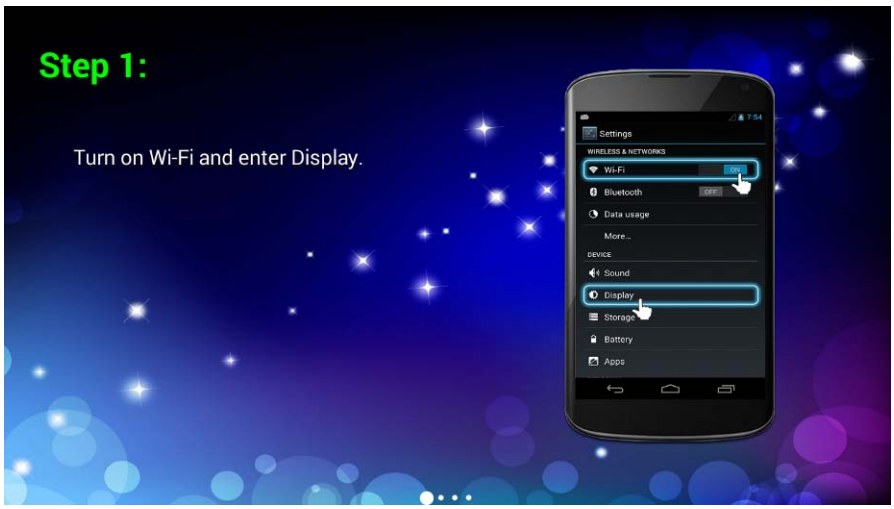
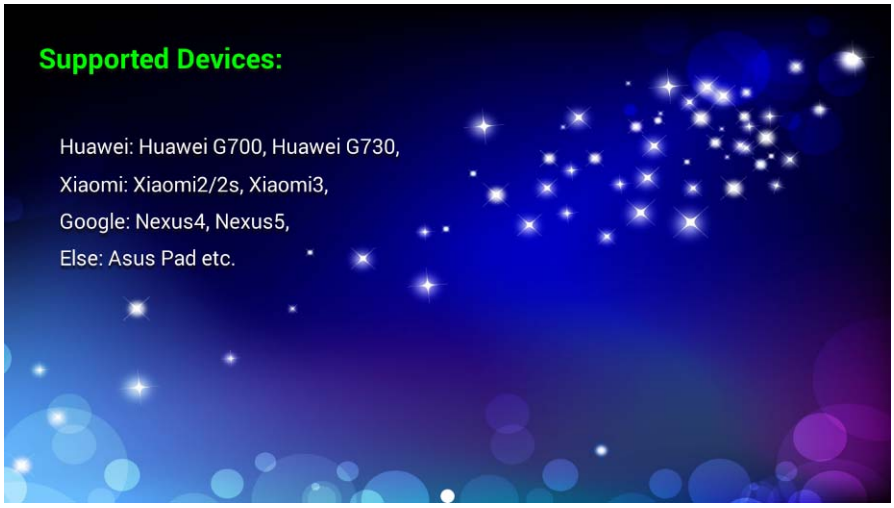


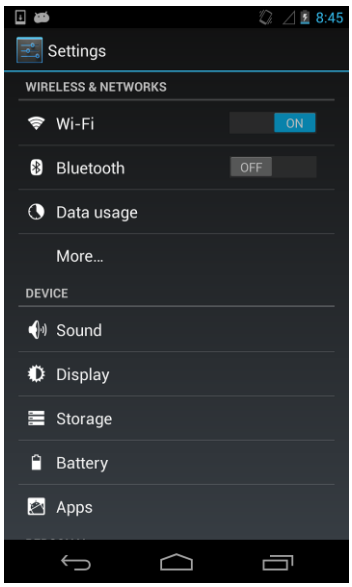
Figure 12-7 Devices supported by the Miracast



Step 6 Configure the mobile phone or pad.

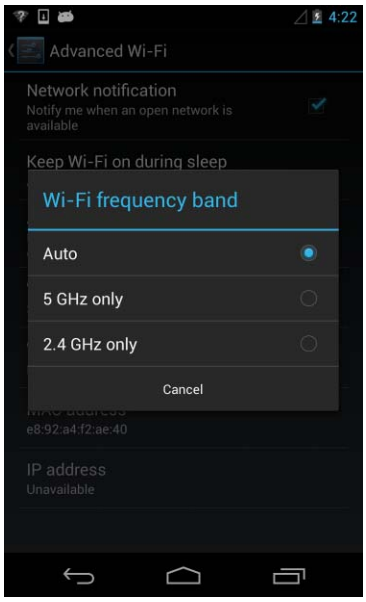
The mobile phone or pad serves as the data source end to send MPEG-TS streams to the sink end over wireless network. Before you start Miracast services, enable Wi-Fi, as shown in Figure 12-8.

Figure 12-8 Enabling Wi-Fi



Select the Wi-Fi frequency band based on the wireless frequency band of the board. If the frequency band is 5 GHz on the board end, select **Auto** or **5 GHz only**; if the frequency band is 2.4 GHz on the board end, select **Auto** or **2.4 GHz only**. See Figure 12-9.

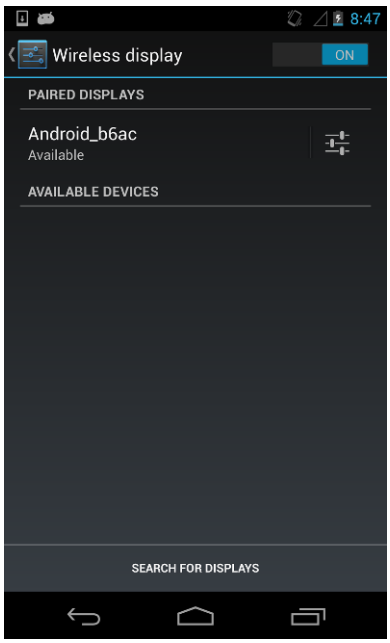
Figure 12-9 Setting the frequency band



After Wi-Fi is enabled, choose **Display** > **Wireless display**. The **Wireless display** screen is displayed, as shown in Figure 12-10. Two device lists are displayed on the **Wireless display** screen.

- **PAIRED DISPLAYS**: list of devices that have been successfully connected.
- **AVAILABLE DEVICES**: list of devices that have not been connected.

Figure 12-10 Wireless display screen



CAUTION

The LG Nexus4 mobile phone is used as an example. The configuration items are similar for other mobile phones or pads.

Step 7 Connect a sink device.

The source end searches for devices. Tap an available device to initiate a connection request.

Wait for a few seconds. Then the sink end displays a message indicating that the device is successfully connected, as shown in Figure 12-11, and screen transferring starts.

Figure 12-11 Device connected

Generally, the connection is established within 10s. However, the time may differ based on network environments. Solutions to poor network conditions:

- If the mobile phone and STB are connected to wireless routers, disconnect them and initiate a Miracast connection request.
- If the HDMI is close to the USB port where the Wi-Fi dongle is connected and interferes with wireless signals, re-connect the Wi-Fi dongle to the USB port by using a USB extension line.
- Use a 5 GHz Wi-Fi dongle.

Solutions to connection failures:

- Re-connection
  - Ensure that the device is available. (Tap the device name on the mobile phone to cancel the connection.)
  - Ensure that the STB is in the **Waiting for connection** state.
- Multiple connection failures
  - On the Wi-Fi Direct screen of the mobile phone, delete group information if any.
  - Restart the mobile phone and STB.
  - Initiate a connection request.

Precautions for initiating a connection request:

- Do not connect multiple mobile phones to an STB at a time. For example, if mobile phone A has been connected to the STB and mirroring has started, mobile phone B cannot connect to the STB.
- Perform Wi-Fi Display connection before Wi-Fi Direct connection, that is, connect to the STB on the Wi-Fi Display screen and then on the Wi-Fi Direct screen.
- Start the Miracast application before you connect to the STB.

Step 8 Transmit audio and video clips over Miracast.

You can transmit audio and video clips after Miracast is successfully connected. The source end packages audio and video information into MPEG2-TS streams and sends them to the STB, as shown in Figure 12-12.

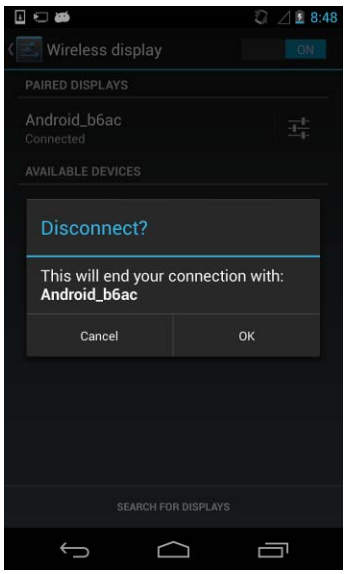
Figure 12-12 Transmitting audio and video clips

Step 9 Disconnect from Miracast.

Disconnect from Miracast by using either of the following methods:

- On the remote control, press the Return key. The Miracast application is disabled.
- On the **Wireless display** screen of the mobile phone, tap the name of the connected device. A dialog box is displayed, asking you whether to disconnect from Miracast, as shown in Figure 12-13. Tap **OK**. Ongoing Miracast services are interrupted.

Figure 12-13 Disconnecting from Miracast



----End

13 Oprofile

NOTE

This chapter takes Hi3798M V100 as an example. The descriptions are similar for other chips.

13.1 Overview

The Oprofile is a Linux-based performance analysis tool for collecting, processing, and analyzing data. It has been integrated on Android. This chapter describes how to use the Oprofile.

The Oprofile collects and analyzes system performance data, including the data generated when the kernel and programs run in various scenarios. It works in either of the following modes:

- Timer mode

  Based on the clock interruption mechanism of the operating system, the Oprofile performs sampling when clock interruption occurs. In timer mode, the program under test cannot shield interrupts during sampling, and the sampling precision is lower than that of event-based sampling.
- Performance counter mode (non-timer mode)

  The performance counter mode is also known as the event-based sampling mode. The internal performance counter of a CPU records the number of times that a certain event occurs, for example, the branch prediction event. When the number reaches the preset value, the Oprofile performs sampling. The performance counter mode provides high precision, and the Oprofile works in this mode by default. Generally, you are advised to select this mode.

13.2 Function Description

13.2.1 Function Details

The Oprofile is used to collect, process, and analyze data on a board.

To use the Oprofile, perform the following steps:

- Collect data on the board.

- Send the data to the compilation host and convert the data.
- Check the data analysis results.

13.2.2 Prerequisites

Prerequisites for using the Oprofile are as follows:

- The Oprofile is installed on the compilation host, allowing you to view data by running the **opreport** command on the Oprofile. For example, to install the Oprofile on Ubuntu, run the following command:

  ```
  # apt-get install oprofile
  ```

- Oprofile configuration items are enabled in the kernel configuration file. To enable the Oprofile configuration items, perform the following steps:

Step 1 Access the directory where the kernel source code is stored:

```
$cd device/hisilicon/bigfish/sdk/source/kernel/linux-3.10.y/
```

Step 2 Select the kernel configuration file, for example, **hi3798mv100_android_defconfig**.

```
$ make ARCH=arm CROSS_COMPILE=arm-hisiv200-linux-
hi3798mv100_android_defconfig
```

Step 3 Access the kernel configuration menu.

```
$ make ARCH=arm CROSS_COMPILE=arm-hisiv200-linux- menuconfig
```

Step 4 Choose **General setup** > **Kernel Performance Events And Counters** and select the following configuration items:

- [*] Profiling support
- <*> OProfile system profiling
- [*] Kernel performance events and counters

Step 5 Compile the kernel and check the compilation results on the board. The following information is displayed when the board starts.

```
oprofile: using arm/armv7-ca9
```

It indicates that the kernel supports the Oprofile and the Oprofile can be used on the board.

----End

13.3 Usage

13.3.1 Connecting to the Target Board

Check the IP address of the target board on the serial port terminal or Android GUI.

For example, check the IP address of the target board on the serial port terminal by running the following command:

```
root@android:/ # busybox ifconfig
```

Figure 13-1 shows the IP address of the target board.

Figure 13-1 IP address of the target board

```
root@android:/ # busybox ifconfig
eth0      Link encap:Ethernet  HWaddr 82:ED:72:1E:90:44
          inet addr:10.67.225.14  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::80ed:72ff:fele:9044/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:457321830 errors:0 dropped:77101 overruns:0 frame:0
          TX packets:212948504 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3253738053 (3.0 GiB)  TX bytes:1388735769 (1.2 GiB)
          Interrupt:103

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@android:/ #
```

To connect to the target board on the compilation host, perform the following steps:

- Check whether the compilation host and target board are connected to the same network. Run the following command on the compilation host:

  ```
  $ ping 10.67.225.14
  ```

- If the IP address is pinged successfully, create an Android debug bridge (ADB) connection between the compilation host and the target board by running the following command:

  ```
  $ adb kill-server && adb connect 10.67.225.14 && adb remount
  ```

Figure 13-2 shows the connection results.

Figure 13-2 ADB connection for the target board

```
[redacted]@Gaea:~/work/V5/master$ adb kill-server && adb connect 10.67.225.14 && adb remount
* daemon not running. starting it now on port 5037 *
* daemon started successfully *
connected to 10.67.225.14:5555
remount succeeded
[redacted]@Gaea:~/work/V5/master$
```

13.3.2 Collecting Data

To collect data on the target board, perform the following steps:

Step 1 Start the target board and run the **opcontrol** command on the serial port terminal to collect data. Create the **vmlinux** file and the system file for the Oprofile in the **data** directory.

- Create the **vmlinux** file:

  ```
  root@android:/ # touch /data/vmlinux
  ```

- Create the system file for the Oprofile:

```
root@android:/ # opcontrol --setup
```

CAUTION

The **oprofile** file in the **/dev** directory disappears after the board restarts. Therefore, perform step 1 each time the board restarts.

When you run the **opcontrol --setup** command, data is cleared from the **data** directory.

Step 2 Set the working mode for the Oprofile and enable the Oprofile background process.

Enable the non-timer mode (performance counter mode). The non-timer mode is enabled by default.

```
root@android:/ # opcontrol --event=CPU_CYCLES:350000 --
vmlinux=/data/vmlinux --kernel-range=0x0,0x0 --verbose
```

You can obtain the **-- kernel-range** parameter by running the following commands:

- Enable the kernel symbol:

```
root@android:/ # echo "0" > /proc/sys/kernel/kptr_restrict
```

- Obtain the **-- kernel-range** parameter:

```
root@android:/ # cat /proc/kallsyms|grep "_text$"
c0008000 T _text
root@android:/ # cat /proc/kallsyms|grep "_etext$"
c0c40e24 A _etext
```

- Run the final commands:

```
root@android:/ # opcontrol --event=CPU_CYCLES:350000 --
vmlinux=/data/vmlinux --kernel-range=0xc0008000,0xc0c40e24 --verbose
```

CAUTION

--event is the flag of the non-timer mode and is mandatory.

Descriptions of the final commands are as follows:

- **CPU_CYCLES**: indicates the detected events. You can view all detected events in non-timer mode by running the **opcontrol --list-event** command.
- **350000**: indicates the number of times that an event sampled by the Oprofile occurs. Generally, the value is larger than 500. The value for the CPU_CYCLES event is larger than that for other events.
- **vmlinux=/vmlinux**: indicates that the Oprofile checks the kernel.
- **kernel-range**: indicates the address range of the kernel checked by the Oprofile.
- **verbose**: indicates that debugging information is displayed.

Enable the timer mode if the non-timer mode is unavailable.

Add the **oprofile.timer=1** field to the kernel startup parameter. If the **oprofile.timer=1** field is not added, the Oprofile works in non-timer mode after the system starts.

```
# opcontrol --timer --vmlinux=/data/vmlinux --kernel-range=0x ,0x  --
verbose
```

Step 3 Collect data.

```
root@android:/ # opcontrol --start --vebose
list_events = 0
setup = 0
Configure /dev/oprofile/enable (1)
```

Step 4 Enable the applications that you want to monitor.

Step 5 Check the data collection status.

```
root@android:/ # opcontrol --status --verbose
list_events = 0
setup = 0
Driver directory: /dev/oprofile
Session directory: /data/oprofile
Counter 0:
    name: CPU_CYCLES
    count: 350000
Counter 1 disabled
Counter 2 disabled
oprofiled pid: 3466
profiler is running
  cpu1        0 samples received
  cpu1        0 samples lost overflow
  cpu1        0 samples invalid eip
  cpu1        0 backtrace aborted
  cpu0   431030 samples received
  cpu0        0 samples lost overflow
  cpu0        0 samples invalid eip
  cpu0        0 backtrace aborted
backtrace_depth: 0
```

Step 6 Stop collecting data.

```
root@android:/ # opcontrol --stop --verbose
list_events = 0
setup = 0
Configure /dev/oprofile/dump (1)
Configure /dev/oprofile/enable (0)
```

Step 7 Clear the collected data as required by running the following command:

```
root@android:/ # opcontrol --reset --verbose
```

```
list_events = 0
setup = 0
```

Step 8 Obtain help information.

```
root@android:/ # opcontrol --help
opcontrol: usage:
   --list-events    list event types
   --help          this message
   --verbose        show extra status
   --verbose-log=lvl set daemon logging verbosity during setup
                  levels are: all,sfile,arcs,samples,module,misc
   --setup          setup directories
   --quick          setup and select CPU_CYCLES:150000
   --timer          timer-based profiling
   --status         show configuration
   --start          start data collection
   --stop           stop data collection
   --reset          clears out data from current session
   --shutdown       kill the oprofile daeman
   --callgraph=depth callgraph depth
   --event=eventspec
      Choose an event. May be specified multiple times.
      eventspec is in the form of name[:count], where :
        name:  event name, see "opcontrol --list-events"
        count: reset counter value
   --vmlinux=file   vmlinux kernel image
   --kernel-range=start,end
                  kernel range vma address in hexadecimal
```

Step 9 Disable the Oprofile background process.

```
root@android:/ # opcontrol --shutdown
WQ on CPU1, prefer CPU0
init: untracked pid 3466 exited
init: untracked pid 3935 exited
```

CAUTION

After you disable the Oprofile background process, the collected data is cleared.

In this case, the **opcontrol --start** and **opcontrol --stop** commands become unavailable. To enable the commands, perform step 2.

The collected data is stored in the following directory:

```
/data/oprofile/samples/
```

----End

13.3.3 Sending and Converting Data

The compiling project is stored in the following directory on the host:

```
/home/xxxx/work/V6/master
```

Compilation environment has been established on the host, overall compilation has been performed once, and the **adb** command is available.

- Set environment variables.
  ```
  xxxx@Gaea:~/work/V6/master$ source build/envsetup.sh
  ```
- Select chip configurations, for example, Hi3798M V100 configurations.
  ```
  xxxx@Gaea:~/work/V6/master$ lunch Hi3798MV100-eng
  ```

To transmit and covert the collected data, perform the following steps:

Step 1 Create and ADB connection between the target board and the host. For details, see section 13.3.1 "Connecting to the Target Board"

Step 2 Transmit and convert data by running the python script (**opimport_pull**) in the source code directory of the Oprofile.

1. Access the source code directory of the Oprofile.
   ```
   $ cd /home/xxxx/work/V6/master/external/oprofile/
   ```
2. Transmit and convert data by running the **python** script and check the results.
   ```
   $ python opimport_pull -r result
   ```

Obtain help information by running the **opimport_pull** script.

```
$ python opimport_pull
Usage:opimport_pull [-s serial_number] [-r] dir
    serial_number: the device being profiled
    -r : reuse the directory if it already exists
    dir: directory on the host to store profile results
```

----End

13.4 Viewing Data Analysis Results

To view the data analysis results, perform the following steps:

Step 1 Access the directory where the Oprofile data analysis results are stored.

```
$ cd /home/xxxx/work/V6/master/external/oprofile/
```

Step 2 Check the data analysis results.

```
$ opreport --session-dir=result/
```

Figure 13-3 shows part of the data analysis results.

Figure 13-3 Data analysis results of the Oprofile

```
[redacted]@Gaea:~/work/V5/master/external/oprofile$ opreport --session-dir=result/
CPU: ARM Cortex-A9, speed 0 MHz (estimated)
Counted CPU_CYCLES events (Number of CPU cycles) with a unit mask of 0x00 (No unit mask) count 350000
CPU_CYCLES:350000|
  samples|      %|
------------------
    17685 46.2027 vmlinux
     8139 21.2634 libc.so
     6151 16.0697 libdvm.so
     1503  3.9266 libMali.so
      833  2.1762 libskia.so
      696  1.8183 libutils.so
      635  1.6590 dalvik-jit-code-cache
      386  1.0084 libbinder.so
      301  0.7864 libhwui.so
      287  0.7498 libgui.so
      286  0.7472 libsurfaceflinger.so
      277  0.7237 libui.so
      189  0.4938 libandroid_runtime.so
      136  0.3553 libz.so
      104  0.2717 libcutils.so
       81  0.2116 libaudioflinger.so
       78  0.2038 oprofiled
       61  0.1594 libEGL.so
       43  0.1123 libandroidfw.so
       36  0.0941 libicuuc.so
       35  0.0914 libsqlite.so
       28  0.0732 gralloc.bigfish.so
       28  0.0732 libharfbuzz.so
       28  0.0732 libm.so
       27  0.0705 libaudioutils.so
       25  0.0653 libGLESv1_CM_mali.so
       24  0.0627 libinput.so
       20  0.0523 libjavacore.so
       18  0.0470 linker
       12  0.0314 libGLESv1_CM.so
       10  0.0261 libEGL mali.so
```

Obtain help information about the opreport tool.

```
$ man opreport
```

----End

13.5 Instance

13.5.1 Viewing Media Data Analysis Results

Prerequisites for viewing media data analysis results are as follows: Image compilation has been completed on the host. Compilation environment variables have been configured. The compiling project is stored in the following directory by default:

```
/home/xxxx/work/V6/master
```

To view media data analysis results, perform the following steps:

Step 1 Start the target board and its serial port terminal. Create a system file for the Oprofile. Run the following commands on the serial port terminal:

```
root@android:/ # touch /data/vmlinux
root@android:/ # opcontrol --setup
```

Step 2 Enable the non-timer mode and the Oprofile background process. Run the following commands on the serial port terminal:

```
root@android:/ # echo "0" >/proc/sys/kernel/kptr_restrict
root@android:/ # cat /proc/kallsyms|grep "_text$"
root@android:/ # cat /proc/kallsyms|grep "_etext$"
```

Figure 13-4 shows the execution results.

