Android Solution

Development Guide

Issue 09

Date 2015-06-25

HiSilicon Technologies Co., Ltd.

Address: Huawei
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 functions, common interfaces, and working principles of the Android solution, as well as the development process of each module and precautions to be taken during development by using samples.

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 (2015-06-25)

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

Section 2.3 is modified, and Section 8.4 is added.

Issue 08 (2015-03-31)

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

Chapter 9 is added.

Issue 07 (2015-01-15)

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

Chapter 2 Development Environment Configuration

Step 3 and step 4 in section 2.2.2 are modified.

Chapter 7 HiTranscoder

Step 8 in the "Compiling libHiTransPlayer_jni" section is modified, and a caution is added to step 4 in the "Compiling the Client Software in the HiTransPlayer Directory" section.

Issue 06 (2014-11-05)

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

Sections 4.4.2 and 4.4.3 are added.

Sections 2.3, 3.2, and 7.4.2.2 are modified.

Issue 05 (2014-09-30)

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

Chapter 6 is modified.

Chapter 8 is added.

Issue 04 (2014-08-14)

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

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 change:

Chapter 6 HiMultiScreen

Section 6.4.4.3 is added.

Issue 02 (2014-03-13)

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

Chapters 5 and 7 are modified.

Issue 01 (2013-12-13)

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

Chapter 5 HiDLNA

Section 5.4.1.1 is added.

Chapter 7 HiTranscoder

Section 7.4.2 is added.

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

Issue 00B01 (2013-08-30)

This issue is the first draft release.

Contents

Figures

Tables

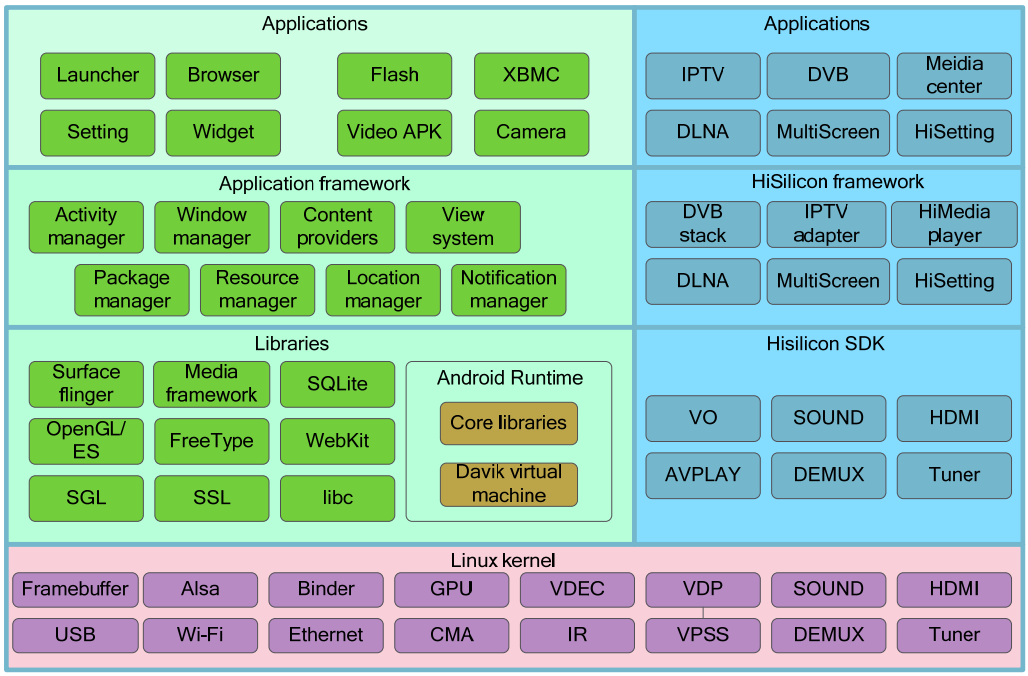
1 Overview

1.1 Overall Architecture of the Android Solution

The overall architecture of the HiSilicon Android solution consists of two parts: Android basic architecture and HiSilicon extended architecture.

The Android inherent design and APIs are retained in the Android basic architecture. Only a few functions are added or modified. The HiSilicon extended architecture provides HiSilicon applications and APIs based on the HiSilicon SDK. Some functions are implemented by using the UNF interfaces, and some functions are implemented by using the interfaces of the Java native interface (JNI) layer and Java layer. Figure 1-1 shows the overall architecture.

Figure 1-1 Overall architecture of the Android solution



The HiSilicon solution retains the Android inherent architecture and decouples the extended functions and APIs from the Android architecture to the greatest extent to ensure the quick upgrade of the Android system and the stability of the extended APIs and improve the

compatibility of Android applications. If the customized functions are based on the HiSilicon extended architecture, the Android basic architecture changes little when the Android system is upgraded, the HiSilicon solution supports quick upgrade version, and the HiSilicon APIs in the extended architecture change little. Therefore the customized functions can be inherited to the greatest extent.