Figure 13-4 Enabling the Oprofile background process

```
root@android:/ # cat /proc/kallsyms|grep "_text$"
c0008000 T _text
c0054afc T core_kernel_text
c0054bb8 T func_ptr_is_kernel_text
c07f0b94 T skb_find_text
c097c704 r err_text
c0980980 R vmstat_text
c09ca754 r mode_text
c0bf67c8 r __ksymtab_skb_find_text
c0c11d32 r __kstrtab_skb_find_text
root@android:/ #
root@android:/ # cat /proc/kallsyms|grep "_etext$"
c0caecd4 A _etext
root@android:/ #
```

Run the following general commands:

```
root@android:/ # opcontrol --event=CPU_CYCLES:350000 --
vmlinux=/data/vmlinux --kernel-range=0xc0008000,0xc0caecd4 --verbose
```

Figure 13-5 shows the execution results.

Figure 13-5 Enabling the Oprofile background process

```
root@android:/ #
xc0008000,0xc0c2de24 --verbose
list_events = 0
setup = 0
Configure /dev/oprofile/0/user (1)
Configure /dev/oprofile/0/kernel (1)
Configure /dev/oprofile/0/unit_mask (0)
Configure /dev/oprofile/0/enabled (1)
Configure /dev/oprofile/0/count (350000)
init: untracked pid 3246 exited
Configure /dev/oprofile/0/event (255)
Configure /dev/oprofile/1/enabled (0)
Configure /dev/oprofile/2/enabled (0)
Starting oprofiled...
command: oprofiled --session-dir=/data/oprofile --events=CPU_CYCLES:255:0:350000:0:1:1 -k /data/vmlinux -r 0xc0008000,0xc0c2de24
Using 2.6+ OProfile kernel interface.
Reading module info.
Using log file /data/oprofile/samples/oprofiled.log
Ready
root@android:/ #
```

Step 3 Collect data by running the following command on the serial port terminal:

```
root@android:/ # opcontrol --start
```

Step 4 Play a media source on the GUI.

Step 5 Check the data collection status by running the following command on the serial port terminal:

```
root@android:/ # opcontrol --status
```

Figure 13-6 shows the execution results.

Figure 13-6 Data collection status

```
root@android:/ # opcontrol --status
Driver directory: /dev/oprofile
Session directory: /data/oprofile
Counter 0:
    name: CPU_CYCLES
    count: 350000
Counter 1 disabled
Counter 2 disabled
oprofiled pid: 2947
profiler is running
  cpu1          0 samples received
  cpu1          0 samples lost overflow
  cpu1          0 samples invalid eip
  cpu1          0 backtrace aborted
  cpu0     186458 samples received
  cpu0          0 samples lost overflow
  cpu0          0 samples invalid eip
  cpu0          0 backtrace aborted
backtrace_depth: 0
root@android:/ # _
```

Step 6 Stop collecting data.

```
root@android:/ # opcontrol --stop
```

Step 7 Check the IP address of the target board by running the corresponding command on the serial port terminal, as shown in Figure 13-7.

Figure 13-7 IP address of the target board

```
root@android:/ # busybox ifconfig
eth0      Link encap:Ethernet  HWaddr 82:ED:72:1E:90:44
          inet addr:10.67.225.14  Bcast:10.255.255.255  Mask:255.0.0.0
          inet6 addr: fe80::80ed:72ff:fele:9044/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1073829 errors:0 dropped:401 overruns:0 frame:0
          TX packets:533208 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1599954046 (1.4 GiB)  TX bytes:36212070 (34.5 MiB)
          Interrupt:103

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

root@android:/ #
```

Step 8 Connect the host to the target board by running the following command in the host shell environment (by default, the environment has been established, and the ADB tool is available):

```
xxxx@Gaea:~/work/V6/master$ adb kill-server && adb connect 10.67.225.14
&& adb remount
```

Step 9 Transmit and convert data and check the preliminary data analysis results by running the following commands:

```
xxxx@Gaea:~/work/V6/master$ cd external/oprofile/
xxxx@Gaea:~/work/V6/master/external/oprofile$ python opimport_pull -r
result/
```

Figure 13-8 shows part of the execution results.

Figure 13-8 Data analysis results of the Oprofile (1)

```
[redacted]@Gaea:~/work/V5/master/external/oprofile$ python opimport_pull -r result/
CPU: ARM Cortex-A9, speed 0 MHz (estimated)
Counted CPU_CYCLES events (Number of CPU cycles) with a unit mask of 0x00 (No unit mask) count 350000
CPU_CYCLES:350000|
  samples|      %|
------------------
   404972 89.1667 vmlinux
    15008  3.3045 libc.so
    11672  2.5699 libavcodec.so
     4289  0.9444 libdvm.so
     2985  0.6572 libhi_avplay.so
     2358  0.5192 libplayer.so
     2128  0.4685 libhimediaplayer_jni.so
     1261  0.2776 libhi_adec.so
     1191  0.2622 libavformat.so
     1164  0.2563 libm.so
     1033  0.2274 libffmpegformat.so
      874  0.1924 libhi_vdec.so
      785  0.1728 libskia.so
      779  0.1715 oprofiled
      765  0.1684 libhi_sync.so
      580  0.1277 dalvik-jit-code-cache
      266  0.0586 libMali.so
      242  0.0533 libavutil.so
      178  0.0392 libutils.so
      172  0.0379 linker
      155  0.0341 libz.so
      145  0.0319 libhi_so.so
      139  0.0306 libHA.AUDIO.FFMPEG_ADEC.decode.so
```

Step 10 Check the data analysis results by running the following commands:

```
xxxx@Gaea:~/work/V5/master/external/oprofile$ opreport --session-dir=result/ -p /home/xxxx/work/V5/master/out/target/product/Hi3798MV100/symbols/ -l
```

Figure 13-9 shows part of the data analysis results.

Figure 13-9 Data analysis results of the Oprofile (2)

```
[redacted]@Gaea:~/work/V5/master/external/oprofile$ opreport --session-dir=result/ -p /home/[redacted]/work/V5/master/out/target/product/Hi3716CV200ES/symbols/ -l
warning: /data/vmlinux could not be found.
warning: /dev/ashmem/dalvik-jit-code-cache could not be found.
CPU: ARM Cortex-A9, speed 0 MHz (estimated)
Counted CPU_CYCLES events (Number of CPU cycles) with a unit mask of 0x00 (No unit mask) count 350000
samples  %        app name                 symbol name
404972   89.5871  vmlinux                  /data/vmlinux
3954      0.8747  libc.so                  memcpy
3127      0.6917  libavcodec.so            vorbis_decode_frame
2750      0.6083  libc.so                  memset
1561      0.3453  libavcodec.so            float_to_int16_interleave_c
1353      0.2993  libavcodec.so            pass
1330      0.2942  libavcodec.so            ff_imdct_half_c
1186      0.2624  libhimediaplayer_jni.so  android::SubtitleFontManager::loadBmpToBuffer(int)
930       0.2057  libc.so                  dlfree
842       0.1863  libavcodec.so            vector_fmul_window_c
819       0.1812  libm.so                  rintf
727       0.1608  libc.so                  pthread_mutex_lock_impl
630       0.1394  libc.so                  bcopy
586       0.1296  libdvm.so                markObjectNonNull(Object const*, GcMarkContext*, bool)
580       0.1283  dalvik-jit-code-cache    /dev/ashmem/dalvik-jit-code-cache
561       0.1241  libavcodec.so            vorbis_floor1_decode
556       0.1230  libavcodec.so            vorbis_inverse_coupling
556       0.1230  libc.so                  strstr
550       0.1217  libc.so                  pthread_mutex_unlock_impl
540       0.1195  libplayer.so             _SVR_PCTRL_MainFunction
505       0.1117  libavcodec.so            vector_fmul_c
464       0.1026  libdvm.so                dalvik_inst
460       0.1018  libavcodec.so            render_line
457       0.1011  libdvm.so                scanObject(Object const*, GcMarkContext*)
427       0.0945  libc.so                  __ioctl
418       0.0925  libc.so                  dlmalloc
```

----End

14 ADB

14.1 Overview

The ADB is a tool in the Android SDK. By using the ADB, you can operate and manage an Android emulator or an Android device. The ADB provides the following functions:

- Runs the shell (command line) of a device.
- Manages port mapping of an emulator or device.
- Enables a computer and device to upload/download files to/from each other.
- Installs the local .apk software on an emulator or Android device.

The ADB is a client-to-server program. The client refers to a computer, and the server refers to an entity or virtual machine of an Android device. The ADB works in a special mode. It enables communication between the IDE and a device by listening on Socket and TCP ports.

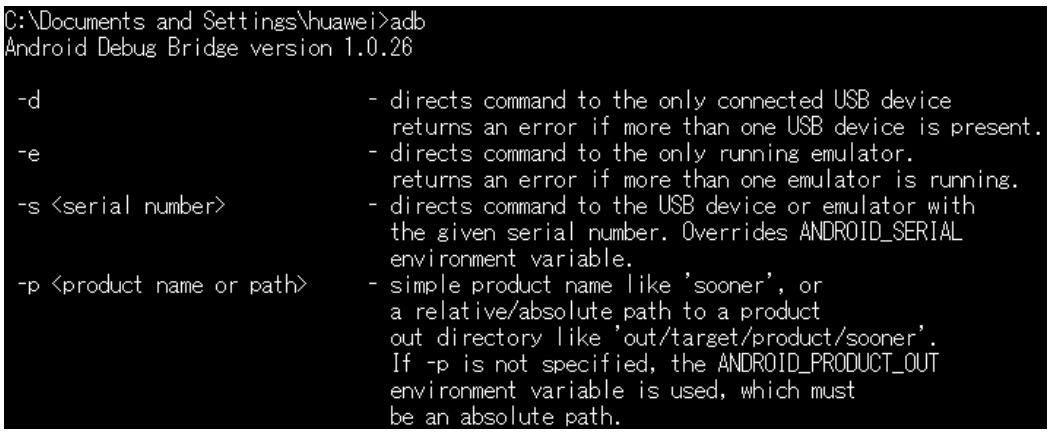
14.2 Installation and Debugging

14.2.1 Installing the ADB on Windows

The required software includes the ADB toolkit or entire SDK and ADB operation software.

- If the entire SDK is obtained, the ADB is automatically installed when the Android environment is set up.
- If the ADB toolkit is obtained, you only need to copy **adb.exe** and **AdbWinApi.dll** to **C:\Windows\System32\.** To check whether the ADB is installed successfully:
  - Type **cmd** in the search box and press **Enter** to access the DOS screen.
  - Type **adb** in the DOS screen. The information shown in Figure 14-1 is displayed, indicating that the upgrade is successful.

Figure 14-1 Successful ADB installation



14.2.2 Installing the ADB on Linux

The software required includes Android SDK Linux. The ADB is installed together with the Android environment constructed. For the method of constructing Android environment on Linux, see the *Android SDK Developer Guide*.

14.2.3 Connecting the ADB to the Android System of a Board

To connect the ADB to the Android system of a board, perform the following steps:

Step 1 Check whether the board runs Android system. If the board runs a non-Android system, burn the Android system to the board by using the fastboot. For details about how to use the fastboot, see the *Fastboot Burning Tool Application Notes*.

Step 2 After confirming that both the board and computer are connected to the Internet, obtain a network IP address for the board. After that, run the following command on the computer to make it connect to the board. (See Figure 14-2.)

```
adb kill-server
adb connect IP address of the board: port number (5555 by default)
adb remount
```

Step 3 Run **adb devices** to check whether the connection is successful. If the IP address of the board is displayed, the connection is successful.

----End

NOTE

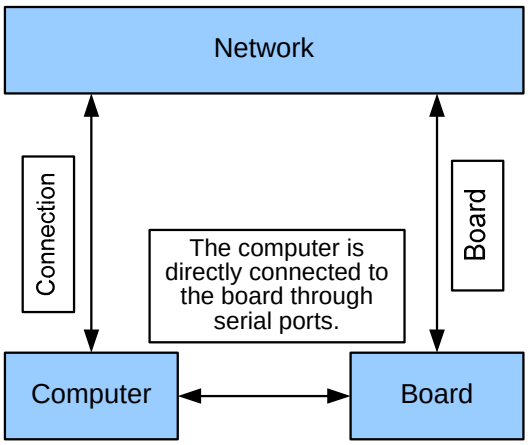
The following problem is detected during use of the Ubuntu: Connected Android devices cannot be detected by running **adb devices**. This problem occurs because a server daemon is started for interaction with Android devices when the **adb** command is executed for the first time. This daemon cannot access devices due to the permission limits of Ubuntu. To resolve this problem, you can run **adb start-server**. Note that you need to add the **adb** directory to the root environment variables or use the full path.

```
adb start-server
```

If the command output shows that the daemon is successfully started, run the **adb** command to use it. To disable the daemon, run the following command:

```
adb kill-server
```

Figure 14-2 Connection between the board and computer



14.3 Common Commands

14.3.1 Checking the Status of a Device

Check the Android device or emulator connected to the computer:

```
adb devices
```

The IP address, port number, and status of the Android device are displayed in the command output.

- The IP address and port number of the Android device are displayed in the following format:

  `IP:Port`

- Two types of status are available:

  offline——The device is not connected to the development machine or fails to respond.

  device——The device is already connected to the development machine.

NOTE

The device status does not indicate that the Android device is available. This status is also returned when an Android device in the startup phase is successfully connected to the development machine. After an Android device is connected to the development machine, you can disconnect the Android device from the network or shut down it and then run **adb devices** to display the last Android device connected to the development machine.

Figure 14-3 Checking device conditions

```
C:\Documents and Settings\huawei>adb devices
List of devices attached
10.85.185.109:5555      device
```

CAUTION

A common command can be successfully executed only when an Android device successfully connects to the development machine that has the ADB installed.

14.3.2 Installing Software

- Install a specified .apk file to a device:

  Command: `adb install <apk file path>`

  Example: `adb install "F:\WishTV\WishTV.apk"`

  Description: Install the WishTV software to a device. See Figure 14-4.
- If the information shown in Figure 14-5 is displayed, you can reinstall the software:

  Command: `adb install -r <apk file path>`

  Example: `adb install -r "F:\WishTV\WishTV.apk"`

  Description: Reinstall the WishTV software. See Figure 14-6.

Figure 14-4 Installing the .apk file to an Android device successfully

```
C:\Documents and Settings\huawei>adb install  "F:\WishTV\WishTV.apk"
2587 KB/s (579629 bytes in 0.218s)
        pkg: /data/local/tmp/WishTV.apk
Success
```

Figure 14-5 Failed to install the .apk file to an Android device

```
C:\Documents and Settings\huawei>adb install  "F:\WishTV\WishTV.apk"
2786 KB/s (579629 bytes in 0.203s)
        pkg: /data/local/tmp/WishTV.apk
Failure [INSTALL_FAILED_ALREADY_EXISTS]
```

Figure 14-6 Reinstalling software

```
C:\Documents and Settings\huawei>adb install -r "F:\WishTV\WishTV.apk"
2587 KB/s (579629 bytes in 0.218s)
        pkg: /data/local/tmp/WishTV.apk
Success
```

14.3.3 Uninstalling Software

Two software uninstallation methods are available:

- Run the **uninstall** command to uninstall software (if the software is installed by running the **install** command).
  - Complete uninstallation:

    Command: `adb uninstall <package>`

Example: `adb uninstall com.wishtv`

Description: Uninstall the WishTV software completely. See Figure 14-7.

- Incomplete uninstallation (with the configurations and cache files retained)

  Command: `adb uninstall -k <package>`

  Example: `adb uninstall -k com.wishtv`

  Description: Retain the configurations and cache files of the WishTV software when uninstalling it. See Figure 14-8.

- Run the **rm** command to remove an .apk file:

  Command: `adb shell rm <filepath>`

  Example: `adb shell rm "system/app/WishTV.apk"`

  Description: Remove the **WishTV.apk** file from the **system/app** directory. See Figure 14-9.

NOTE

If you cannot determine which command is used for installing the software, try to run the **uninstall** command for uninstallation. If the command fails to be run, run the **rm** command to remove the software.

You can use the File Explorer function of the DDMS tool to determine the package corresponding to the .apk file. See Figure 14-10. For the method of using the DDMS, see chapter 15 "DDMS."

When you run the **rm** command to remove the software, software-related packages and cache files that may hinder successful installation of the software next time may be left over in the system. It is recommended that you run the **install** and **uninstall** commands to install and uninstall software respectively because the **uninstall** command processes software-related files according to command parameters, eliminating affects on next software installation.

Figure 14-7 Complete software uninstallation

```
C:\Documents and Settings\huawei>adb uninstall com.wishtv
Success
```

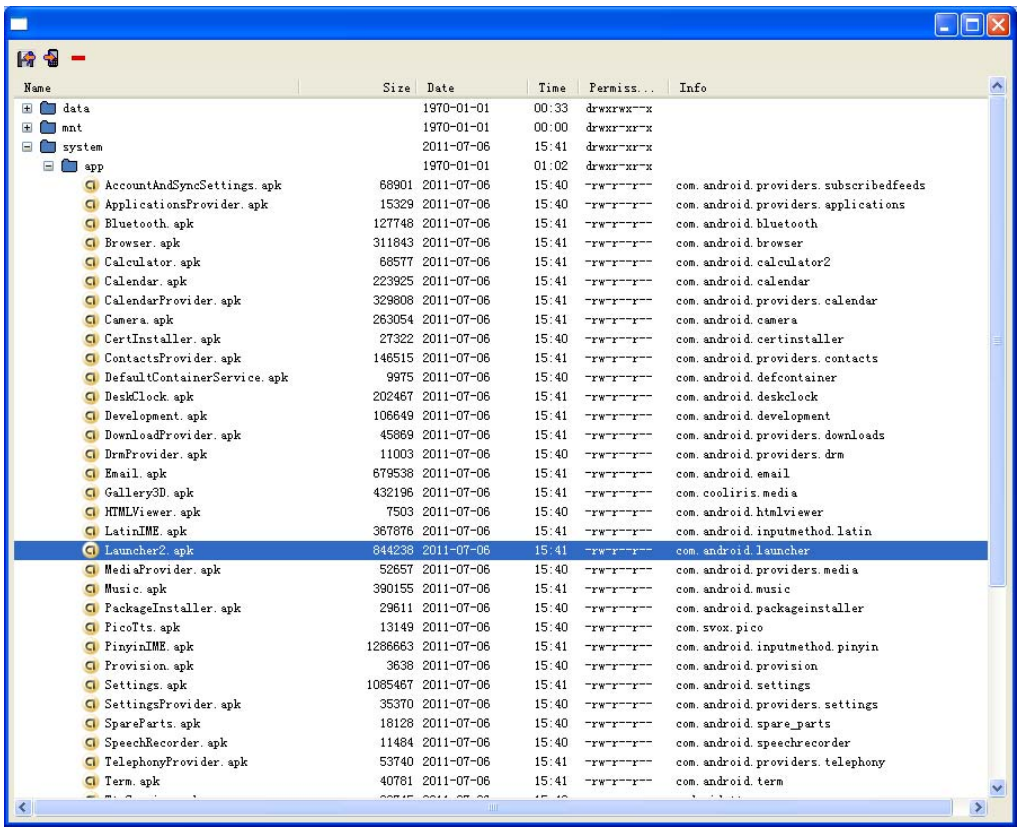
Figure 14-8 Incomplete software uninstallation

```
C:\Documents and Settings\huawei>adb uninstall -k  com.wishtv
The -k option uninstalls the application while retaining the data/cache.
At the moment, there is no way to remove the remaining data.
You will have to reinstall the application with the same signature, and fully uninstall it.
If you truly wish to continue, execute 'adb shell pm uninstall -k com.wishtv'
```

Figure 14-9 Removing the software from the system

```
C:\Documents and Settings\huawei>adb shell
# rm system/app/WishTV.apk
rm system/app/WishTV.apk
# exit
exit
```

Figure 14-10 Reading system files



14.3.4 Entering Shell of a Device or Emulator

Enter the shell environment of a device or emulator:

```
adb shell
```

You can run various Linux commands on Linux Shell, as shown in Figure 14-11. If you only want to run one command, use the following method:

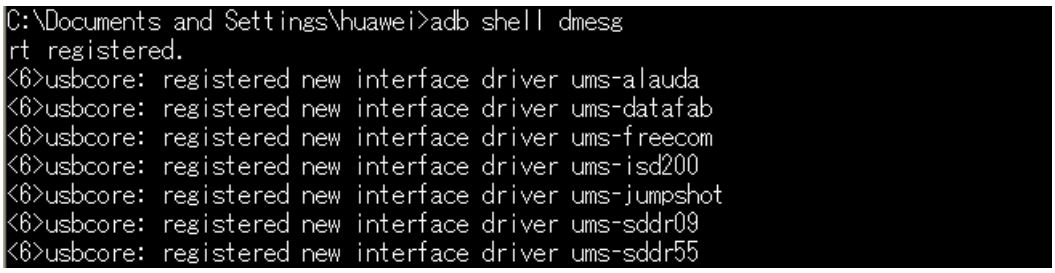
```
Command: adb shell [command]
Example: adb shell dmesg
```

Description: Print the debugging information of the kernel. See Figure 14-12.

Figure 14-11 Entering the shell environment of a device or emulator

```
C:\Documents and Settings\huawei>adb shell
# cd data/app
cd data/app
# exit
exit
```

Figure 14-12 Displaying kernel information



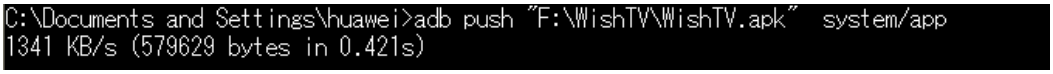
14.3.5 Uploading Files on the Local Computer to a Device

You can run the **push** command to upload any files or folders on the local computer to a device. Generally, the local path refers to the local computer and remote path refers to the board connected to the ADB.

```
Command: adb push <local path> <remote path>
Example: adb push "F:\WishTV\WishTV.apk" "system/app"
```

Description: Upload the local **WishTV.apk** file to the **system/app** directory of an Android device. See Figure 14-13.

Figure 14-13 Copying files to an Android device



CAUTION

When uploading files to a board, you must enable the write permission of the target path file system. Otherwise, the uploading will fail.

NOTE

A local path uses back slashes (\) and the directory of an Android device uses forward slashes (/).

14.3.6 Downloading Files on a Device to the Local Computer

You can run the **pull** command to download files or folders on a device to the local computer.

Command: `adb pull <remote path> <local path>`

Example 1: adb pull system/app/Contacts.apk F:\

Description: Download the files or folders in the **system/app** directory of an Android device to the **F:** path of the local computer. See Figure 14-14.

Figure 14-14 Downloading a single file on an Android device to a local path

Example: `adb pull system/app F:\`

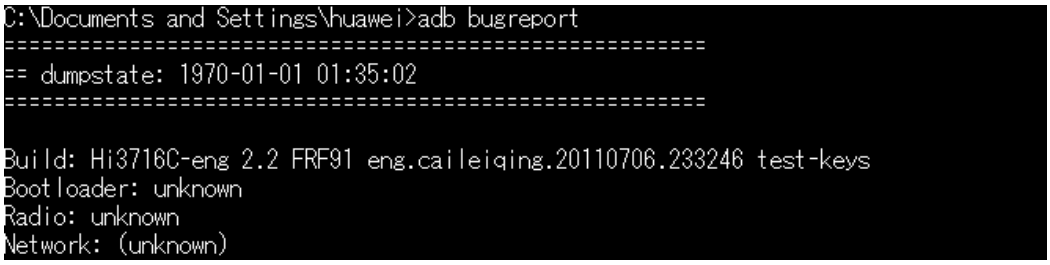
Description: Download a file or folder in the **system/app** directory of an Android device to the **F:** path of the local computer. See Figure 14-15.

Figure 14-15 Downloading files on an Android device to a local path

14.3.7 Viewing Bug Reports

To view all error reports generated by an Android system, you can run the **adb bugreport** command to display the dumpsys, dumpstate, and logcat information of the Android system. See Figure 14-16.

Figure 14-16 Viewing bug reports of a device



14.3.8 Viewing System Information of a Device

Figure 14-17 shows the specific command for viewing system information of a device in adb shell.

```
adb shell getprop
```

Figure 14-17 Viewing system information

14.3.9 Using the Monkey Program

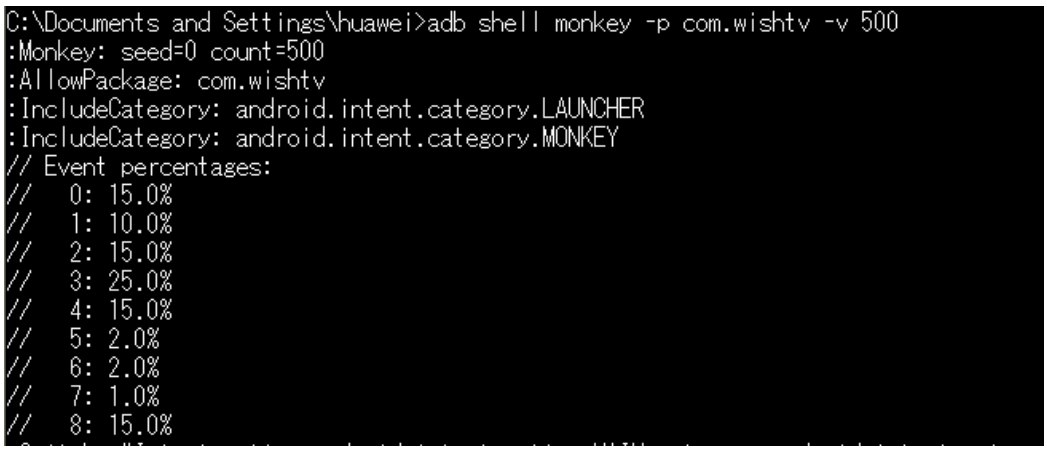
The Monkey program can simulate a series of user events on an emulator or device at random, for example, click and slide. Therefore, the Monkey program can be used to conduct load test on applications repeatedly at random.

Command: `adb shell monkey [options] <event-count>`

Example: `adb shell monkey -p com.wishtv -v 500` (for description of command parameters, see Table A-4)

Description: Start your software and trigger 500 events.

Figure 14-18 Repeated and random load test by using the Monkey program



14.3.10 Enabling Logcat Logs

The Android log system records the system debugging information and allows you to view it. The logs output by various software and system modules are stored in a cyclic buffer. To view and use the logs in this buffer, run the **logcat** command. For details, see section 16.3 "Logcat Commands."

14.4 USB ADB

14.4.1 Overview

The ADB is divided into two types based on the mode for connecting the PC and the board:

- Network ADB: The PC connects to the board over a wired or wireless network.
- USB ADB: The PC connects to the board by using a USB cable.

Note the following when using the USB ADB:

- Only the USB0 port is supported.
- Only the USB 2.0 protocol is supported.
- You cannot run multiple clients (such as the CLI, Eclipse, and Peasecod) at a time.
- The PC can connect to only one device.

14.4.2 Usage

The ADB can be used in the following scenarios:

- The PC connects to the board over the network but not the USB port.

  In this case, only the network ADB is enabled, and the ADB can be used only after it connects to the board IP address. The USB ADB is unavailable.
- The PC connects to USB0 of the board by using only the USB cable.

  The USB ADB driver provided by HiSilicon must be installed on the PC. Currently the driver supports only Windows 7 and Windows XP. For details about the installation, see the *Installation Guide for the USB ADB PC Driver*.

  In this case, only the USB ADB is enabled and can be directly used. The network ADB is unavailable.
- The PC connects to the board by using both the network and USB cable.

  In this case, the USB ADB is enabled by default and can be used directly.

  However, the network ADB can be used only after the ADB connects to the board IP address.

14.4.2.1 Using only the USB ADB

To use only the USB ADB, perform the following steps:

Step 1 Choose **Settings** > **Developer options**, and ensure that **USB debugging** is selected. For Android 4.2, the **Developer options** option is not displayed by default in the user version. In the eng version, if you want to hide the **Developer options** option, you need to modify the code in the following file:

packages/apps/Settings/src/com/android/settings/Settings.java

Before modification:

```
if (!showDev) {
```

After modification:

```
if (!showDev || true) {
```

Step 2 Ensure that the PC is not connected to the board by using the ADB command for connecting to the board IP address.

Step 3 Open the command interface of the PC, and enter **adb devices**. The USB ADB device is displayed, as shown in Figure 14-19.

Figure 14-19 Using the USB ADB

Then you can run the ADB commands.

----End

14.4.2.2 Using the USB ADB and Network ADB

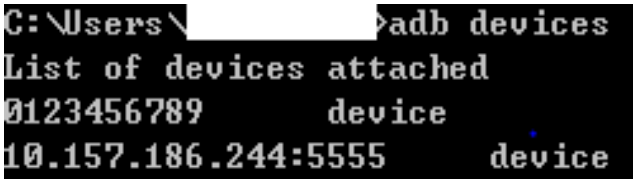
To use the USB ADB and network ADB at the same time, perform the following steps:

Step 1 Choose **Settings** > **Developer options**, and ensure that **USB debugging** is selected.

Step 2 Ensure that the PC is connected to the board by using the ADB command for connecting to the board IP address (**adb connect** *board IP address*).

Step 3 Open the command interface of the PC, and enter **adb devices**. If two devices are displayed, as shown in Figure 14-20, both the USB ADB and network ADB are connected successfully.

Figure 14-20 Using both the USB ADB and network ADB



- **0123456789** is the name of the board corresponding to the USB ADB.
- **10.157.186.244:5555** is the name of the board corresponding to the network ADB.

  If you need to run the ADB commands at this time, you must specify the device name in the ADB command, for example:
  - For the network ADB, run **adb –s 10.157.186.244:5555** ***xxx***.

    xxx indicates the adb shell command to be executed.

    For example, **adb –s 10.157.186.244:5555 shell**.
  - For the USB ADB, run **adb –s 0123456789** ***xxx***.

    xxx indicates the adb shell command to be executed.

    For example, **adb –s 0123456789 shell**.

If you want to use only the network ADB, remove the USB extension cable. Then you can directly enter **adb *xxx*** (*xxx* is the adb shell command to be executed) to use the ADB, for example, **adb shell**.

If you want to use only the USB ADB, enter **adb disconnect** *board IP address* to disable the network ADB. Then you can directly enter **adb *xxx*** (*xxx* is the adb shell command to be executed) to use the ADB, for example, **adb shell**.

----End

15 DDMS

15.1 How to Start the DDMS

15.1.1 Introduction

The DDMS is a debugging and monitoring service for the Dalvik virtual machine in the Android development environment. The DDMS provides the following functions:

- Takes snapshots of the tested device
- Checks the running threads and displays the heap information based on specific processes
- Displays the logcat information
- Displays the broadcasting status
- Simulates calling
- Receives short messages
- Displays virtual coordinates

15.1.2 Starting the DDMS

The DDMS is stored in **android-sdk-windows\tools**. You can start the DDMS in the following ways:

Double-click **ddms.bat** (as shown in Figure 15-1). The DDMS window shown in Figure 15-6 is displayed.

Figure 15-1 DDMS shortcut

To start the DDMS when the Eclipse debugging program is running, perform the following steps:

Step 1 Click  in the upper right corner of Eclipse. A drop-down list is displayed, as shown in Figure 15-2.

Step 2 Choose **Other** from the drop-down list. A list of all Eclipse tools is displayed, as shown in Figure 15-3.

Step 3 Select **DDMS** and click **OK**. The DDMS debugging tool is displayed, as shown in Figure 15-4.

----End

Figure 15-2 Menu of some Eclipse tools

Figure 15-3 List of all Eclipse tools

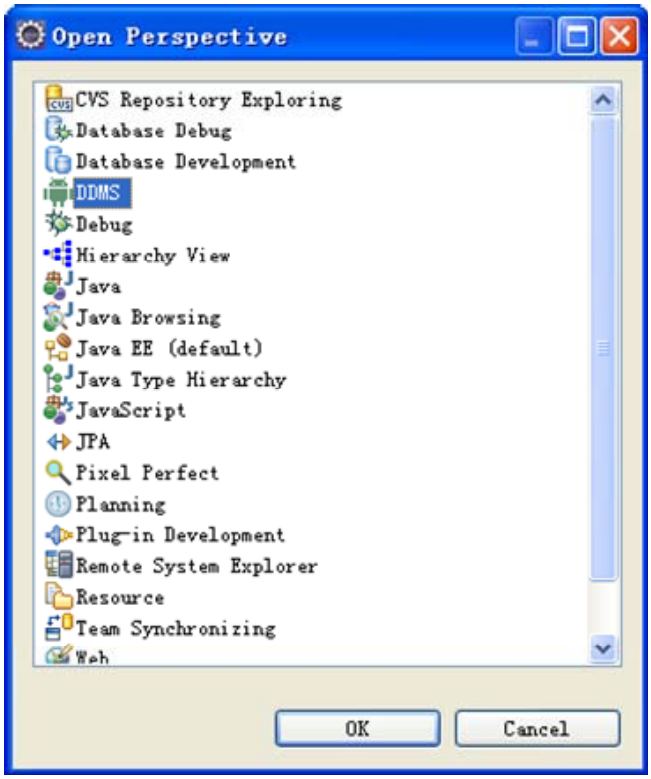
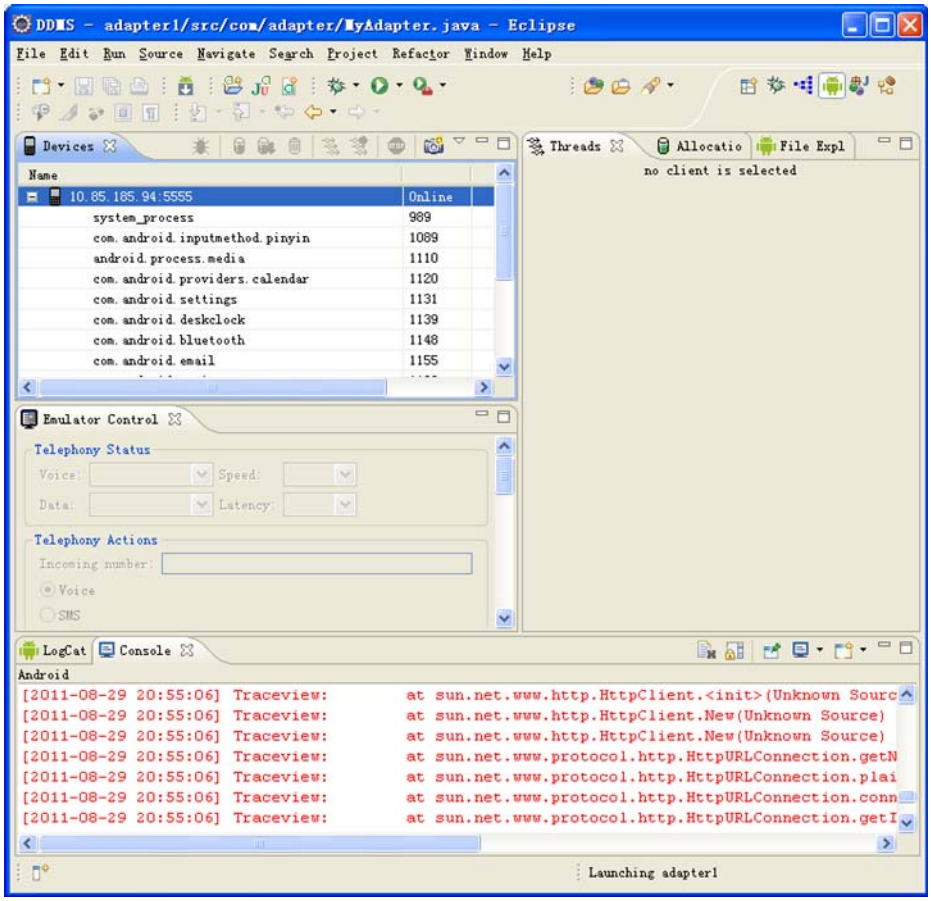


Figure 15-4 DDMS debugging tool



To start the DDMS through the CMD window, perform the following steps:

Step 1 Choose **Start** > **Run** or press **Windows key**+**R**.

Step 2 Type **cmd** and press **Enter**.

The CMD window shown in Figure 15-5 is displayed.

Step 3 Type **ddms** and press **Enter**.

The DDMS window shown in Figure 15-6 is displayed.

----End

Figure 15-5 CMD window

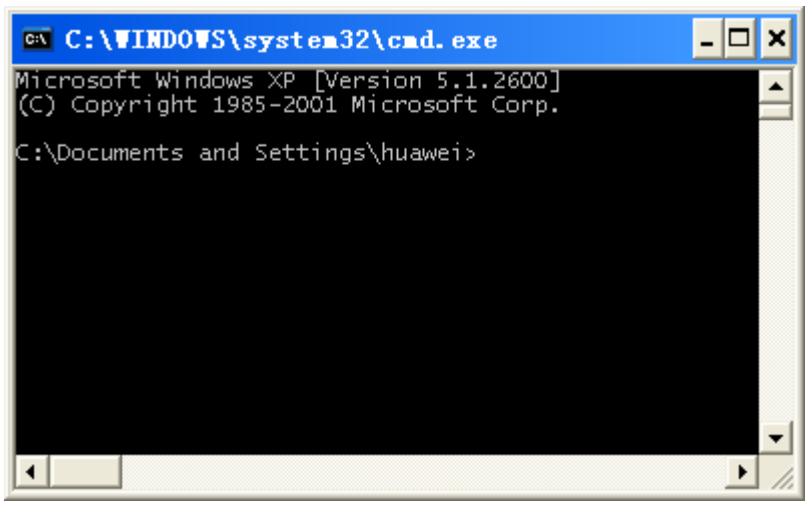
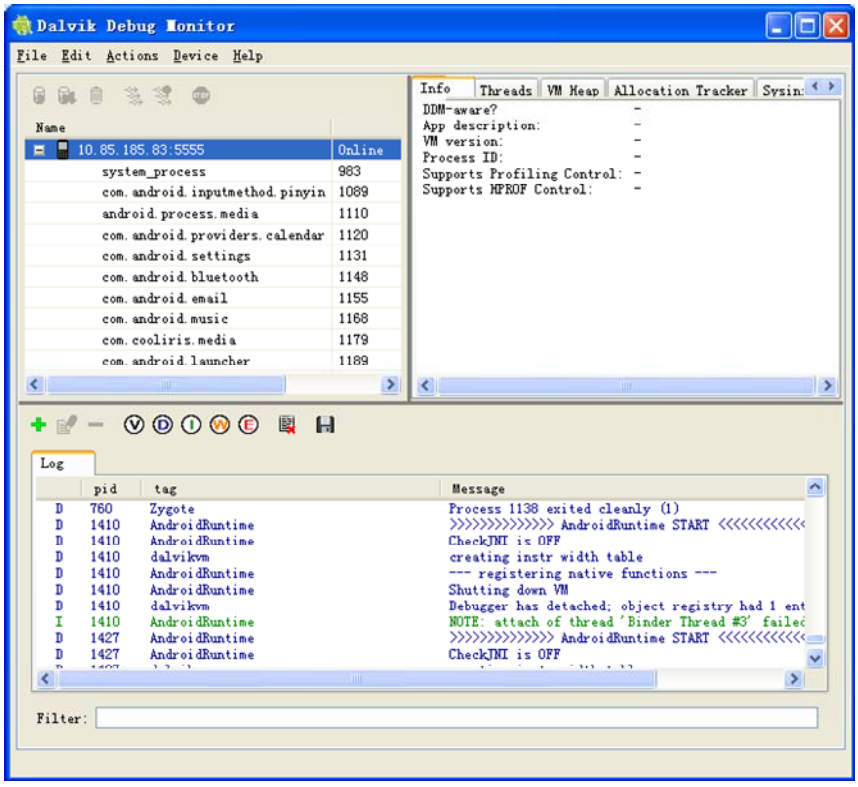


Figure 15-6 DDMS window displayed when the DDMS is started directly



CAUTION

Ensure that the ADB is connected to the device before using the DDMS. You can run **adb devices** to check whether they are connected.

When the DDMS is started in different ways, the displayed DDMS windows differ in terms of layout and function. For details, see Figure 15-4 and Figure 15-6.

15.2 DDMS Operating Principles

The operating principles of the DDMS are as follows:

- The DDMS is used to connect the IDE to a test terminal (emulator or connected device). The IDE and test terminal monitor the debugging information over their own ports. The DDMS monitors the connection status of the test terminal in real time. When a new test terminal is connected, the DDMS captures its ID and establishes a debugger by using the ADB. This ensures that commands are sent to the new test terminal.
- The DDMS monitors the application (APP) of the first terminal over port 8600, and the port ID allocated to the APP is 8601. If there are multiple terminals and APPs, the port IDs are allocated in this way. The DDMS receives the commands from all terminals over port 8700.
- Each Android APP runs in a Dalvik virtual machine instance, and each instance is a separate process space. For a virtual machine, its thread mechanism, memory allocation and management, and mutual exclusive (Mutex) rely on the operating system at the bottom layer. The mutex is also called synchronization primitive. Each Android APP thread corresponds to a Linux thread. Therefore, the virtual machine can rely on the thread scheduling and management mechanism of the operating system.
- When the DDMS is started, the device monitoring service is established between the DDMS and ADB to monitor devices. When a device is connected or disconnected, the service notifies the DDMS.
- When a device is connected, the VM monitoring service is established between the DDMS and ADB to monitor the virtual machine on the device.
- After the DDMS connects to the debugger on the virtual machine by using the ADB daemon, the DDMS can communicate with the virtual machine.

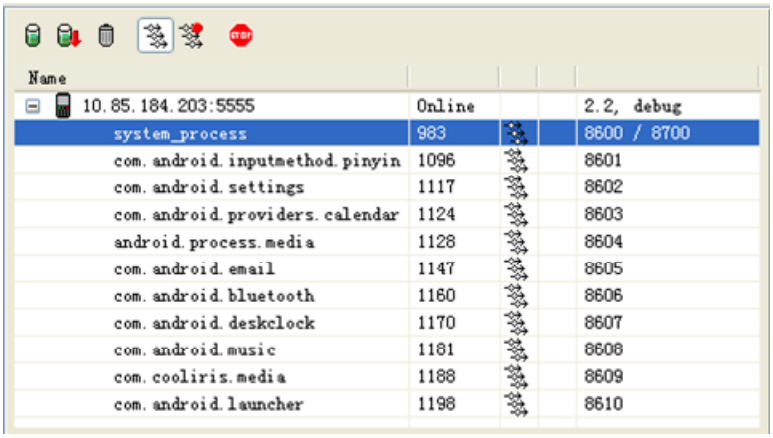
15.3 DDMS Function Descriptions

This section describes the DDMS functions by using the DDMS window shown in Figure 15-6 as an example. As shown in Figure 15-6, the DDMS window is divided into three areas.

15.3.1 Area Functions

15.3.1.1 Area 1

Figure 15-7 Area 1 of the DDMS window



- The descriptions of the icons in Figure 15-7 are as follows:
  - : displays the updates of the memory or heap information about a running process or APP.
  - : downloads the memory or heap information about a running process or APP to a file.
  - : collects garbage in real time.
  - : displays the updates of the APP threads.
  - : records the frequently-used functions of APPs.
  - : pauses the virtual machine.
- The descriptions of other contents in Figure 15-7 are as follows:
  - Root directory: device list (including the device IP address and port ID), that is, 10.85.184.203:5555.
  - The descriptions of the contents in the subdirectory under the device list are as follows (from left to right):

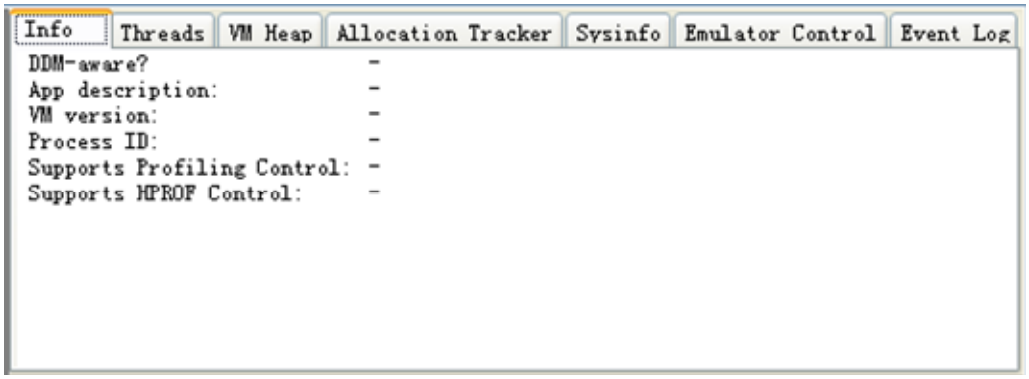
    First column: indicates the list of running processes or APPs.