The Android basic architecture consists of the following layers:

- Application layer

  Android inherent application layer. The open-source applications such as the flash, browser (HTML5 video), video client, and XBMC are supported.
- Framework layer

  Android inherent framework layer. A few modifications are made by HiSilicon to support some extended functions such as the wireless network and Point-to-Point Protocol over Ethernet (PPPoE).
- Library layer

  Android inherent library layer. Except for the support of basic inherent libraries, the adaptation of the hardware abstraction layer (HAL) is also implemented above the kernel, including the adaptation of the OpenMAX, Alsa, graphics processing unit (GPU), and camera.

The HiSilicon extended architecture consists of the following layers:

- Application layer

  This layer implements applications such as the file browser, media player, and HiSetting developed by HiSilicon. Customer applications such as the IPTV, DVB, and MultiScreen can also be implemented at this layer.
- Framework layer

  HiSilicon extended framework such as the HiMediaPlayer, DLNA, MultiScreen, DisplaySetting, and AudioSetting. You can also develop your own extended framework at this layer based on the HiSilicon SDK.
- SDK

  HiSilicon UNF API implementation layer, implementing the hardware and driver encapsulation.

1.2 Functions Modified and Added by HiSilicon

1.2.1 Android Basic Architecture

The following describes the functions modified and added by HiSilicon:

- Modified or adapted functions:
  - Media framework extension

    Involved modules: media framework, view system
  - Device mounting and unmounting

    Involved modules: VOLD, MountService
  - Volume adjustment

    Involved module: audio framework
  - Upgrade over the USB flash drive

Involved module: Recovery

- HAL adaptation

  Involved modules: OpenMAX, GPU, Alsa, camera, Wi-Fi driver, PowerManager

- Added functions:
  - Wireless network

    Involved modules: Ethernet, Setting
  - PPPoE

    Involved modules: PPPoE, Setting
  - File system extension (UBI, NTFS)

    Involved modules: VOLD, Recovery

1.2.2 HiSilicon Extended Architecture

- Java layer APIs provided by HiSilicon
  - HiMediaPlayer
  - HiDLNA
  - HiMultiScreen
- C/C++ layer APIs provided by HiSilicon
  - HiTranscoder
  - HiSilicon SDK UNF APIs (See the *HMS Development Guide*)
- New features
  - Display configuration: display resolution and display region adjustment, picture configuration
  - Audio configuration: HDMI output, SPDIF output, blu-ray degradation output
  - Video configuration: video aspect ratio
  - Multi-screen interaction: HiDLNA, HiMultiScreen, HiTranscoder, and Miracast
  - Video playback: HiMediaPlayer
  - Application: media center, media (blu-ray 3D) player, NFS, Samba, HiAnimation

2 Development Environment Configuration

This chapter describes how to set up the Android development environment and compile and burn images on the Linux server end. For details about how to set up the Android development environment on the PC end, see the Google developer website:

http://developer.android.com/sdk/index.html

NOTE

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

2.1 Requirements on the Android Development Environment

The following describes the requirements on the Android development environment:

- Operating system (OS): 64-bit Ubuntu 10.04
- Available hard disk space: at least 100 GB
- Python version: 2.4–2.7
- JDK version: 1.6
- Cross compilation tool chain: arm-hisiv200-linux-

CAUTION

If Ubuntu of other versions is used, incompatibility may occur. Ubuntu10.04 is proved to be the most stable version for developing the Android.

The OS must be 64-bit for the development of Android (2.3x) Gingerbread or later.

2.2 Setting Up a Development Environment

2.2.1 Automatic Configuration

The automatic script (**ServerInstall.sh**) for installing the system environment and cross compilation tool is provided in the **SDK/Software/ServerInstall** directory. You need to run this script in the directory where the script locates to implement automatic configuration as the root user:

```
Software/ServerInstall$sudo ./ServerInstall.sh
```

CAUTION

Connect the server to the Internet before running this script.

Run the script in the directory where this script locates.

Run the script as the root user.

User interaction is required during script execution.

If the script execution fails, run this script again after resolving problems by following the instructions, or manually configure the development environment based on the following section.

2.2.2 Manual Installation

To manually configure the development environment, perform the following steps:

Step 1 Add the upgrade sources.

Ensure that the network connection is normal and add installation sources.

```
$ sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu hardy
main multiverse"
$ sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu hardy-
updates main multiverse"
```

Or edit **sources.list**:

```
$ vim /etc/apt/sources.list
```

Add the following contents at the end of **sources.list** and save it:

```
deb http://archive.ubuntu.com/ubuntu hardy-updates main multiverse
deb http://archive.ubuntu.com/ubuntu hardy main multiverse
```

NOTE

The preceding websites are only for reference. Check whether the **hardy** and **hardy-updates** directories exist at *http://archive.ubuntu.com/ubuntu*. This website is updated in real time. If the directories do not exist, add the link that contains the two directories.