    Second column: indicates the IDs of running processes or APPs.

    Fourth column: indicates the debug windows for the running processes or APPs.

15.3.1.2 Area 2

Figure 15-8 Area 2 of the DDMS window

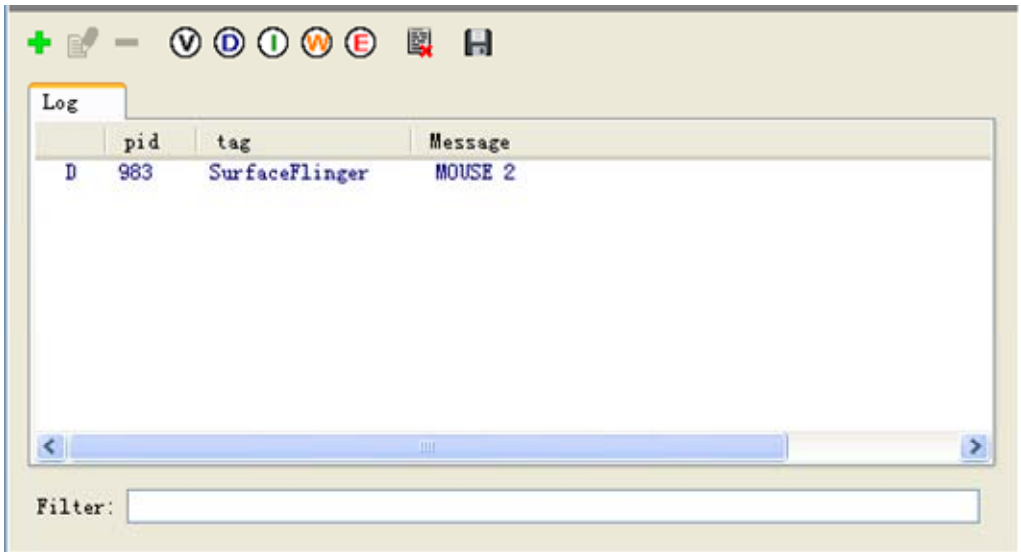


The descriptions of the items in Figure 15-8 area as follows:

- **Info**: indicates the information about the client corresponding to an APP.
- **Threads**: indicates the threads corresponding to an APP.
- **VM Heap**: indicates the details of the virtual memory or heap corresponding to an APP.
- **Allocation Tracker**: indicates the memory allocation related to an APP.
- **Sysinfo**: indicates the graphic analysis on the system status.
- **Emulator Control**: indicates the device status and behavior obtained from emulation. Note that the emulation function is removed from the system of STB products.

15.3.1.3 Area 3

Figure 15-9 Area 3 of the DDMS window



- The descriptions of the icons in Figure 15-9 are as follows:

  - : creates a log filter. Only the logs that meet the filter criteria are displayed in the filter.
  - : edits the selected filter.
  - : deletes the selected filter.
  - : clears generated logs.
  - : saves logs to a file.
  - : displays only the logs with a priority higher than V.
  - : displays only the logs with a priority higher than D.
  - : displays only the logs with a priority higher than I.
  - : displays only the logs with a priority higher than W.
  - : displays only the logs with a priority higher than E.
- The descriptions of other contents in Figure 15-9 are as follows:
  - The descriptions of the contents in the box for displaying logs are as follows (from left to right):

    First column: indicates the log priority.

    Second column: indicates the APP ID.

    Third column: indicates the log tag.

    Fourth column: indicates the log details.
  - Log content filtration box

    Filters logs by contents.

15.3.1.4 Function Supplement

The following describes the function items displayed when the DDMS is stared in other ways:

- Screenshot (see Figure 15-10):
  - This function is available when you click Devices in the left part of Figure 15-4.
  - This function is also available when you choose **Device** > **Screen capture** on the menu bar shown in Figure 15-6. See Figure 15-11.

Figure 15-10 Screenshot icon

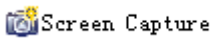
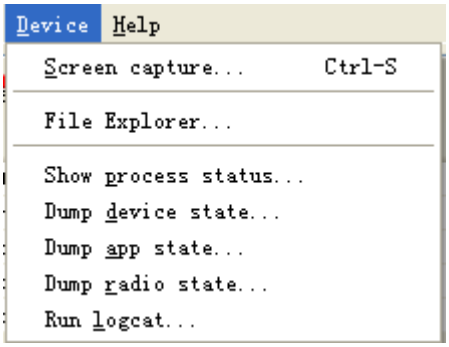


Figure 15-11 Device menu



- Clicking  shown in Figure 15-4 downloads files from the system.
- Clicking  shown in Figure 15-4 uploads files to the system.
- Clicking 

shown in Figure 15-4 views the system directory. If the DDMS window shown in Figure 15-6 is displayed, you can view the system directory by choosing **Device** > **File Explorer**.

15.3.2 Function Details

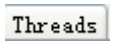
This section describes only the frequently-used functions. For more reference materials, go to http://developer.android.com/guide/developing/debugging/ddms.html.

15.3.2.1 Logcat

The logcat is the most frequently-used function for the debugger. This function displays logs that contain the running status. As there are a large number of logs, the logs need to be filtered. For details about how to filter logs, see section 16.2 "Logcat GUI Descriptions."

15.3.2.2 Viewing Information

To view the threads corresponding to an APP, perform the following steps:

Step 1 Select an APP to be viewed in area 1 shown in Figure 15-7. Click 

in area 2 shown in Figure 15-8.

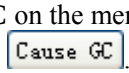
Step 2 To view the details about a thread such as the thread type and usage, double-click the thread.

----End

To view the virtual memory heap information about an APP, perform the following steps:

Step 1 Select an APP to be viewed in area 1 shown in Figure 15-7, and click  in area 1.

Step 2 Choose **Action** > **Cause GC** on the menu bar shown in Figure 15-7 or click 

in area 2 shown in Figure 15-8 and 

.

Step 3 To view the details about a data row in area 2 shown in Figure 15-8, click the data row. A graph shown in Figure 15-12 is displayed, showing the details about the data row.

----End

To view the memory allocated to an APP, perform the following steps:

Step 1 Select an APP to be viewed in area 1 shown in Figure 15-7.

Step 2 Click Start Tracking in area 2 shown in Figure 15-8 and then Get Allocations.

Step 3 To view the packet information, click the corresponding data row. See Figure 15-13.

----End

To view the system information, perform the following steps:

Step 1 Select a device to be viewed in area 1 shown in Figure 15-7.

Step 2 Click Sysinfo in area 2 shown in Figure 15-8, and choose **Memory usage** from the drop-down list. See Figure 15-14.

----End

Figure 15-12 Virtual memory analysis

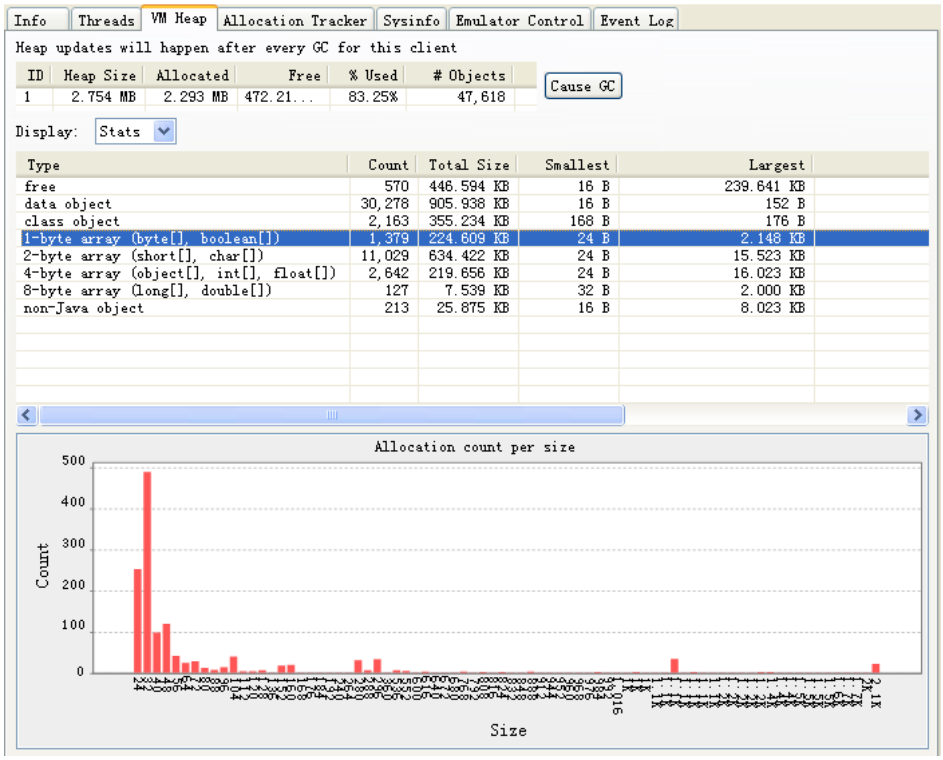


Figure 15-13 Tracking the memory allocation

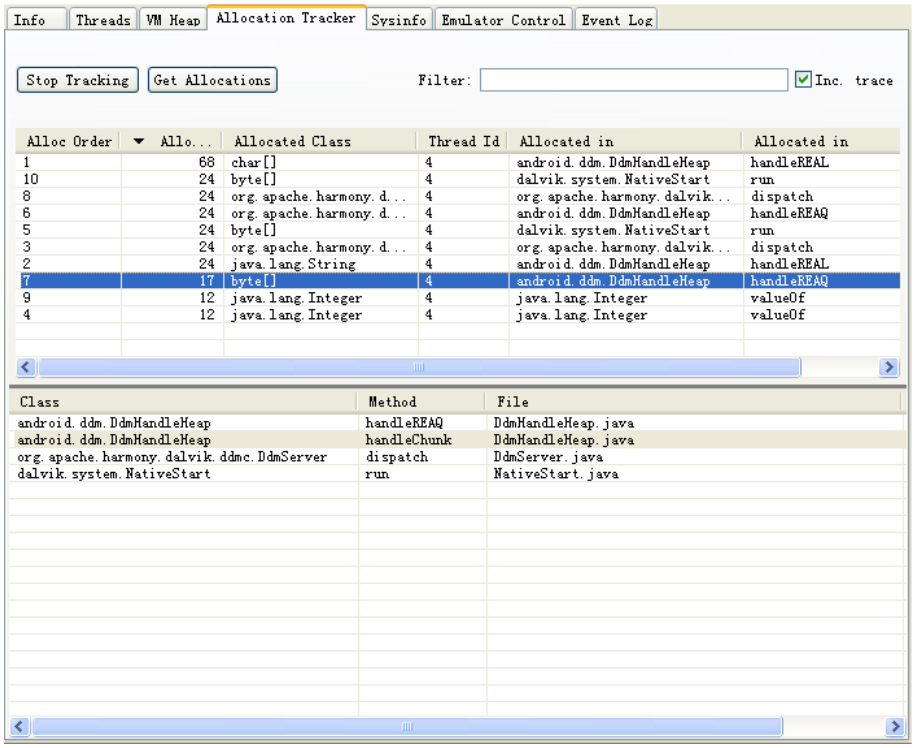
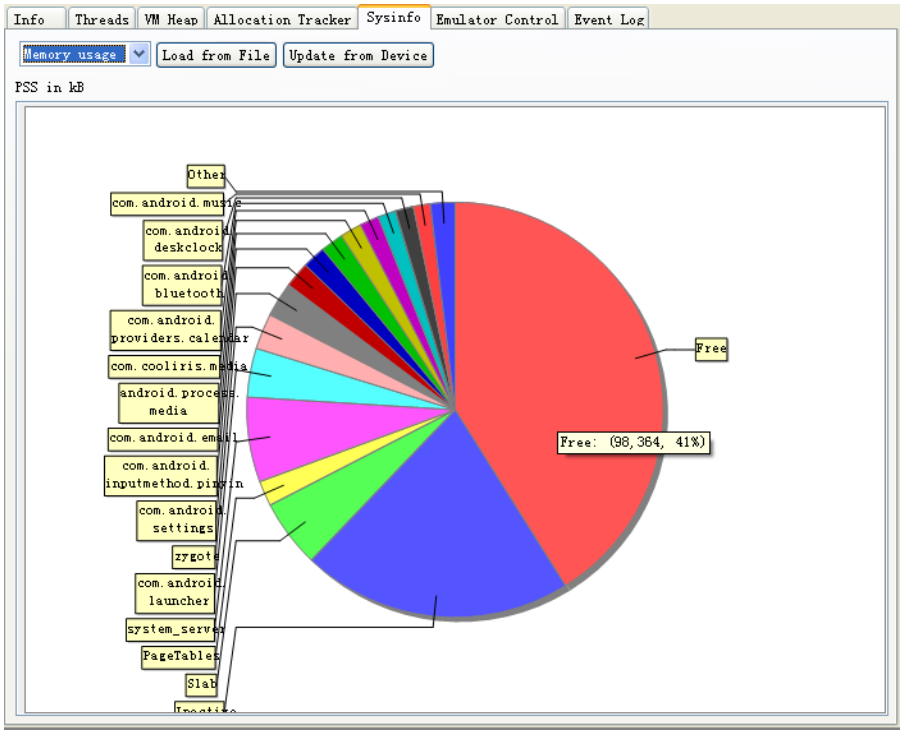


Figure 15-14 System information



16 Logcat

16.1 Overview

The Android log system records the system debugging information and allows you to view it. The logs are stored in the buffers of the software and system. To view and use buffers, run the **logcat** command. The logcat is the function that is the most frequently-used by the debugger. This function displays logs that contain the running status. As there are a large number of logs, the logs need to be filtered. Logs can be filtered in the following ways:

- By log priority

  The log priorities are listed as follows from low to high:

  - V - Verbose (lowest priority)
  - D - Debug
  - I - Info
  - W - Warning
  - E - Error
  - F - Fatal
  - S - Silent (highest priority)
- By log tag
- By APP ID
- By log contents

The Eclipse debug tool and Android DDMS provide graphical logcat, which facilitates debugging. This chapter describes how to use the logcat from two aspects:

- Graphical user interface (GUI) of the logcat
- Logcat commands

CAUTION

A log is generated only when:

- An emulator or a device is successfully connected.
- A device or an APP to be debugged is selected.

16.2 Logcat GUI Descriptions

The logcat GUIs of the Eclipse debug tool and Android DDMS are the same. This section describes how to use the logcat by using the logcat GUI (as shown in Figure 15-9) of the Android DDMS as an example. For details about the logcat functions, see section 15.3 "DDMS Function Descriptions."

- Filter logs by priority

   correspond to the log priorities V, D, I, W, and E respectively. The log text color is the same as the corresponding priority color. When you click  , only the logs whose priorities are V or higher are displayed. When you click  , only the logs whose priorities are D or higher are displayed. The rule is applicable to other priority icons.

CAUTION

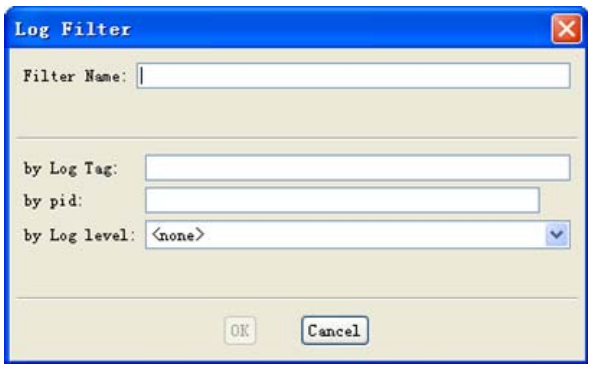
Only the selected filters can be used to filter logs by priority.

- Filter logs by using a user-defined filter

  Click  . The user-defined filter window is displayed, as shown in Figure 16-1.

  Enter a filter name in **Filter name**, set filter criteria, and click **OK**. Then the logs that meet the filter criteria are displayed in the filter.

Figure 16-1 User-defined filter



The descriptions of the items in Figure 16-1 are as follows:

- **Filter Name**: indicates the filter name. The filter can be renamed.
- **by Log Tag**: filters logs by log tag.
- **by pid**: filters logs by APP ID.
- **by Log level**: filters logs by priority.

16.3 Logcat Commands

16.3.1 Using Logcat Commands

You can view the buffer for storing system logs by running a **logcat** command

```
Command: [adb] logcat [<option>] [<filter-spec>]
Example: adb logcat or adb shell logcat
```

NOTE

If logs are viewed over the serial port, "adb" can be omitted.

For example, you can run only **logcat**.

16.3.2 Exporting Filtered Logs

16.3.2.1 Common Methods of Displaying Specific Logs

The following are the common methods of displaying specific logs:

- Control the priorities of the logs to be exported.

  Command: `adb logcat *:W`

  Description: Display only the logs whose priorities are warning or higher.
- Control the tags and priorities of the logs to be exported.

  Example: `adb logcat ActivityManager:I MyApp:D *:S`

  Description: Display all logs except the logs with the ActivityManager tag whose priorities are higher than Info and the log with the MyApp tag whose priorities are higher than Debug.
- Display the logs with a specific tag.

  Example: `adb logcat WishTV:* *:S` or `adb logcat -s WishTV`

  Description: Display only the logs with the WishTV tag.
- Display the logs with a specific priority and tag.

  Example: `adb logcat WishTV:I *:S`

  Description: Display only the logs with the WishTV tag and priority I.

NOTE

Note the following when using "*:S":

- "*:S" indicates that the priority levels for all tags are set to **silent**.
- Ensure that log output is restricted to the filters that you have explicitly specified.

16.3.2.2 Filtering Logs by Using a Regular Expression (grep)

Before using a regular expression, you need to familiarize yourself with regular expressions. The **grep** command is available only under shell or on Linux. This section describes only some functions of the **grep** command. The major parameters of the **grep** command are as follows:

- -i: The commands are case-insensitive.
- -v: The logs that do not meet the filter criteria rather than the logs that meet the filter criteria are selected.

The following are the command methods of filtering logs:

- By priority

  Example: `logcat | grep "^D"`

  Description: Display only the logs with priority D.
- By tag

  Example: `logcat | grep "^[D-S]\/WishTV"`

  Description: Display only the logs with the WishTV tag.
- Exact filter

  Example: `logcat | grep -i "^[D-S]\/WishTV.\+: HomeScreenActivity"`

  Description: Display only the logs with the WishTV tag that contain the HomeScreenActivity string.

NOTE

In regular expressions, the common symbols are described as follows:

- ^: start of a row
- $: end of a row
- .: any single character
- []: character range, for example a-e. When any character in the range is found, the logs containing the character are filtered. You can use "^" to exclude characters.

16.3.2.3 Controlling the Log Output Format

The log information contains multiple metadata fields such as tag and priority. Specific metadata fields can be displayed by changing the log output format. By using the **-v** parameter, you can obtain the logs with a specific format.

```
[adb] logcat [-v <format>]
```

The values of **format** are as follows:

- brief - Display priority, tag, and process ID (default).
- process - Display only the process ID.
- tag - Display only the priority and tag.
- thread – Display only the thread and tag of a process.
- raw – Display the raw logs without other metadata fields.
- time - Display the time, calling time, priority, tag, and process ID.
- long- Display all metadata fields and the messages separated by a blank line.

Command: `adb logcat -v thread`

Description: Display the logs in thread format.

16.3.2.4 Viewing the Available Log Buffer

The Android log system provides multiple log buffers for storing different types of log information, including radio, event, and main. When you run the **logcat** command, the log information in the main buffer is displayed by default. Generally, log information of an APP is stored in this buffer. The buffers are described as follows:

- radio - buffer for viewing the radio and telephony information
- event - buffer for viewing log information related to system events
- main - buffer for viewing default log information (the main buffer is used by default)

Command: `[adb] logcat [-b <buffer>]`

Example: `adb logcat -b radio`

Description: View the radio and telephony information in the log buffer.

17 Procrank and Dumpsys

17.1 Procrank

17.1.1 Overview

Procrank is a debugging tool inherent in the Android system. It runs in the shell environment of a device and generates memory snapshots for processes to obtain the memory utilization information, including:

- Virtual set size (VSS): size of utilized virtual memory, including the memory space occupied by shared databases
- Resident set size (RSS): size of utilized physical memory, including the memory space occupied by shared databases
- Proportional set size (PSS): size of utilized physical memory, including the proportional memory space of shared databases
- Unique set size (USS): size of physical memory exclusively occupied by a process, not including the memory space occupied by shared databases

CAUTION

- USS indicates the utilized memory size of a process and will be fully released after the process is killed.
- VSS or RSS includes the utilized memory size of shared databases and does not affect the memory status of a single process.
- PSS indicates the utilized memory size of a single process in the shared memory space after the shared memory space is segmented based on the specified proportions.

17.1.2 Usage

17.1.2.1 Running the procrank Command

- Command format:

```
procrank [ -W ] [ -v | -r | -p | -u | -h ]
```

- Common instructions:

- **-v**: sort out the outputs by VSS
- **-r**: sort out the outputs by RSS
- **-p**: sort out the outputs by PSS
- **-u**: sort out the outputs by USS
- **-R**: transfer to the ascending or descending mode
- **-w**: show the statistics of working sets
- **-W**: reset the statistics of working sets
- **-h**: show help information

- Examples:
  - Display memory snapshots by running the **procrank** command, as shown in Figure 17-1.
  - Display memory snapshots in descending order of VSS by running the **procrank –v** command.

NOTE

By default, the outputs of the **procrank** command are sorted out by PSS.

Figure 17-1 Displaying memory snapshots

```
root@android:/ # procrank
procrank
  PID      Vss      Rss      Pss      Uss  cmdline
 3396   48436K   48180K   29286K   26972K  com.android.systemui
 2158   38368K   38328K   18529K   15268K  system_server
 2889   32608K   32440K   15185K   13512K  com.hisilicon.android.allsettings
 1497   34780K   34692K   12563K    8720K  zygote
 2863   30032K   30000K   11298K    9116K  com.android.launcher
 2403   27972K   27928K   10798K    9192K  com.sohu.inputmethod.sogoupad
 2908   51852K   26664K    8317K    6252K  com.hisilicon.android.tvsettings
 2579   20580K   20524K    5215K    4000K  com.android.bluetooth
 2701   20308K   20244K    4491K    3092K  com.hisilicon.multiscreen.server
 1499    7540K    7540K    4375K    3536K  /system/bin/mediaserver
 2685   18560K   18504K    3793K    2668K  com.hisilicon.dlna.settings
```

17.1.2.2 Searching for Specified Information

This section describes how to view the memory utilization status of a specified process.

- Command format:

  ```
  procrank | grep [cmdline | PID]
  ```

  cmdline indicates the application program to be searched for, and **PID** indicates the application process to be searched for.

- Example:

  Display the memory utilization status of the **systemui** process, as shown in Figure 17-2.

  ```
  procrank | grep "com.android.systemui" or
  procrank  | grep 3396
  ```

Figure 17-2 Memory utilization status of the systemui process

```
root@android:/system/bin # procrank | grep "com.android.systemui"
 3396   49300K   49000K   30851K   28176K  com.android.systemui
```

17.1.2.3 Tracking Memory Utilization Status of Processes

You can identify memory leakage by tracking the memory utilization status of processes.

For example, you can continuously display the memory snapshots of a process by using the script compilation method and compare the USS segments to identify memory leakage.

To track the memory utilization status of the com.android.systemui process and identify memory leakage, perform the following steps:

Step 1 Compile the **example.sh** script.

```
#!/bin/bash
while true;do
adb shell procrank | grep "com.android.systemui"
sleep 1
done
```

Step 2 Connect to the board by using the ADB and run the **./example.sh** script, as shown in Figure 17-3.

Figure 17-3 Displaying the memory utilization status of the com.android.systemui process on Linux

```
~$ ./example.sh
2226   49024K   48692K   30259K   27596K  com.android.systemui
2226   49036K   48704K   30271K   27608K  com.android.systemui
2226   49040K   48708K   30275K   27612K  com.android.systemui
2226   49040K   48708K   30275K   27612K  com.android.systemui
2226   49040K   48708K   30275K   27612K  com.android.systemui
2226   49040K   48708K   30275K   27612K  com.android.systemui
```

----End

NOTE

If the USS of a process keeps increasing, memory leakage may occur. As shown in Figure 17-3, the USS of the com.android.systemui process keeps increasing for a period and then remains at a fixed value, indicating that memory allocation and utilization are normal.

17.2 Dumpsys

17.2.1 Overview

The Dumpsys is a debugging tool inherent in the Android system. It runs in the shell environment of a device and obtains status information of ongoing services. Ongoing services refer to server processes run in the Android binder mechanism.

CAUTION

The Dumpsys displays information under the following conditions:

- The Dumpsys displays information of services that have been loaded to the ServiceManager.
- The Dumpsys displays no information when the dump function in the server code is not implemented.

17.2.2 Usage

Displaying Information of All Services

Function: Display the status information of all ongoing services.

Format: `dumpsys`

NOTE

Information of all services is displayed when you run the **dumpsys** command. You are advised not to perform this operation.

Displaying Information of a Specified Service

Function: Display the dump information of a specified service.

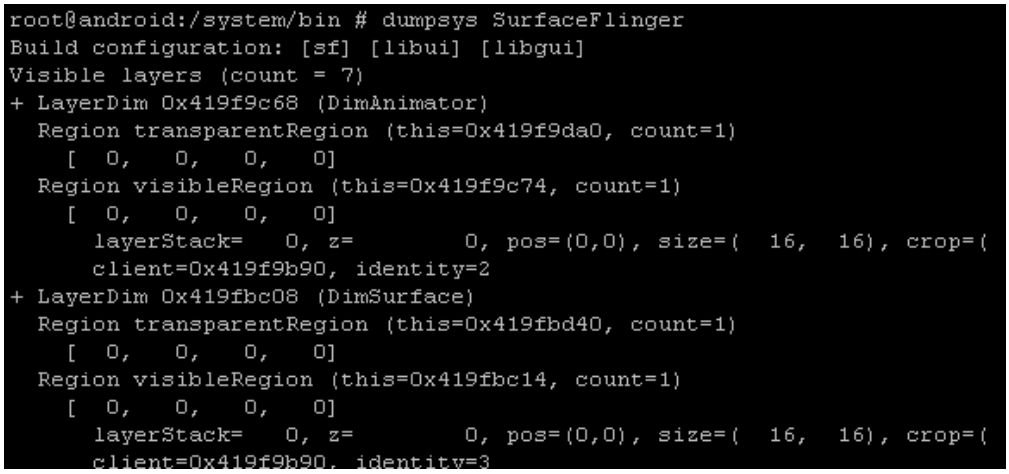
Format: `dumpsys [servicename]`

Example: Display the information of the SurfaceFlinger service by running the **dumpsys SurfaceFlinger** command. The command outputs shown in Figure 17-4 are displayed.

NOTE

The service name is case sensitive and must be entered in full name.

Figure 17-4 Displaying information of the SurfaceFlinger service



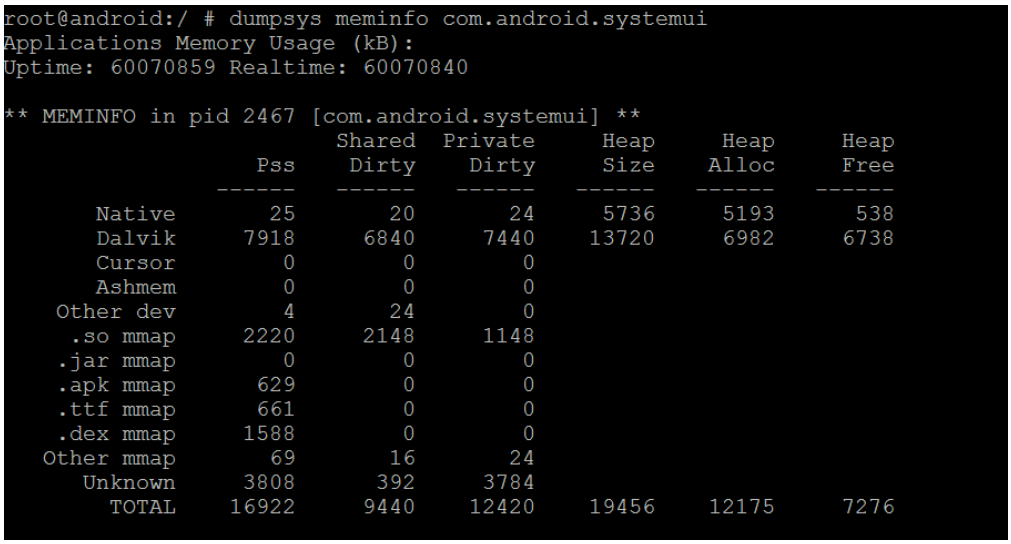
Displaying Information of a Specified Application Process in a Service

Function: Display the information of a specified application process in a service.

Format: `dumpsys [servicename] [processname]`

Example: Display the memory utilization information of the com.android.systemui process in the meminfo service by running the following command: **dumpsys meminfo com.android.systemui**. The command outputs shown in Figure 17-5 are displayed.

Figure 17-5 Displaying information of a specified application process in a specified service



17.2.2.1 Searching for Specified Information

Searching for Existing Services

Function: Display all services, as shown in Figure 17-6.

Format: `dumpsys | grep "DUMP OF SERVICE"`

Figure 17-6 Displaying all services

```
root@android:/ # dumpsys | grep "DUMP OF SERVICE"
DUMP OF SERVICE HiDisplay:
DUMP OF SERVICE SurfaceFlinger:
DUMP OF SERVICE accessibility:
DUMP OF SERVICE account:
DUMP OF SERVICE activity:
DUMP OF SERVICE airplay:
DUMP OF SERVICE alarm:
DUMP OF SERVICE appwidget:
```

Searching for a Specified Field in a Specified Service

Function: Search for a specified field in a specified service, as shown in Figure 17-7.

Format: `dumpsys serivcename | grep [fieldname]`

Example: Display all layers in the SurfaceFlinger service.

```
dumpsys SurfaceFlinger | grep Layer
```

Figure 17-7 Displaying all layers in the SurfaceFlinger service.

```
root@android:/system/bin # dumpsys SurfaceFlinger | grep Layer
+ LayerDim 0x419fbc08 (DimSurface)
+ Layer 0x419fe108 (com.android.systemui.ImageWallpaper)
+ Layer 0x41c08008 (com.android.settings/com.android.settings.Settings)
+ LayerDim 0x419f9c68 (DimAnimator)
+ Layer 0x41a01108 (com.android.settings/com.android.settings.Settings)
+ Layer 0x419fbd88 (SystemBar)
+ Layer 0x41d90808 (Sprite)
```

18 Dolby Certification

18.1 Overview

18.1.1 Introduction

Dolby certification includes digital media adapters (DMA) certification and Broadcast certification.

- DMA certification: dedicated certification for Internet-enabled products, mainly for over-the-top (OTT) STBs
- Broadcast certification: certification for DVB products in the broadcasting field, mainly for the devices using the tuner as the stream input source

18.1.2 Networking Environment

For the DMA stream input source, only the Internet playback or local playback (Internet-enabled) function needs to be tested.

For the Broadcast certification, the tuner works as the input device for tests. In addition, the Internet playback or local playback function also needs to be tested if the device supports both the tuner playback and Internet playback or local playback functions.

Figure 18-1 and Figure 18-2 show the networking environments for the DMA certification and Broadcast certification respectively.

Figure 18-1 DMA networking environment

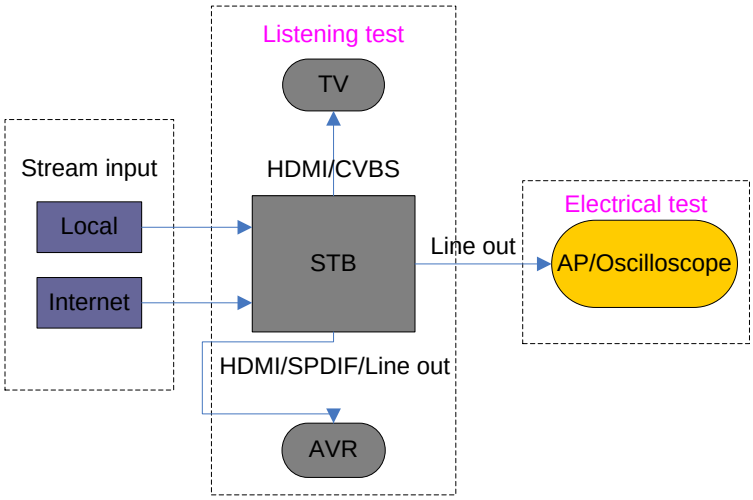
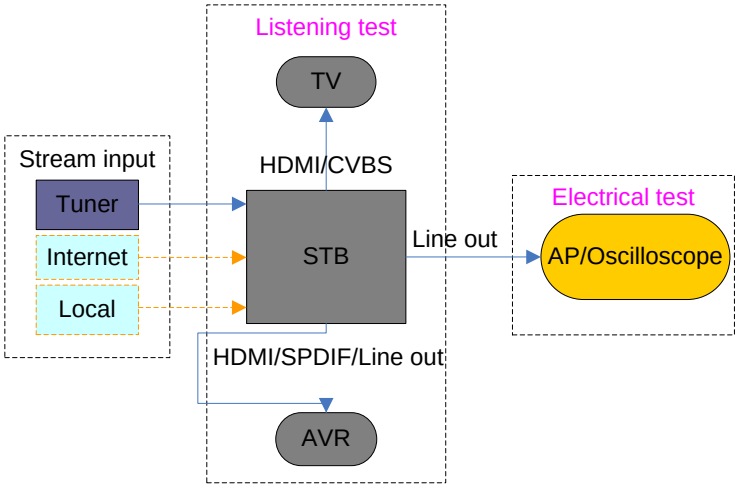


Figure 18-2 Broadcast networking environment



18.1.3 Test Form

The certification tests are classified into three categories:

- Listening test: normal stream playback, stream information display, and user setting
- Electrical test: normal stream playback, stream information display, user setting, and electrical specifications
- Recommended tests (optional): normal stream playback and stream information display

Table 18-1 is the test form.

Table 18-1 Dolby certification test form

Type	Test Item		Test Content	Expected Result	Remarks
	DMA	Broadcast			
Listening test	Channel identification (Internet enabled)	Channel identification (Internet enabled)	Audio channel identification	The audio output is smooth without noises and intermittence.	When streams from different audio channels are played, the sound output from the audio-left channel differs from that from the audio-right channel. For details, see the *ddp_s_test_bdplayertestprocedure*.
	Complicated streams (Internet enabled)	Complicated streams (Internet enabled)	Encoding and decoding of complicated streams		-
	Data-rate support (Internet enabled)	Data-rate support (Internet enabled)	Playback of streams with variable data rate		-
	Error concealment (Internet enabled)	Error concealment (Internet enabled)	Playback of streams with error frames		The stream itself has errors. The sound heard may be discontinuous because the last frame is repeated.
	Streams encoded by the consumer encoder (Internet enabled)	Streams encoded by the consumer encoder (Internet enabled)	Decoding of streams encoded by the consumer encoder		-
	Additional bitstream information (BSI) (Internet enabled)	Additional BSI streams (Internet enabled)	Decoding of streams with additional parameters		-
	AV playback	AV playback (Internet enabled)	AV playback		-

Type	Test Item		Test Content	Expected Result	Remarks
	DMA	Broadcast			
	(Internet enabled)				
	None	Associated audio	-		-
		Channel identification	Audio channel identification		-
		Complicated streams	Encoding and decoding of complicated streams		-
		Data-rate support	Playback of streams with variable data rate		-
		Error concealment	Playback of streams with error frames		-
		Streams encoded by the consumer encoder	Decoding of streams encoded by the consumer encoder		-
		Additional BSI streams	Decoding of streams with additional parameters		-
		Dual-/mono-channel support	Dual- and mono-channel playback		When the channel mode is switched during stream playback, the audio-left and audio-right channels output audio properly, and the **1+1** identifier appears in the audio

Type	Test Item		Test Content	Expected Result	Remarks
	DMA	Broadcast			
					information on the interface.
Electrical test	Reference level and relative output level	Reference level and relative output level	Configuration of the reference volume	The specifications of each test item are met.	Adjust the system volume, and set the line-out level to 100 mV.
	Dialogue normalization	Dialogue normalization	Dialogue normalization		-
	Dynamic range control	Dynamic range control	Dynamic range control		See section 0" FAQs."
	Two-channel downmix type	Two-channel downmix type	Whether the two-channel downmix type is correct		Start the testing after the streams are played for 13s.
	Two-channel downmix overload	Two-channel downmix overload	Whether overload occurs when the master volume is not controlled in downmix mode		-
	LFE downmix	LFE downmix	LFE downmix		Connect the audio-left channel on the board to the audio precision (AP) analyzer for testing.
	Maximum output level	Maximum output level	Maximum volume output		-
	Interchannel phase	Interchannel phase	Interchannel phase		-
	AV synchronization	AV synchronization	AV synchronization		The time by which the audio leads the video must be within 20 ms, and the time by which the audio lags behind the video must be

Type	Test Item		Test Content	Expected Result	Remarks
	DMA	Broadcast			
					within 30 ms.
	None	Sample rate support	Playback of streams with variable bit rate		-
		Stream information transitions	Stream information transitions		See section 18.3 "Electrical Specifications Tests."
		DVB loudness matching	Volume matching		Ensure that the volume remains the same after the programs are searched again.
		Reference level and relative output level (Internet enabled)	Reference volume		Perform the AV synchronization test again by using streams from the Internet.
		AV synchronization (Internet enabled)	AV synchronization		
Recommended tests	Frequency response and bass management	None	Frequency response and bass management	-	Recommended tests for DMA certification
	Channel delay and polarity		Synchronization test for different audio channels		
	Signal-to-noise ratio		SNR		
	THD+N vs frequency		Distortion		

Type	Test Item		Test Content	Expected Result	Remarks
	DMA	Broadcast			
	Low-frequency overload		Low-frequency overload		
	Sample rate support		Playback of streams with different sampling rate		

18.1.4 Requirements on Stream Information Display and User Setting

Requirements on stream information display for the DMA certification are as follows:

When streams are played, the basic information about the streams can be displayed, including **Audio** (stream decoding type), **Channel Config** (channel mode) (note 1), **Volume**, and **Out Channel** (output channel mode). (The information should be displayed according to the Dolby standard, with no other information displayed to avoid misunderstanding.)

- In the **Audio** item, **Dolby Digital Plus** and **Dolby Digital** should be displayed.
- It is recommended that relevant information be displayed with a non-transparent background, so that it does not overlap and interfere with the video display of the streams. For details, see Figure 18-3.

Requirements on user setting are as follows:

- Dynamic range compression (DRC) configuration (it is the night mode, used for dynamic control of the audio output level to prevent the audio output level from being too large or too small) (note 2). The DRC can be set to **On** or **Off** and the corresponding value is 100 and 0.

  It is recommended that the user settings be integrated into the system setting menu.
- HDMI output setting. It is required that the STB identifies the extended display identification data (EDID) of the connected devices to automatically decide whether to enable the passthrough function.

  Note that in the HiSilicon solution, the STB can identify the EDID in AUTO mode but not in RAW mode. You are advised to contact with Dolby on the configuration method.
- The **Dolby Certification** option under the **Advanced Options** of the HiSilicon player menu controls the stream information display, which should be enabled before testing or certification (note 3).

NOTE

- Note 1: The **Audio** and **Channel Config** information can be obtained by the player using the HiMediaPlayerInvoke interface (CMD_GET_DOLBYINFO).
- Note 2: The DRC configuration is implemented by the player using the HiMediaPlayerInvoke interface (CMD_SET_DOLBY_RANGEINFO).
- Note 3: The HiSilicon player provides user setting options. If a self-developed player is used, you are advised to design the player based on the HiSilicon player.

18.1.5 Requirements on Electrical Specifications

Requirements on the electrical specifications test are as follows:

- Verify the output level, downmixing capability, error tolerance, and underload under the high bit rate during stream playback by using the AP analyzer.
- Test AV synchronization by using an oscilloscope.

NOTE

- The A/V synchronization range can be set by the player using the HiMediaPlayerInvoke interface (CMD_SET_AVSYNC_START_REGION).
- The power consumption control over the analog video output should be disabled at the same time, which is implemented by calling the HiMediaPlayerInvoke interface (CMD_SET_DAC_DECT_ENABLE).

- Ensure that the AP analyzer inputs are connected to the specific audio-right and -left channels during electrical specifications tests. If the AP analyzer inputs are connected to incorrect audio channels, the electrical specifications cannot meet the requirements.
- (Recommended) Disable HDMI passthrough during the electrical specifications test because when the DDP HDMI passthrough is enabled, the audio output interface must work at the sampling frequency four times of the original stream frequency. For example, if the sampling rate of the DDP streams is 48 kHz, the audio output interface must work at 192 kHz when HDMI passthrough is enabled. In this case, the pulse code modulation (PCM) performs four times resampling, which affects the precision.

Figure 18-3 Dolby stream information display reference

18.2 FAQs

18.2.1 What Are the Differences Between the DMA Certification and Broadcast Certification?

Differences between the DMA certification and Broadcast certification are as follows:

- Networking environment. During DMA certification, local streams or streams from the network are tested; during Broadcast certification, streams input from the tuner are tested.
- DRC configuration. During DMA certification, the UI allows the user to enable or disable the DRC function with the On/Off switch. During Broadcast certification, the DRC switch is not required.
- DRC modes. The DOLBYPLUS-DRC_LINE mode is used in the DMA certification, and the default DOLBYPLUS-DRC_RF mode is used in the Broadcast certification.

  For the DMA certification, the DRC mode has been configured in the internal code and only the enable and disable control (On/Off) need to be set. For the Broadcast certification, only the DRC RF mode needs to be configured.
- Test item range. Some test items are mandatory for the Broadcast certification but only recommended for the DMA certification.

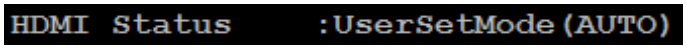
18.2.2 How Do I Determine that the Device that Does Not Support DDP Can Output DD Data Properly (Supporting EDID Automatic Detection) in HDMI Passthrough Mode?

Play DDP streams by performing the following steps:

Step 1 Ensure that the passthrough configuration is correct.

1. Set **HDMI Output** to **Auto**.
2. Set **HBR Degrade Output** to **Auto**.
3. Run **cat /proc/msp/sound0** to check whether the passthrough configuration is correct, as shown in Figure 18-4.

Figure 18-4 Sound proc information



Step 2 Connect the STB to the TV or amplifier that supports only the Dolby Digital passthrough function.

1. Run **cat /proc/msp/hdmi0_sink**. The displayed proc information in Figure 18-5 shows that the device supports PCM and Dolby Digital (DD).

Figure 18-5 HDMI proc information

2. Run **cat /proc/msp/sound0**. The displayed information in Figure 18-6 shows that the streams are output in Dolby Digital passthrough mode.

Figure 18-6 Sound proc information

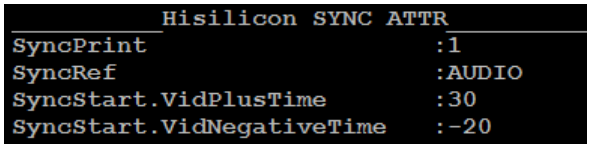
----End

18.2.3 What Are the Self-test Items?

The self-test items are listed as follows:

- Ensure that the system provides the DRC switch (On/Off) for the DMA certification.
- Ensure that the Dolby stream information is displayed correctly, and the default values or process values are not displayed (Dolby stream information is obtained by the decoder during working, and the default values should not be obtained too soon).
- Check whether the system synchronization parameters are configured correctly. Run **cat /proc/msp/sync00** to check if the audio and video synchronization range is set to −20 ms to +30 ms, as shown in Figure 18-7.

Figure 18-7 Sync proc information



18.2.4 How Do I Check Whether Underload Occurs During the Stream Playback from the Displayed proc Information?

To check whether underload occurs during the stream playback from the displayed proc information, perform the following steps:

Step 1 Run **cat /proc/msp/sound0** to check the sound proc information, as shown in Figure 18-8.

Step 2 Check the port data statistics, including the following:

```
DmaCnt(000784), BufEmptyCnt(000004), FiFoEmptyCnt(000004)
```

- **DmaCnt**: count of the normal data playback
- **BufEmptyCnt** and **FiFoEmptyCnt**: counts of buffer and FIFO underload times

CAUTION

Note that at the start of the stream playback or loopback, it is normal that **BufEmptyCnt** and **FiFoEmptyCnt** increase in the single-digit place. However, if the value increases during playback, underload occurs.

Figure 18-8 Sound proc information

```
root@android:/ # cat  /proc/msp/sound0

----------------------------------------  Sound[0]  Status  ----------------------------
SampleRate      :48000
SPDIF Status    :UserSetMode(RAW) DataFormat(PCM)
HDMI Status     :UserSetMode(AUTO) DataFormat(PCM)

----------------------------------------  OutPort Status  ----------------------------
ADAC0: Status(start), Mute(off), Vol(2), TrackMode(STEREO)
       SampleRate(048000), Channel(02), BitWidth(16), *Engine(PCM), *AOP(0x0), *PortID(0x12)
       DmaCnt(000784), BufEmptyCnt(000004), FiFoEmptyCnt(000004)

SPDIF0: Status(start), Mute(off), Vol(2), TrackMode(STEREO)
       SampleRate(048000), Channel(02), BitWidth(16), *Engine(PCM), *AOP(0x1), *PortID(0x21)
       DmaCnt(000784), BufEmptyCnt(000004), FiFoEmptyCnt(000004)

HDMI0: Status(start), Mute(off), Vol(2), TrackMode(STEREO)
       SampleRate(048000), Channel(02), BitWidth(16), *Engine(PCM), *AOP(0x2), *PortID(0x13)
       DmaCnt(000784), BufEmptyCnt(000005), FiFoEmptyCnt(000004)
```

----End

18.3 Electrical Specifications Tests

18.3.1 Synchronization Specification Test

The methods of testing the synchronization specifications are as follows:

- Check the synchronization specifications by using an oscilloscope.
- Check the synchronization specifications by recording the waveform (with a 3.5 mm Stereo-to-RCA cable) using the PC sound device (3.5 mm microphone).

To check the specifications by recording the waveform, perform the following steps:

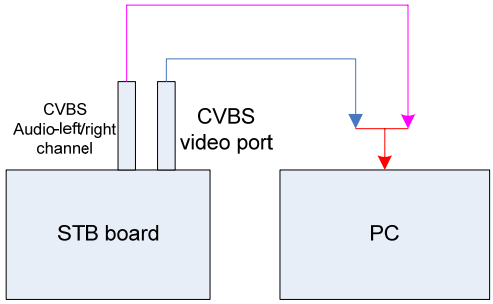
Step 1 Build a test environment, as shown in Figure 18-9. You are advised to use Cool Edit to record the sound on the PC.

Step 2 After the test environment is built, play the streams used to test the synchronization specifications, record the sound, check the waveforms, enlarge the waveforms, and test the AV signal synchronization specifications by holding the left mouse button down and dragging among AV signals.

CAUTION

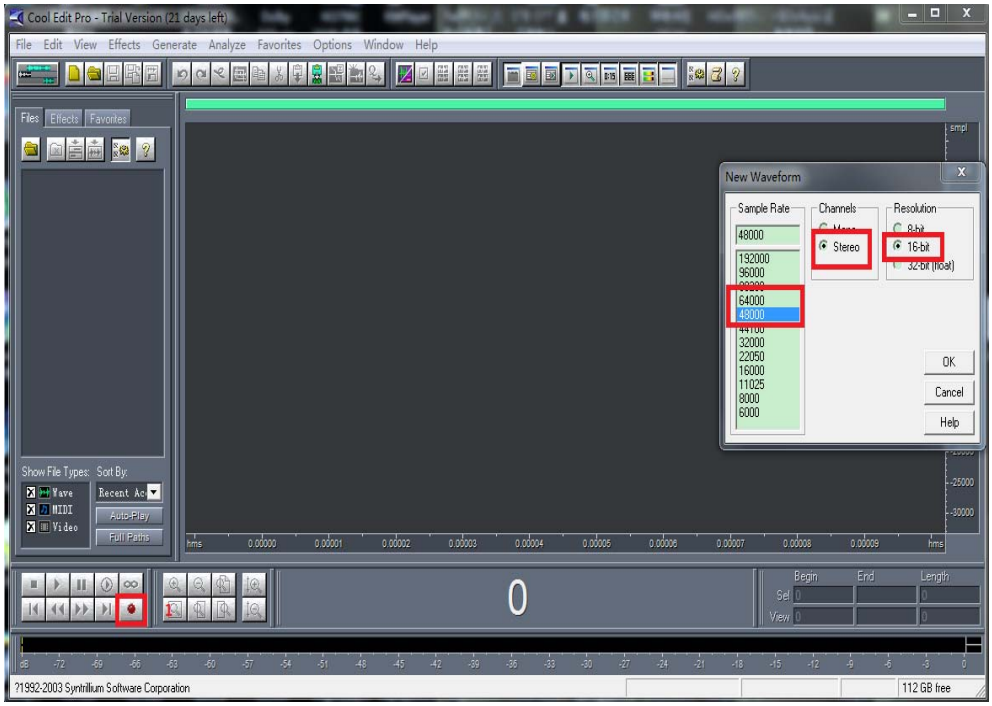
- Only one of the CVBS audio-left and auto-right channels needs to be connected. The data of the audio-left and audio-right channels is synchronized under normal conditions.
- The system volume needs to be adjusted during sound recording to avoid clipping.

Figure 18-9 Environment for recording waveforms for the AV synchronization specification test



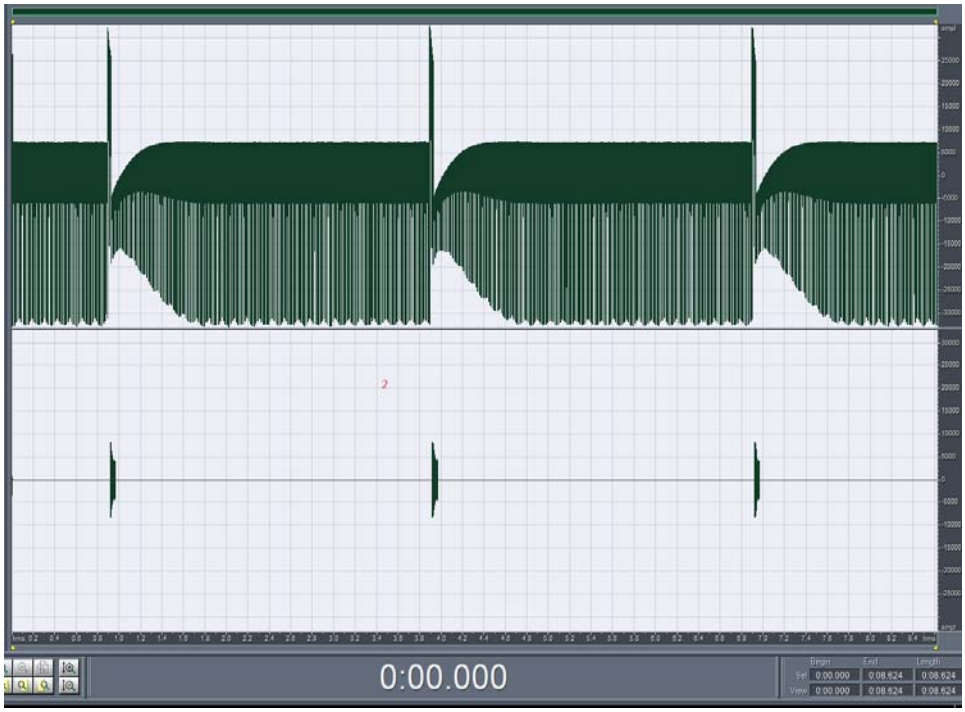
Step 3 Open Cool Edit, and click the red circle button on the lower left part of the UI (as shown in area 1 in Figure 18-10). The **New Waveform** dialog box is displayed, as shown in Figure 18-10.

Figure 18-10 Recording waveforms using Cool Edit



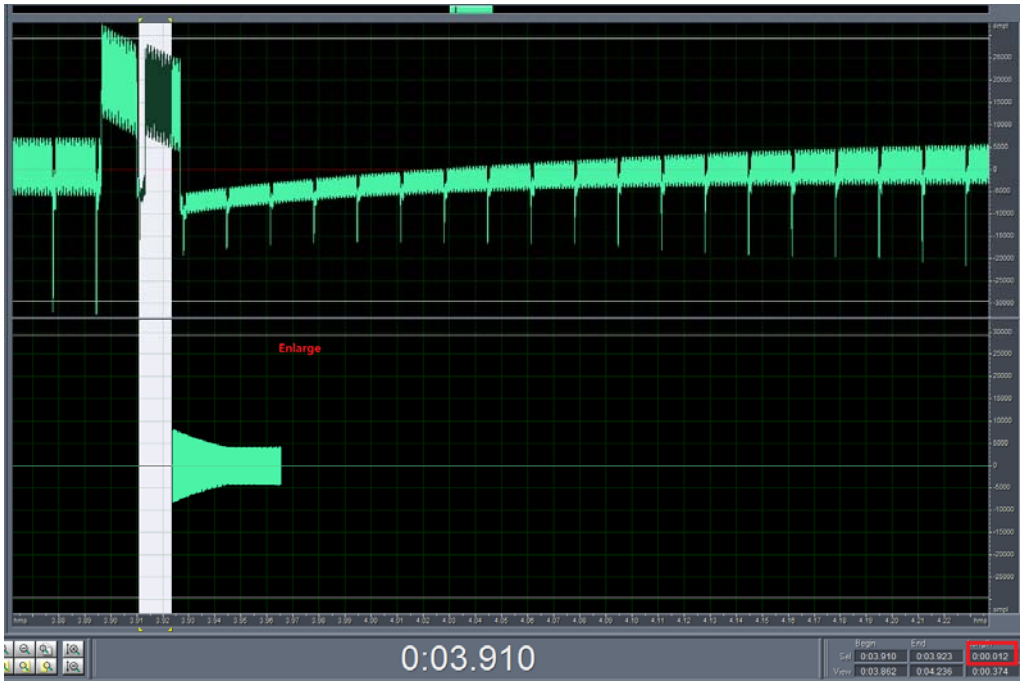
Step 4 Select **48000** in the **Sample Rate** combo box, select **Stereo** and **16-bit** respectively in the **Channels** and **Resolution** check boxes, and click **OK**. The recorded waveforms are displayed, as shown in Figure 18-11. After the recording completes, click the red circle button again or click the pause button.

Figure 18-11 Pausing the recording



Step 5 Enlarge the waveforms by scrolling the middle mouse button. Hold down the left mouse button and drag the pointer between the two ends of the signal to be checked. A 12 ms time delay is shown in the lower right corner, as shown in Figure 18-12.

Figure 18-12 Enlarging a waveform



----End

18.3.2 BSI Specification Test

When testing the BSI specifications, record the audio and check whether the duration of the transited waveforms exceeds 20 ms. The recording process is the same as that described in section 18.3.1 "Synchronization Specification Test." The recorded waveforms are displayed in Figure 18-13 and are analyzed as follows:

- The glitches are caused by the changes of stream attributes. Enlarge the glitches to check whether the duration exceeds the required time period.
- Check the duration of the glitches (at most 10 ms). See Figure 18-14.
- If an AP analyzer is used to test the BSI specification, the test result will exceed 20 ms. However, if the duration of the transited waveforms obtained from the PC sound recording analysis is shorter than 20 ms, this test item can pass the Dolby certification (this has been confirmed with the Dolby laboratory).

Figure 18-13 BSI specification analysis 1

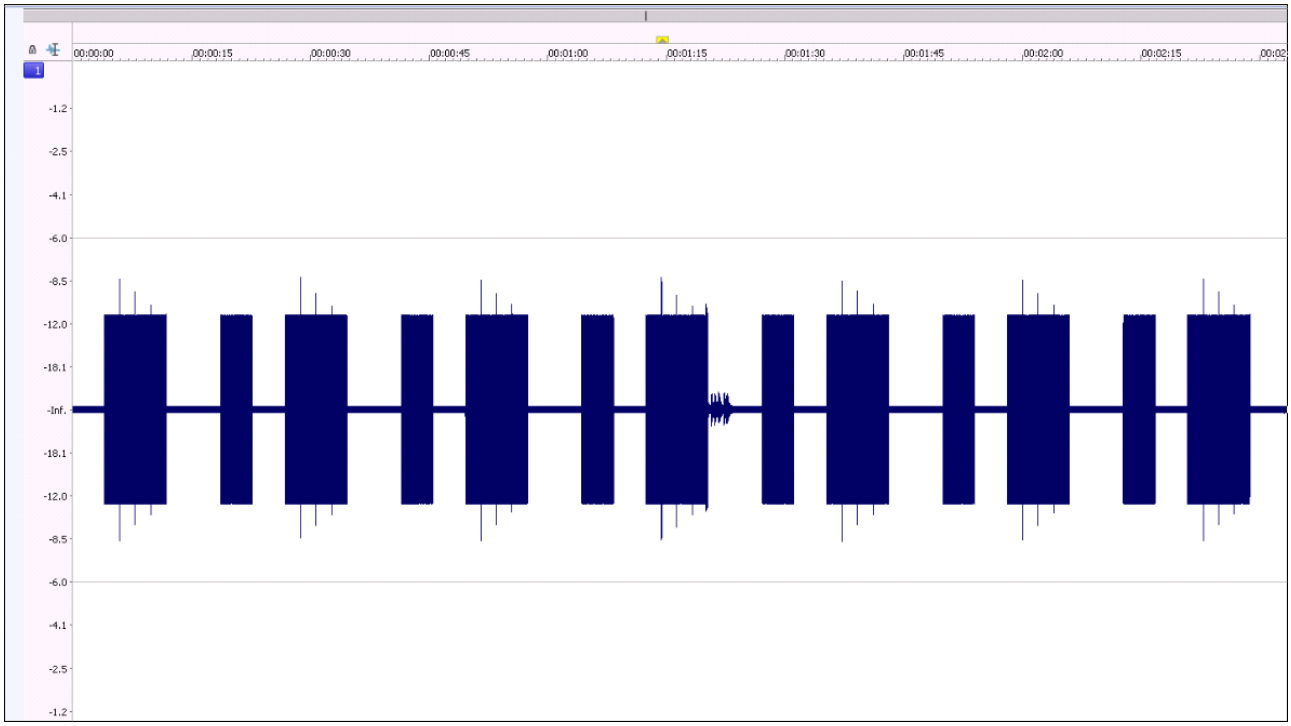
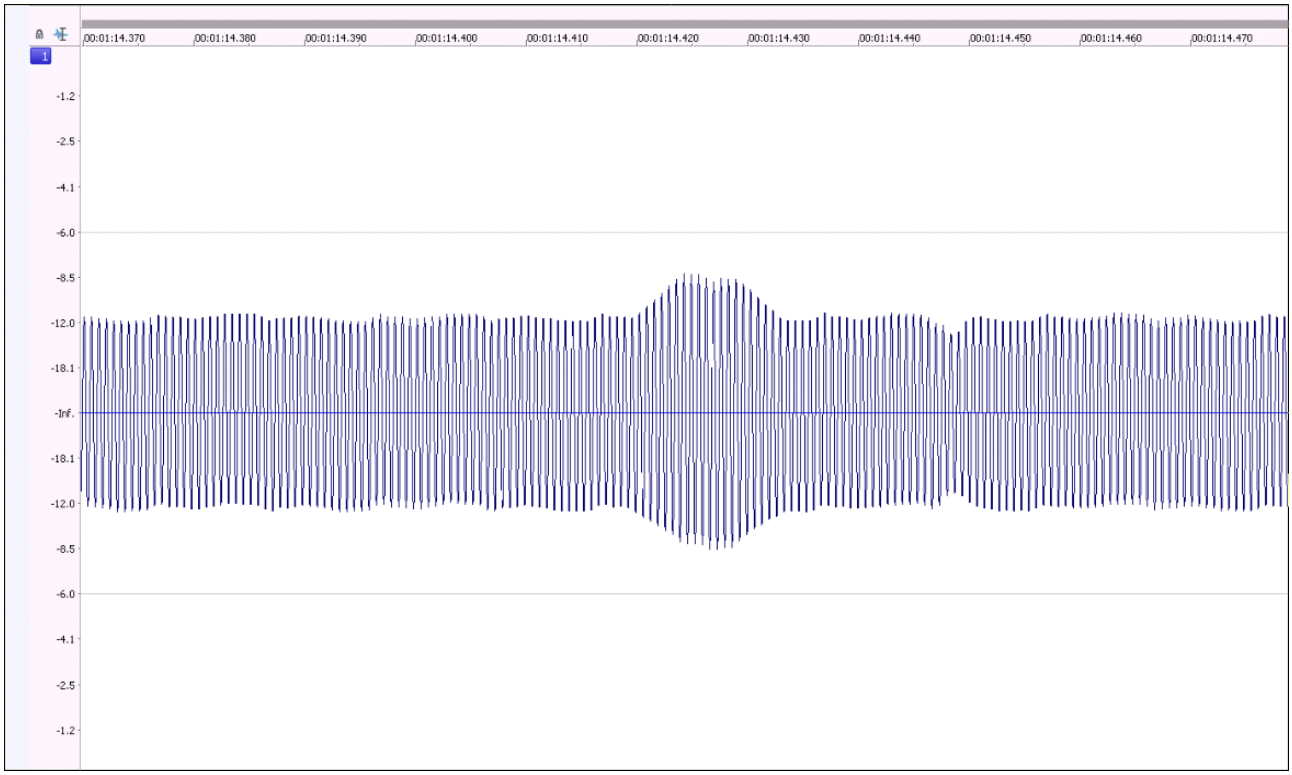


Figure 18-14 BSI specification analysis 2



18.4 Customer Self-Test Form

18.4.1 DMA Self-Test Form

Prerequisites for the test are as follows:

- The chip OTP supports Dolby.
- A genuine Dolby decoding library has been obtained, and the library has been copied to **board/system/lib** (Android) or **/usr/lib** (Linux)
- The audio input and output ports have been configured properly.

```
Android:
 HDMI Output-> Auto
 Spdif Output -> RAW
 HBR DegradeOutput -> Auto
Linux:
 HI_UNF_SND_SetHdmiMode(HI_UNF_SND_0, HI_UNF_SND_OUTPUTPORT_HDMI0,
                        HI_UNF_SND_HDMI_MODE_AUTO)
 HI_UNF_SND_SetSpdifMode(HI_UNF_SND_0, HI_UNF_SND_OUTPUTPORT_SPDIF0,
                        HI_UNF_SND_SPDIF_MODE_RAW)
```

- Test streams have been obtained from Dolby or HiSilicon.
- The networking environment has been built. For details, see Figure 18-1.
- If the HiSilicon player is used for the test, ensure that corresponding patches in the list of patches to be integrated for the version have been integrated.

Table 18-2 DMA self-test form

Test Item	Dolby® Digital Plus Decoder for Blu-ray Disc Players Test Procedure Issue 4			Test Results			
	Test Item	**Test Signal**	**Expected Result**	**Subjective Listening Test**	**Whether Underload Occurs (cat /proc/msp/sound0)**	**Specifications**	**Remarks (Symptoms and proc Information When the Test Fails)**
Audio-left/right channel	2.3.1 Channel Identification (Internet enabled)	2ch_24fps_voices_id_2.mp4	The audio-left channel plays **Left Channel** and the audio-right channel plays **Right Channel** when the stream is played.	Pass	Pass	N/A	-
Limited stream	2.3.3 Data-Rate Support (Internet enabled)	2ch_24fps_datarate_128_2.mp4	The audio output is smooth without noises and intermittence when the stream is played.	Pass	Pass		-
		5ch_24fps_datarate_3024_7.mp4		Pass	Pass		-
UI	2.3.7 A/V Playback (Internet enabled)	5ch_24fps_avplaback1_7.mp4	Both audio and video are output when the stream is played. The audio output is smooth without noises and intermittence, and the video output is smooth. For requirements on the stream information display, see section 18.1.4 "Requirements on Stream Information Display and User Setting."	Pass	Pass		-

Test Item	Dolby® Digital Plus Decoder for Blu-ray Disc Players Test Procedure Issue 4			Test Results			
	Test Item	Test Signal	Expected Result	Subjective Listening Test	Whether Underload Occurs (cat /proc/msp/sound0)	Specifications	Remarks (Symptoms and proc Information When the Test Fails)
Passthrough test	3.1.4 Two-Channel Downmix Type	5ch_24fps_downmix_type_7_ddp.mp4	HDMI connects to the TV that supports PCM decoding, and the TV has sound output.	Pass	Pass		-
			HDMI connects to an amplifier that does not support HDMI 1.3 (supports only the 5.1 output), and the amplifier displays **DD** or **Dolby Digital**. The passthrough output is normal.	Pass	Pass		-
			HDMI connects to an amplifier that supports HDMI 1.3 or above (supports the HDMI 7.1 output), and the amplifier displays **DDP** or **Dolby Digital Plus**. The passthrough output is normal.	Pass	Pass		-
			The S/PDIF interface connects to an amplifier and the amplifier displays **DD** or	Pass	Pass		-

Test Item	Dolby® Digital Plus Decoder for Blu-ray Disc Players Test Procedure Issue 4			Test Results			
	Test Item	Test Signal	Expected Result	Subjective Listening Test	Whether Underload Occurs (cat /proc/msp/sound0)	Specifications	Remarks (Symptoms and proc Information When the Test Fails)
			Dolby Digital. The passthrough output is normal.				
DRC test	3.4.3 Dynamic Range Control	2ch_24fps_drc_2.mp4	The On/Off switch is provided.	N/A	Pass		-
Synchronization specification	3.4.9 Audio/Video Synchronization	5ch_24fps_av_sync_7.mp4	The time by which the audio leads the video must be within 20 ms, and the time by which the audio lags behind the video must be within 30 ms.	N/A	Pass	(__ms~__ms)	-
System information	None	Android version: _____ Linux version: _____ HiSilicon chip information: Sound Proc (cat /proc/msp/sound0) Dolby library information: ADEC	The system information needs to be filled in for HiSilicon to locate the issues quickly when a self-test item fails.	N/A			-

Test Item	Dolby® Digital Plus Decoder for Blu-ray Disc Players Test Procedure Issue 4			Test Results			
	Test Item	Test Signal	Expected Result	Subjective Listening Test	Whether Underload Occurs (cat /proc/msp/sound0)	Specifications	Remarks (Symptoms and proc Information When the Test Fails)
		Proc (cat /proc/msp/adec00)					

18.4.2 Broadcast Self-Test Form

Prerequisites for the test are as follows:

- The chip OTP supports Dolby.
- A genuine Dolby decoding library has been obtained, and the library has been copied to **board/system/lib** (Android) or **/usr/lib** (Linux).
- The audio input and output ports have been configured properly.

```
Android:
 HDMI Output-> Auto
 Spdif Output -> RAW
 HBR DegradeOutput -> Auto
Linux:
 HI_UNF_SND_SetHdmiMode(HI_UNF_SND_0, HI_UNF_SND_OUTPUTPORT_HDMI0,
                        HI_UNF_SND_HDMI_MODE_AUTO)
 HI_UNF_SND_SetSpdifMode(HI_UNF_SND_0, HI_UNF_SND_OUTPUTPORT_SPDIF0,
                         HI_UNF_SND_SPDIF_MODE_RAW)
```

- Test streams have been obtained from Dolby or HiSilicon.
- The networking environment has been built. For details, see Figure 18-2.

Table 18-3 Broadcast self-test form

Test Item	Dolby® Digital Plus Decoder for Consumer Broadcast Products Test Procedure Issue 4			Test Results			
	Test Item	Test Signal	Expected Result	Subjective Listening Test	Whether Underload Occurs (cat /proc/msp/ sound0)	Specifications	Remarks (Symptoms and proc Information When the Test Fails)
Limited stream	2.2.3 Data-Rate Support	2ch_dtrtswp_64to640_2_dd_DVB_h264_25fps.trp	The audio output is smooth without noises and intermittence when the stream is played.	Pass	Pass	N/A	-
		2ch_dtrtswp_96to640_2_ddp_DVB_h264_25fps.trp		Pass	Pass		
UI	2.2.7 Dual-Mono Support	2ch_dualmono_0_ddp_DVB_h264_25fps.trp	During the stream playback, there is sound output, and the **1+1** identifier appears in the audio information on the interface. 	Pass	Pass		-
		2ch_dualmono_0_dd_DVB_h264_25fps.trp	During the stream playback, there is sound output, and the **1+1** identifier appears in the audio information on the interface. 	Pass	Pass		-

Test Item	Dolby® Digital Plus Decoder for Consumer Broadcast Products Test Procedure Issue 4			Test Results			
	Test Item	Test Signal	Expected Result	Subjective Listening Test	Whether Underload Occurs (cat /proc/msp/ sound0)	Specifications	Remarks (Symptoms and proc Information When the Test Fails)
	3.1.6 Loudness Matching	loudness_23_mp2_DVB_h264_25fps.trp	During the stream playback, there is sound output, and the **MP2** identifier appears in the audio information on the interface. 	Pass	Pass		-
Passthrough test	None	6ch_diff_mus1_7_ddp_DVB_h264_25fps.trp	HDMI connects to the TV that supports PCM decoding, and the TV has sound output.	Pass	Pass		-
			HDMI connects to an amplifier that does not support HDMI 1.3 (supports only the 5.1 output), and the amplifier displays **DD** or **Dolby Digital**. The passthrough output is normal.	Pass	Pass		-
			HDMI connects to an amplifier that supports HDMI 1.3 or above (supports the HDMI 7.1 output), and the amplifier displays **DDP** or **Dolby Digital Plus**. The passthrough output is normal.	Pass	Pass		-
			The S/PDIF interface connects to an amplifier (optional test item, recommended if the amplifier supports S/PDIF input), and the amplifier displays **DD** or **Dolby Digital**. The passthrough output is normal.	Pass	Pass		-
Synchronizati	3.1.12 Audio/	2ch_av_sync_2_dd_DVB_h264_25fps.tr	The amount of time by which the audio leads the video is within 20 ms, and the amount of time by	N/A	Pass	(__ms−__ms)	-

Test Item	Dolby® Digital Plus Decoder for Consumer Broadcast Products Test Procedure Issue 4			Test Results			
	Test Item	Test Signal	Expected Result	Subjective Listening Test	Whether Underload Occurs (cat /proc/msp/ sound0)	Specifications	Remarks (Symptoms and proc Information When the Test Fails)
on specification	Video Synchronization	p	which the audio lags the video is within 30 ms.				
		2ch_av_sync_2_ddp_DVB_h264_25fps.trp		N/A	Pass	(__ms~__ms)	
System information	None	Android version: _____ Linux version: _____ HiSilicon chip information: Sound Proc (cat /proc/msp/sound0) Dolby library information: ADEC Proc (cat /proc/msp/adec00)	The system information needs to be filled in for HiSilicon to locate the issues quickly when a self-test item fails.	N/A			-

19 Passthrough Output Settings

19.1 Passthrough Settings When the Genuine Dolby and DTS Libraries Are Used

Table 19-1 describes the passthrough settings when the genuine Dolby and DTS libraries are used.

Table 19-1 Passthrough settings when the genuine Dolby and DTS libraries are used

Stream Type	**Genuine DDP/DTSHD Libraries Used**										
	HDMI Settings	**Disabled**		**Auto**		**LPCM**		**RAW**		**HBR2LBR**	
	S/PDIF Settings	**LPCM**	**RAW**	**LPCM**	**RAW**	**LPCM**	**RAW**	**LPCM**	**RAW**	**LPCM**	**RAW**
DDP	HDMI	N/A	N/A	EDID	EDID	LPCM 2.0	LPCM 2.0	DDP	DDP	DD	DD
	S/PDIF	LPCM 2.0	DD	LPCM 2.0	DD	LPCM 2.0	DD	LPCM 2.0	DD	LPCM 2.0	DD
	ADAC	LPCM 2.0								LPCM 2.0	LPCM 2.0

Stream Type	Genuine DDP/DTSHD Libraries Used											
	HDMI Settings	Disabled		Auto		LPCM		RAW		HBR2LBR	
	S/PDIF Settings	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW
DTSHD	HDMI	N/A	N/A	EDID	EDID	LPCM 2.0	LPCM 2.0	DTSHD	DTSHD	DTS	DTS
	S/PDIF	LPCM 2.0	DTS	LPCM 2.0	DTS	LPCM 2.0	DTS	LPCM 2.0	DTS	LPCM 2.0	DTS
	ADAC	LPCM 2.0								LPCM 2.0	LPCM 2.0

NOTE

- S/PDIF and HDMI settings: S/PDIF and HDMI output modes configured on the Android setting menu
- Stream type: format of the played stream
- HDMI, S/PDIF, and ADAC: output ports

19.2 Passthrough Settings When Only the Dolby and DTS Passthrough Libraries Are Used

Table 19-2 describes the passthrough settings when only the Dolby and DTS passthrough libraries are used.

Table 19-2 Passthrough settings when only the Dolby and DTS passthrough libraries are used

Stream Type	DDP/DTSHD Passthrough Libraries Used										
	HDMI Settings	Disabled		Auto		LPCM		RAW		HBR2LBR	
	S/PDIF Settings	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW
DDP	HDMI	N/A	N/A	EDID	EDID	Mute	Mute	DDP	DDP	Mute	Mute
	S/PDIF	Mute	Mute	Mute	Mute	Mute	Mute	Mute	Mute	Mute	Mute
	ADAC	Mute	Mute	Mute	Mute	Mute	Mute	Mute	Mute	Mute	Mute
DTSHD	HDMI	N/A	N/A	EDID	EDID	Mute	Mute	DTSHD	DTSHD	DTS	DTS
	S/PDIF	Mute	DTS	Mute	DTS	Mute	Mute	Mute	DTS	Mute	DTS
	ADAC	Mute	Mute	Mute	Mute	Mute	Mute	Mute	Mute	Mute	Mute

19.3 Passthrough Settings When the Genuine TrueHD Library Is Used

Table 19-3 describes the passthrough settings when the genuine TrueHD library is used.

Table 19-3 Passthrough settings when the genuine TRUEHD library is used

Stream Type	Genuine TrueHD Decoding Library (with PCM Decoding Function) Used																			
	HDMI Settings	Auto						LPCM		RAW						Disabled				
	Blu-ray Next Generation	Auto		RAW 5.1		RAW 7.1		N/A		Auto		RAW 5.1		RAW 7.1		Auto		RAW 5.1		RAW 7.1	
	S/PDIF Settings	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW
TrueHD+DD	HDMI	EDID①		DD		TrueHD		LPCM 2.0		TrueHD		DD		TrueHD		Mute		DD (output muted)		TrueHD (output muted)	
	S/PDIF	EDID②	EDID③	LPCM 2.0	DD	EDID⑥		LPCM 2.0	DD	EDID⑥		LPCM 2.0	DD	EDID⑥		EDID⑦		LPCM 2.0	DD	EDID⑥	
	ADAC	EDID②		LPCM 2.0		EDID⑥		LPCM 2.0		EDID⑥		LPCM 2.0		EDID⑥		EDID⑦		LPCM 2.0		EDID⑥	
TrueHD	HDMI	EDID⑤		LPCM 2.0		TrueHD		LPCM 2.0		TrueHD		LPCM 2.0		TrueHD		TrueHD (output muted)		Mute		TrueHD (output muted)	
	S/PDIF	EDID④		LPCM 2.0		EDID⑥		LPCM 2.0		EDID⑥		LPCM 2.0		EDID⑥		EDID⑥		LPCM 2.0		EDID⑥	
	ADAC	EDID		LPCM 2.0		EDID⑥		LPCM 2.0		EDID⑥		LPCM 2.0		EDID⑥		EDID⑥		LPCM 2.0		EDID⑥	

Stream Type	Genuine TrueHD Decoding Library (with PCM Decoding Function) Used																					
	HDMI Settings	Auto						LPCM		RAW						Disabled						
	Blu-ray Next Generation	Auto		RAW 5.1		RAW 7.1		N/A		Auto		RAW 5.1		RAW 7.1		Auto		RAW 5.1		RAW 7.1		
	S/PDIF Settings	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	LPCM	RAW	
		④																				

NOTE

- EDID①: If the playback device supports the TrueHD data, the TrueHD data is output. If the playback device does not support the TrueHD data, the DD data is output. If there is no DDP genuine decoding library, the DDP passthrough library is used. If there is no DDP passthrough library, the output is muted. If the playback device does not support passthrough, the DD decoding PCM 2.0 data is output. If there is no DDP genuine decoding library, the output is muted (the TrueHD data is supported only by the chip that supports Dolby).
- EDID②: Muted (when the HDMI outputs the TrueHD data) or LPCM 2.0 (when the HDMI outputs the DD or LPCM 2.0 data, or the HDMI cable is removed after the HDMI outputs the TrueHD data and before the stream playback stops)
- EDID③: Muted (when the HDMI outputs the TrueHD data or when the HDMI output is muted), DD (when the HDMI outputs the DD or LPCM 2.0 data), or LPCM 2.0 (when the HDMI cable is removed after the HDMI outputs the TrueHD data and before the stream playback stops)
- EDID④: Muted (when the HDMI outputs the TrueHD data) or LPCM 2.0 (when the HDMI outputs the LPCM 2.0 data)
- EDID⑤: If the playback device supports the TrueHD data, the TrueHD data is output. Otherwise, the LPCM 2.0 data is output.
- EDID⑥: Muted (when the HDMI outputs the TrueHD data) or LPCM 2.0 (when the HDMI cable is not inserted or is removed)
- EDID⑦: If the DDP library is used during the previous playback, the output status is the same as that when the Blu-ray next generation is configured as RAW 5.1. If the TrueHD library is used during the previous playback, the output status is the same as that described in EDID⑥.

19.4 Multi-Channel Passthrough Settings

Table 19-4 describes the multi-channel passthrough settings.

Table 19-4 Multi-channel passthrough settings

HDMI Setting	**Disabled**	**Auto**			**LPCM**			**RAW**			**HBR2LBR**		
LPCM Source Signal (Hz)	**Channel**												
	All	**2**	**5.1**	**7.1**	**2**	**5.1**	**7.1**	**2**	**5.1**	**7.1**	**2**	**5.1**	**7.1**
32000	Mute	2-channel 48 kHz	EDID		2-channel 48 kHz			2.0 PCM RAW	7.1 PCM RAW	7.1 PCM RAW	2.0 PCM RAW	2.0 PCM RAW	2.0 PCM RAW
44100													
48000													
88200													
96000													
176400													
192000													

NOTE

EDID: If the playback device supports the 7.1 channel, 7.1 PCM data is output in passthrough mode. Otherwise, 2-channel 48 kHz PCM is output.

CAUTION

When data is output in the PCM RAW mode, volume adjustment is invalid.

20 Customization of Low-Memory Versions

20.1 Overview

The HiSilicon low-memory versions are designed with specified service specifications and restrictions to meet customer requirements on cost reduction.

The HiSilicon low-memory versions are classified into two types by the STB RAM size:

- 512 MB version
  - Full high definition (FHD) version: This version is supported by the 512 MB STB by default. It supports FHD, H.265, and SBS/TAB, and does not support 4K decoding, MVC, and CTS. Its application compatibility is limited.
  - Ultra high definition (UHD) version: This version is supported by the 512 MB STB after the configuration file is modified. It supports UHD, H.265, and SBS/TAB/MVC, and does not support the 4K output and CTS. Its 4K decoding performance and application compatibility is limited.

NOTE

In the two 512 MB versions, the FHD version is supported by default, and the UHD version can be obtained by compiling the FHD version after the compilation configuration is modified. For details about the modification method, see section 20.2.2.2 "Compilation and Configuration of the 512 MB UHD Version."

- 768 MB version

  UHD version: This version is supported by the 768 MB STB by default. It supports UHD, H.265, SBS/TAB/MVC, 4K decoding, and 4K output. It does not support CTS, and the application compatibility is limited.

NOTE

The 768 MB version adopts the asymmetric memory. To support this version, corresponding system boot table needs to be replaced. For details about the replacement method, see section 20.2.3.2 "Boot Table Replacing."

The low-memory versions and the 1 GB version share the same image, which means that the same image is supported by the STBs with 512 MB, 768 MB, or 1 GB memory. The system automatically identifies these STBs by reading the DDR size.

The low-memory version comes from the standard version with tailored specifications. The specifications/functions that are not supported by the HiSilicon low-memory versions are as follows:

- CTS.
- 1080p UI (The media memory zone (MMZ) memory usage increases by about 67 MB for the 1080p UI)
- Digital right management (DRM)
- Dynamic wallpaper

20.2 Version Introduction

20.2.1 512 MB FHD Version

20.2.1.1 Specifications

Table 20-1 lists the detailed specifications for the 512 MB FHD version (other specifications are the same as those of the 1 GB version).

Table 20-1 Specifications of the 512 MB FHD version

Specifications	Description
UI	720p UI supported
Output interface	HDMI 1.4 and A/V output
Video decoding capability	1080p@60 fps, with a maximum of 100 Mbit/s bit rate
3D	1080p SBS and TAB supported; MVC not supported
4K	4K decoding not supported
PIP	Supported, 720p for both windows
Video encoding capability	1080p@30 fps H.264 encoding supported
CTS	Not supported
DRM	Not supported
fastplay	Supported, 1080p 2D streams
Static wallpaper	Supported (disabled by default)
Dynamic wallpaper	Not supported
DLNA	DMP or DMR supported. DLNA is not supported in scenarios where the playback resources in the background are not released.
airplay	Not supported
multiscreen	Not supported
transcode	Not supported

Specifications	Description
Video communication	Supported (maximum 720p)
IPTV application	ANDROIDMOV, Wasu, BestTV, and CNTV supported
Browser	Supported (with low compatibility). The browser exits abnormally when the occupied memory exceeds 200 MB.
3D game	Supported (with low compatibility). The 3D game exits abnormally when the occupied memory exceeds 200 MB.

20.2.1.2 Memory Usage

The 512 MB FHD version supports FHD and H.265, and does not support 4K decoding, MVC, and CTS. Its application compatibility is limited.

The detailed decoding configuration is as follows:

- Eight decoding frame buffers, with six buffers for storing reference frames and two buffers for storing buffer frames
- Six vector processing subsystem (VPSS) frame buffers
- 5 MB stream buffer
- 112 MB reserved memory for MMZ
- 250 MB memory available for applications

Table 20-2 lists the detailed memory usage.

Table 20-2 Memory usage

Item		FHD 1080p SBS/TAB (512 MB)
RAM	Total RAM	500388 KB
MMZ	Base memory	13812 KB
	One-channel 3D playback (SBS/TAB)	82538 KB
	720p FB	10440 KB
	Total MMZ (theoretically)	106790 KB
	Total MMZ (reserved)	114688 KB
Kernel	Total kernel	34576 KB
Android	Available memory for Android (total)	385700 KB
	Android runtime	122880 KB
	Available memory for applications	262820 KB

The statistics in Table 20-2 are based on the HiSilicon standard SDK and is for reference only. The statistics may change when users modify or add a third-party Android package (APK) based on the actual requirements.

20.2.1.3 Version Test Result

Table 20-3 describes the test result of the 512 MB FHD version.

Table 20-3 Test result of the 512 MB FHD version

Typical Application Scenario	Free RAM (KB)	Used RAM (KB)	Used MMZ (MB)
Original launcher	322713	109806	46
One-channel 1080p stream playback	219624	149830	83
One-channel 1080p 2D Blu-ray playback	204279	161029	84
One-channel 1080p SBS playback	208625	155657	84
Game (Riptide GP 2)	104072	197406	46

20.2.2 512 MB UHD Version

20.2.2.1 Detailed Specifications

Table 20-4 lists the detailed specifications for the 512 MB UHD version (other specifications are the same as those of the 1 GB version).

Table 20-4 Detailed specifications of the 512 MB UHD version

Specifications	Description
UI	720p UI supported
Video decoding capability	4K@30 fps, with a maximum of 100 Mbit/s bit rate
3D	1080p SBS, TAB, and MVC supported
4K	4K decoding supported
PIP	One 1080p and one 720p PIP supported
Video encoding capability	1080p@30 fps H.264 encoding supported
CTS	Not supported

Specifications	Description
DRM	Not supported
fastplay	Supported, 1080p 2D streams
Static wallpaper	Supported (disabled by default)
Dynamic wallpaper	Not supported
PQ	Not supported
DLNA	DMP or DMR supported. DLNA is not supported in scenarios where the playback resources in the background are not released.
airplay	Not supported
Multi-screen	Not supported
Transcode	Not supported
Video communication	Supported (maximum 1080p)
IPTV application	ANDROIDMOV, Wasu, BestTV, and CNTV supported
Browser	Supported (with low compatibility). The browser exits abnormally when the occupied memory exceeds 150 MB.
3D game	Supported (with low compatibility). The 3D game exits abnormally when the occupied memory exceeds 150 MB.

20.2.2.2 Compilation and Configuration of the 512 MB UHD Version

The FHD version is supported by the 512 MB version by default. To change the FHD version to the UHD version with 4K decoding capability, perform the following steps:

NOTE

The following section takes Hi3798M V100 chip as an example.

Step 1 Find the SDK configuration file **SDK_CFGFILE**, which is defined in the **BoardConfig.mk** file in the directory **device\hisilicon\Hi3798MV100** as follows:

- If it is a level-2 security chip, the SDK configuration file is **hi3798mdmo_hi3798mv100_android_security_cfg.mak**.
- Otherwise, the SDK configuration file is **hi3798mdmo_hi3798mv100_android_cfg.mak**.

Step 2 Modify the configuration of the decoding capability in the SDK configuration file under **device\hisilicon\bigfish\sdk\configs\hi3798mv100**.

Before modification:

```
# CFG_HI_VIDEO_PRE_CAP_MVC_IN_512 is not set
CFG_HI_VIDEO_PRE_CAP_1080P_IN_512=y
# CFG_HI_VIDEO_PRE_CAP_2160P_IN_512 is not set
# CFG_HI_VIDEO_PRE_CAP_4K_IN_512 is not set
CFG_HI_VIDEO_MAX_REF_FRAME_NUM_IN_512=6
CFG_HI_VIDEO_MAX_DISP_FRAME_NUM_IN_512=2
CFG_HI_VIDEO_SCD_BUF_SIZE_IN_512=2
CFG_HI_VIDEO_MAX_VDH_BUF_IN_512=60
```

After modification:

```
# CFG_HI_VIDEO_PRE_CAP_MVC_IN_512 is not set
# CFG_HI_VIDEO_PRE_CAP_1080P_IN_512 is not set
# CFG_HI_VIDEO_PRE_CAP_2160P_IN_512 is not set
CFG_HI_VIDEO_PRE_CAP_4K_IN_512=y
CFG_HI_VIDEO_MAX_REF_FRAME_NUM_IN_512=3
CFG_HI_VIDEO_MAX_DISP_FRAME_NUM_IN_512=3
CFG_HI_VIDEO_SCD_BUF_SIZE_IN_512=10
CFG_HI_VIDEO_MAX_VDH_BUF_IN_512=140
```

Step 3 Modify the bootargs configuration file under the chip directory **device\hisilicon\Hi3798MV100\etc**. The files are as follows:

- **bootargs_Hi3798MV100-emmc.txt** (configuration file for the eMMC)
- **bootargs_Hi3798MV100-nand.txt** (configuration file for the NAND)

Before modification:

```
bootargs_512M=mem=512M mmz=ddr,0,0,112M
```

After modification:

```
bootargs_512M=mem=512M mmz=ddr,0,0,200M
```

NOTE

The size of the reserved MMZ must be an integral multiple of 4 MB.

This configuration supports the decoding of steams with a maximum of six reference frame buffers. Some streams may need seven reference frame buffers, which can be met by adjusting the size of the MMZ to implement the decoding. You are advised to adopt the configuration with seven reference frame buffers when the memory demand of the application is not harsh. The modification method is as follows:

1. Modify the SDK configuration file as follows:

   ```
   CFG_HI_VIDEO_MAX_REF_FRAME_NUM_IN_512=3
   CFG_HI_VIDEO_MAX_DISP_FRAME_NUM_IN_512=3
   CFG_HI_VIDEO_SCD_BUF_SIZE_IN_512=10
   CFG_HI_VIDEO_MAX_VDH_BUF_IN_512=150
   ```

2. Adjust the MMZ size by modifying the bootargs configuration file under the chip directory.

Before modification:

```
bootargs_512M=mem=512M mmz=ddr,0,0,200M
```

After modification:

```
bootargs_512M=mem=512M mmz=ddr,0,0,212M
```

CAUTION

The recommended decoding configuration for all low-memory default versions and the 512 MB UHD version is the stable configuration that has been repeatedly tested by HiSilicon. You are advised not to change such configuration. If you modify the decoding configuration based on the preceding instructions, you need to verify the stability of the configuration by yourself.

----End

20.2.2.3 Memory Usage

The 512 MB UHD version supports UHD, H.265, and SBS/TAB/MVC, and does not support 4K output and CTS. Its 4K decoding and application compatibility is limited.

The detailed decoding configuration is as follows:

- Six decoding frame buffers, with three buffers for storing reference frames and three buffers for storing buffer frames
- Six VPSS frame buffers
- 10 MB stream buffer
- 200 MB reserved memory for MMZ
- 160 MB memory available for applications

Table 20-5 lists the detailed memory usage.

Table 20-5 Memory usage

Item		**UHD 4K Decoding (512 MB)**
RAM	Total RAM	500212 KB
MMZ	Base memory	13812 KB
	One-channel 4K playback	166000 KB
	720p FB	10440 KB
	Total MMZ (theoretically)	190252 KB
	Total MMZ (reserved)	204800 KB
Kernel	Total kernel	34576 KB
Android	Available memory for Android (total)	295412 KB

Item		UHD 4K Decoding (512 MB)
	Android runtime	122880 KB
	Available memory for applications	172532 KB

The statistics in Table 20-5 are based on the HiSilicon standard SDK and is for reference only. The statistics may change when users modify or add a third-party APK based on the actual requirements.

20.2.2.4 Version Test Result

Table 20-6 describes the test result of the 512 MB UHD version.

Table 20-6 Test result of the 512 MB UHD version

Typical Application Scenario	Free RAM (KB)	Used RAM (KB)	MMZ Used (MB)
One-channel 1080p stream playback	162685	84654	161
One-channel 2D Blu-ray playback	163085	84315	161
One-channel 3D Blu-ray playback	141350	76342	185
One-channel 3D SBS playback	163345	84217	161
One-channel 4K playback	150988	80463	175
Game (Riptide GP 2)	116120	100456	113

20.2.3 768 MB 4K UHD Version

20.2.3.1 Detailed Specifications

Table 20-7 lists the detailed specifications for the 768 MB 4K UHD version (other specifications are the same as that of the 1 GB version).

Table 20-7 Detailed specifications of the 768 MB 4K UHD version

Specifications	Description
Video decoding format	Same as that of the 1 GB version
Video decoding capability	4K@30 fps, with a maximum of 100 Mbit/s bit rate

Specifications	Description
3D	1080P SBS, TAB, and MVC supported
4K	4K output supported
PIP	Two 1080p PIPs supported
Video encoding capability	1080p@30 fps H.264 encoding supported
CTS	Not supported
DRM	Not supported
fastplay	Supported, 1080p 2D streams
Static wallpaper	Supported (disabled by default)
Dynamic wallpaper	Not supported
DLNA	DMP and DMR supported. DLNA is not supported in scenarios where the playback resources in the background are not released.
airplay	Not supported
multiscreen	Not supported
transcode	Not supported
Video communication	Supported (maximum 1080p)
IPTV application	ANDROIDMOV, Wasu, BestTV, and CNTV are supported.
Browser	Supported. The browser exits abnormally when the occupied memory exceeds 280 MB.
3D game	Supported. The 3D game exits abnormally when the occupied memory exceeds 280 MB.

20.2.3.2 Boot Table Replacing

Because the hardware supports only the design with two DDR SDRAMs and the 768 MB UHD version uses asymmetric memory, the boot table needs to be replaced when the 768 MB 4K UHD version is compiled. Take the Hi3798M V100MODE1A board as an example:

The CFG and REG names of the eMMC and NAND are defined in the **BoardConfig.mk** file in **device\hisilicon\Hi3798MV100**. For example, the eMMC is defined as follows:

```
EMMC_BOOT_CFG_NAME :=
```

```
hi3796mdmo1a_hi3796mv100_ddr3_1gbyte_16bitx2_4layers_emmc.cfg
EMMC_BOOT_REG_NAME :=
hi3796mdmo1a_hi3796mv100_ddr3_1gbyte_16bitx2_4layers_emmc.reg
```

In the CFG and REG files under the directory **device\hisilicon\bigfish\sdk\source\boot\sysreg**, find the corresponding table file for the 768 MB Hi3798M V100MODE1A board, change the preceding variables to:

```
EMMC_BOOT_CFG_NAME :=
hi3798mdmo1a_hi3798mv100_ddr3_768mbyte_16bitx2_4layers_emmc.cfg
EMMC_BOOT_REG_NAME :=
hi3798mdmo1a_hi3798mv100_ddr3_768mbyte_16bitx2_4layers_emmc.reg
```

20.2.3.3 Memory Usage

The 768 MB UHD version supports UHD, H.265, SBS/TAB/MVC, 4K decoding, and 4K output. It does not support CTS, and the application compatibility is limited.

The detailed decoding configuration is as follows:

- Eight decoding frame buffers, with five buffers for storing reference frames and three buffers for storing buffer frames
- Six VPSS frame buffers
- 20 MB stream buffer
- 300 MB reserved memory for MMZ
- 300 MB memory available for applications

Table 20-8 lists the detailed memory usage.

Table 20-8 Memory usage

Item		UHD 4K Specifications (768 MB)
RAM	Total RAM	759708 KB
MMZ	Base memory	13812 KB
	One-channel 4K playback	256354 KB
	720p FB	10440 KB
	Total MMZ (theoretically)	280606 KB
	Total MMZ (reserved)	307200 KB
Kernel	Total kernel	34576 KB
Android	Available memory for Android (total)	452508 KB
	Android runtime	143360 KB
	Available memory for applications	309148 KB

The statistics in Table 20-8 are based on the HiSilicon standard SDK and is for reference only. The statistics may change when users modify or add a third-party APK based on the actual requirements.

20.2.3.4 Version Test Result

Table 20-9 describes the test result of the 768 MB UHD version.

Table 20-9 Test result of the 768 MB UHD version

Typical Application Scenario	Free RAM (KB)	Used RAM (KB)	USED MMZ (MB)
Original launcher	457119	114966	138
One-channel 1080p stream playback	332714	166991	212
One-channel 2D Blu-ray playback	335415	166737	212
One-channel 3D Blu-ray playback	308078	168640	235
One-channel 3D SBS playback	333333	156911	219
One-channel 4K playback	347318	156193	263
Game (Riptide GP 2)	280226	252338	143

20.2.4 Customizable Specifications

Some specifications are supported in the solution, but are disabled by default in the delivery version. You can enable or disable these customizable specifications based on the requirements as well as the modification methods and memory evaluation in this document. The customizable specifications include the following:

- Static wallpaper
- Fast boot

20.2.4.1 Static Wallpaper

The low-memory version determines whether to load the wallpaper service based on the read attribute **persist.low_ram.wp.enable**. This attribute is defined in the **customer.mk** file under the chip directory. The default value is **false**, which means that the wallpaper service is not loaded. You can set the attribute to true to enable the wallpaper service.

The service is loaded by the systemServer and resides in the system. The memory usage is about 20 MB. For the 512 MB version with insufficient memory space, more benefits can be obtained with the attribute disabled.

20.2.4.2 Fast Boot

Similar to the static wallpaper specifications, whether to compile the fast boot depends on the attribute **BOARD_QBSUPPORT** during compilation. Its default value is **false**. You can set the value to **true** by modifying the **customer.mk** file in the chip directory to compile a fast boot version.

Theoretically, the low-memory version does not conflict with the fast boot version, and they can be compatible with each other. However, when the fast boot is enabled, extra 5 MB memory is occupied in the kernel.

20.2.5 Memory Statistics

20.2.5.1 Memory Checking

NOTE

The following section takes the Hi3798M V100 768 MB UHD version as an example.

- Check the status of the system memory by running **cat /proc/meminfoon** over the serial port.

Figure 20-1 System memory status

```
root@Hi3798MV100:/ # cat proc/meminfo
MemTotal:         759708 kB
MemFree:           92404 kB
Buffers:            2408 kB
Cached:           253320 kB
SwapCached:            0 kB
Active:           104032 kB
Inactive:         245436 kB
```

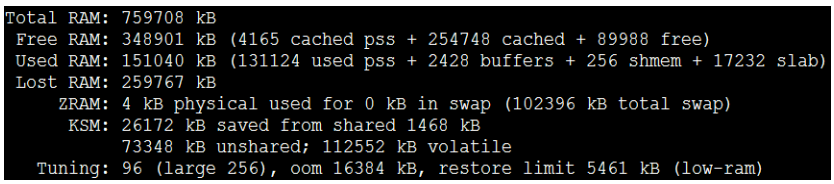
- Check the status of the pre-allocated memory by running **cat /proc/media-mem** over the serial port.

Figure 20-2 Pre-allocated memory status

```
----------------------------------------------------
|Summary:
----------------------------------------------------
|  MMZ Total Size  |  MMZ Used  | ION Used  |  Idle  |
----------------------------------------------------
|       300MB         214MB          0MB         85MB
----------------------------------------------------
```

- Check the status of the total memory and memories of individual applications by running **dumpsys meminfo** over the serial port.

Figure 20-3 Status of the total memory and memories of individual applications



- Check the memory usage of the graphic layer by running **dumpsys SurfaceFlinger**.

20.2.5.2 Memory Statistics Based on dumpsys meminfo

- Available memory for the Android system = Total RAM − MMZ
- Android runtime memory = Used RAM − Visible − A Service − B Service − Previous – Foreground + cache resident space + waterline reserved space + Wi-Fi memory

  Visible, A Service, B Service, Previous, and Foreground are the specific memories of the system, which are displayed by running **dumpsys meminfo**. If certain item does not exist, the corresponding value is 0.
- Available memory for APPs = Available memory for the Android system − Android runtime
- Cache resident space = 20 MB (empirical value)
- Waterline reserved space = 4 MB
- Wi-Fi memory = 6 MB (empirical value)

20.2.6 Fault Location and Debugging

For the low-memory version, the decoding specifications are tailored. As a result, some streams are not decoded. During debugging and testing, playback failure occurs due to insufficient decoding capability. To make it easier for you to locate such an issue and distinguish it from other issues, a log mechanism is provided for the system. When the playback failure is caused by insufficient decoding capability, a clear message is displayed in the log.

Locating method: When stream playback fails, run **logcat -s HI_VDEC** over the serial port. If the following information is displayed, the playback failure is caused by insufficient decoding capability.

```
E/HI_VDEC ( 1357): [1260234219 ERROR-HI_VDEC]:VFMW_CheckEvt[1217]:Over
Capability of Reserve Memory!!!
```

21 Memory Optimization of the 1 GB Version

21.1 Overview

The memory of the 1 GB version is optimized to provide more available memory for the application.

The optimized memory version has tailored the standard specifications of the 1 GB version. The restrictions are as follows:

- CTS is not supported.
- Dynamic switching between the 720p UI and 1080p UI is not supported.
- Dynamic wallpaper is not supported.

21.2 Android System Memory

The **ro.config.low_ram** attribute is used to determine whether to enable the memory optimization function of the 1 GB version.

- **ro.config.low_ram** is defined in **device/hisilicon/Hi37xxMV100/customer.mk**.
- When **ro.config.low_ram** is **true**, the preceding function is enabled.
- When **ro.config.low_ram** is **false**, the preceding function is disabled. The default value is **false**.

The optimized memory version reads the **persist.low_ram.wp.enable** attribute to determine whether to load the wallpaper service.

- **persist.low_ram.wp.enable** is defined in **device/hisilicon/Hi37xxMV100/customer.mk**.
- When **persist.low_ram.wp.enable** is **true**, the wallpaper service is loaded.
- When **persist.low_ram.wp.enable** is **false**, the wallpaper service is not loaded. The default value is **false**.

The application blacklist **limitApplications.xml** can be modified as required. In the optimized memory version, the APKs in the blacklist will not be installed and used by the system (**system/etc**).

- **limitApplications.xml** is located in **device/hisilicon/Hi37xxMV100/etc/**.

- To add or delete an APK, modify the content in the <key-1g> tag.

  For example:

  If the MultiScreen function does not need to be supported, add the following content between <key-1g> and </key-1g>.

  ```
  <apkName>MultiScreenServer.apk</apkName>
  <apkName>HandsetForMultiScreen.apk</apkName>
  <apkName>HiMediaShare.apk</apkName>
  ```

21.3 MMZ

The default size of the reserved MMZ in the 1 GB version of Hi3798M V100/Hi3796M V100 is 435 MB. The MMZ is used as follows:

- 13962 KB for 1-channel 720p transcoding
- 14058 KB for 1-channel 720p transcoding
- 119700 KB MMZ for the 1080p UI (27540 KB FB and 92160 KB surface), and 102600 KB MMZ for the 720p UI (10440 KB FB and 92160 KB surface)

The possible scenarios are as follows:

- When mirroring, transcoding, and 1080p UI are not used and the 720p UI is used:

  The MMZ can be reduced by 45120 KB (44.0625 MB). It is recommended that the MMZ of the 1 GB version be set to 390 MB.
- When mirroring and transcoding are not used and the 1080p UI is used:

  The MMZ can be reduced by 28020 KB (27.3633 MB). It is recommended that the MMZ of the 1 GB version be set to 405 MB.
- TDE compose is enabled to improve the graphics refresh performance.

  The TDE compose function requires the MMZ. Therefore, when the GPU calls Gralloc, Gralloc instructs the ION driver to allocate the MMZ memory.

  The TDE compose function is enabled by default. If the TDE compose function is disabled, Gralloc does not use the MMZ and uses the Android system memory instead. In this case, the size of the reserved MMZ can be reduced by canceling the MMZ reserved for the GPU surface.
  - The **ro.config.tde_compose** attribute is used to determine whether to enable the TDE compose function. **ro.config.tde_compose** is defined in **device/hisilicon/Hi37xxMV100/customer.mk**.
    - When **ro.config.tde_compose** is **true**, the TDE compose function is enabled. The default value is **true**.
    - When **ro.config.tde_compose** is **false**, the TDE compose function is disabled.
  - If the TDE compose function is disabled (**ro.config.tde_compose** = **false**), the MMZ can be reduced by 92160 KB (90 MB).
- The preceding reduced size of the MMZ is calculated theoretically. After the MMZ is modified, you are advised to test the 4K H.265 playback and 3D MVC playback scenarios to check whether exceptions such as playback intermittence occur.

A Appendix

Table A-1 describes ADB commands.

Table A-1 ADB commands

Category	Command	Description	Remarks
Options	-d	Manages the ADB only over the USB port.	If the ADB is managed in multiple ways including the USB port, an error code is returned. Example: `adb -d shell`
	-e	Manages the ADB only using the emulator instance.	If the ADB is managed in multiple ways including the emulator instance, an error code is returned. Example: `adb -e shell`
	-s <serialNumber>	Manages the ADB by sending a command with the allowed serial number of an emulator or a device (for example, emulator-5556).	If no serial number is specified, an error occurs. Example: `adb -s shell`
General	devices	Views the list of all connected emulators or devices.	Example: `adb devices`
	help	Views all commands supported by the ADB.	Example: `adb help`
	version	Views the version number of the ADB.	Example: `adb version`
Debug	logcat [<option>] [<filter-specs>]	Displays log data on the screen.	Example: `adb logcat MyApp:D *:S`
	bugreport	Views the bug report including the dumpsys, dumpstate, and logcat information.	Example: `adb bugreport`
	jdw	Views the available Java	You can use the forward jdwp:<pid> port-

Category	Command	Description	Remarks
		Debug Wire Protocol (JDWP) of a specific device.	forwarding specification to connect to a specific JDWP process. Example: `adb forward tcp:8000 jdwp:472 jdb -attach localhost:8000`
Data	install <path-to-apk>	Pushes an Android application (specified as a full path to an .apk file) to the data file of an emulator or a device.	Example: `adb install F:\WishTV.apk`
	pull <remote> <local>	Copies a specific file from an emulator or a device to the computer.	Example: `adb pull system/app F:\`
	push <local> <remote>	Copies a specific file from the computer to an emulator or a device.	Example: `push F:\WishTV. apk /system/app`

Ports and Networking	forward <local> <remote>	Forwards socket connections from a specified local port to a specified remote port on the emulator or device instance.	The port information is as follows:
			`tcp:<portnum>`
			`local:<UNIX domain socket name>`
			`dev:<character device name>`
			`jdwp:<pid>`

Scripting	get-serialno	Views the serial number of the ADB instance.	Example: `adb get-serialno` `adb get-state`
	get-state	Views the current status of an emulator or a device.	
	wait-for-device	Blocks execution until the device is online, that is, until the instance state is **device**.	Commands can be mounted in the ADB commander in advance. The commands in the commander are executed only after an emulator or a device is connected. Example: `adb wait-for-device shell getprop` Note that the ADB is not started until the system starts. Therefore, commands are executed after the system starts. For example, the Android package is required when you run the **install** command. The package is available only after the system starts. Example: `adb wait-for-device install <app>` The preceding command is executed only after an emulator or a device connects to the ADB server and the Android system starts. If the command is executed before the system starts, an error occurs.

Category	Command	Description	Remarks
Server	start-server	Checks whether the ADB server process is running.	Example: `adb start-server`
	kill-server	Terminates the ADB server process.	Example: `adb kill-server`
Shell	shell	Controls an emulator or a device by running a remote shell command.	Example: `adb shell` `adb shell rm system/app/WishTV.apk`
	shell[<shellCommand>]	Issues a shell command in the target emulator/device instance and then exits the remote shell.	

Table A-2 describes common logcat commands.

Table A-2 Common logcat commands

Option	Description
-b <buffer>	Loads an available log buffer to view information. The buffers include event, radio, and main. The main buffer is used by default
-c	Clears the logs displayed on the screen.
-d	Displays logs on the screen.
-f <filename>	Specifies the <filename> of the log to be displayed. filename: stdout (default), stderr
-g	Displays the size of the specified log buffer and exits.
-n <count>	Sets the maximum number of logs to <count>. The default value is **4**. This command must be executed with the **-r** command.
-r <kbytes>	Outputs logs every <kbytes> of output. The default value is **16**. This command must be executed with the **-f** command.
-s	Sets the default filter level to **silent**.
-v <format>	Sets the log input format. The default format is **brief**. The supported formats include brief, process, tag, thread, raw, time, and long.

Table A-3 describes some ADB shell commands.

Table A-3 ADB shell commands

Shell Command	Description	Remarks
dumpsys	Clears the system data displayed on the screen. Example: `adb shell dumpsys`	The DDMS provides comprehensive debugging function.
dumpstate	Clears the status of a file. Example: `adb shell dumpstate`	
logcat [<option>]...[<filter-spec>]...	Enables radio logging and prints output to the screen. Example: `adb shell logcat`	
dmesg	Displays the major debugging information on the screen. Example: `adb shell dmesg`	
start	Starts or restarts an emulator or a device. Example: `adb shell start`	N/A
stop	Stops execution of an emulator or a device. Example: `adb shell stop`	N/A

Table A-4 describes common Monkey commands.

Table A-4 Common Monkey commands

Category	Option	Description
General	--help	Prints a simple usage guide.
	-v	Each -v on the command line will increment the verbosity level. Level 0 (the default) provides little information beyond startup notification, test completion, and final results. Level 1 provides more details about the test as it runs, such as individual events being sent to your activities. Level 2 provides more detailed setup information such as activities selected or not selected for testing.
Events	-s <seed>	Seed value for pseudo-random number generator. If you re-run the Monkey with the same seed value, it will generate the same sequence of events.
	--throttle <milliseconds>	Inserts a fixed delay between events. You can use this option to slow down the Monkey. If not specified, there is no delay and the events are generated as rapidly as possible.
	--pct-touch <percent>	Adjust percentage of touch events. (Touch events are a down-up event in a single place on the screen.)
	--pct-motion <percent>	Adjust percentage of motion events. (Motion events consist of a down event somewhere on the screen, a series of pseudo-random movements, and an up event.)
	--pct-trackball <percent>	Adjust percentage of trackball events. (Trackball events consist of one or more random movements, sometimes followed by a click.)

Category	Option	Description
	--pct-nav <percent>	Adjust percentage of "basic" navigation events. (Navigation events consist of up/down/left/right, as input from a directional input device.)
	--pct-majornav <percent>	Adjust percentage of "major" navigation events. (These are navigation events that will typically cause actions within your UI, such as the center button in a 5-way pad, the back key, or the menu key.)
	--pct-syskeys <percent>	Adjust percentage of "system" key events. (These are keys that are generally reserved for use by the system, such as Home, Back, Start Call, End Call, or Volume controls.)
	--pct-appswitch <percent>	Adjust percentage of activity launches. At random intervals, the Monkey will issue a startActivity() call, as a way of maximizing coverage of all activities within your package.
	--pct-anyevent <percent>	Adjust percentage of other types of events. This is a catch-all for all other types of events such as keypresses, other less-used buttons on the device, and so forth.
Constraints	-p <allowed-package-name>	If you specify one or more packages this way, the Monkey will *only* allow the system to visit activities within those packages. If your application requires access to activities in other packages (for example, to select a contact) you'll need to specify those packages as well. If you do not specify any packages, the Monkey will allow the system to launch activities in all packages. To specify multiple packages, use the -p option multiple times — one -p option per package.
	-c <main-category>	If you specify one or more categories this way, the Monkey will *only* allow the system to visit activities that are listed with one of the specified categories. If you do not specify any categories, the Monkey will select activities listed with the category Intent. CATEGORY_LAUNCHER or Intent. CATEGORY_MONKEY. To specify multiple categories, use the -c option multiple times — one -c option per category.
Debugging	--dbg-no-events	When specified, the Monkey will perform the initial launch into a test activity, but will not generate any further events. For best results, combine with -v, one or more package constraints, and a non-zero throttle to keep the Monkey running for 30 seconds or more. This provides an environment in which you can monitor package transitions invoked by your application.
	--hprof	If set, this option will generate profiling reports immediately before and after the Monkey event sequence. This will generate large (–5Mb) files in data/misc, so use with care. See Traceview for more information on trace files.
	--ignore-crashes	Normally, the Monkey will stop when the application crashes or experiences any type of exception not handled. If you specify this option, the Monkey will continue to send events to the system, until the count is completed.

Category	Option	Description
	--ignore-timeouts	Normally, the Monkey will stop when the application experiences any type of timeout error such as an "Application Not Responding" dialog. If you specify this option, the Monkey will continue to send events to the system, until the count is completed.
	--ignore-security-exceptions	Normally, the Monkey will stop when the application experiences any type of permissions error, for example if it attempts to launch an activity that requires certain permissions. If you specify this option, the Monkey will continue to send events to the system, until the count is completed.
	--kill-process-after-error	Normally, when the Monkey stops due to an error, the application that failed will be left running. When this option is set, it will signal the system to stop the process in which the error occurred. Note, under a normal (successful) completion, the launched process(es) are not stopped, and the device is simply left in the last state after the final event.
	--monitor-native-crashes	Watches for and reports crashes occurring in the Android system native code. If --kill-process-after-error is set, the system will stop.
	--wait-dbg	Stops the Monkey from executing until a debugger is attached to it.