Step 2 Update the installation package.

```
$ sudo apt-get update
```

Step 3 Install the tools.

```
$ sudo apt-get install git-core gnupg flex bison gperf \
build-essential zip curl zlib1g-dev libc6-dev lib32ncurses5-dev \
ia32-libs x11proto-core-dev libx11-dev lib32readline-gplv2-dev \
lib32z1-dev libgl1-mesa-dev g++-multilib mingw32 tofrodos \
python-markdown libxml2-utils xsltproc python uboot-mkimage lzma
```

NOTE

- The preceding tool packages are only for reference. For details about the tool packages on which each Android version depends, visit source.android.com/source/initializing.html. Information on this website is updated in real time.
- **uboot-mkimage** and **lzma** must be installed because they are the tool packages on which the HiSilicon solution depends.

Step 4 Install JDK 1.6.

```
$ sudo apt-get install sun-java6-jdk
```

NOTE

The Ubuntu installation source does not provide the JDK installation package any more. You can download the corresponding JDK from www.oracle.com. For details about the mapping between the JDK and Android version, see source.android.com/source/initializing.html.

You are advised to perform the operations by following instructions.

If there are several JDK versions, switch to the specific version by running the following command:

```
$ sudo update-alternatives --config java
```

There are two manual modes for replacing the current Java version (**/usr/bin/java** is provided).

Table 2-1 Replacing the current Java version in manual mode

Option	Path	Priority	Status
0	/usr/lib/jvm/java-6-openjdk/jre/bin/java	1061	Automatic mode
1	/usr/lib/jvm/java-1.5.0-sun/jre/bin/java	53	Manual mode
2	/usr/lib/jvm/java-6-openjdk/jre/bin/java	1061	Manual mode

Press **Enter** to retain the current setting **[*]**, enter a number such as **1**, or edit **/etc/profile** manually by running the following command:

```
$ sudo vim /etc/profile
```

Add the following contents at the end of **/etc/profile** and save it**:**

```
export JAVA_HOME="/usr/lib/jvm/java-1.6.0-sun"
export PATH="/usr/lib/jvm/java-1.6.0-sun/bin":$PATH
```

The JDK path varies according to the version of the installed JDK.

Step 5 Verify the JDK version number.

```
$ java -version
The version number of JDK 1.6 is as follows:
java version "1.6.0_26"
Java(TM) SE Runtime Environment (build 1.6.0_26-b03)
Java HotSpot(TM) 64-Bit Server VM (build 20.1-b02, mixed mode)
```

CAUTION

If the JDK 1.6 version is earlier than JDK 1.6.0_24, for example, if the version is JDK 1.6.0_19, upgrade the JDK to JDK 1.6.0_24 or later; otherwise, errors may occur during compilation.

Step 6 Install the cross compilation tool chain.

CAUTION

Install the cross compilation tool chain only when you obtain the release package for the first time or when the cross compilation chain in the release package needs to be upgraded.

You need to install the cross compilation tool chain as the root user. Before running the installation command, ensure that the shell on the server is bash, because the SDK can work only under the bash. If the shell on the server is not bash, you are advised to uninstall dash or change the default shell to bash. To be specific, you can delete the soft link pointing to the existing shell, and recreate a soft link pointing to the bash as follows:

$ cd /bin

$ rm –f sh

$ ln –s /bin/bash /bin/sh

When you install the cross compilation tool chain, the existing compiler with the same name is replaced. If you modified the HiSilicon compiler that you used before, the modifications may be invalid. Therefore, if the compilation was modified, back up the cross compilation tool chain and then install it.

The following is the directory of the cross compilation tool chain:

```
Software/ServerInstall/arm-hisiv200-linux
```

Run the following command in the directory for installation package of the cross compilation tool chain as the root user:

```
$ ./cross.install
```

Log out of the server and log in to the server again.

Enter **arm-hisiv200-linux-** and press **Tab**. If the commands can be automatically completed, the installation is successful.

If the commands cannot be automatically completed, check whether the environment variables are correct by running the following command:

```
$ echo $PATH
```

If the environment variables are incorrect, edit the **/etc/profile** file as the root user.

```
$ sudo vi /etc/profile
```

Add the following statement:

```
export PATH="/home/root/bin/x86-arm/arm-hisiv200-linux/target/bin:$PATH"
```

Step 7 Set the maximum number of files that can be opened at the same time.

When the following errors occur during compilation, you need to set the maximum number of files that can be opened at the same time.

```
Exception in thread "main" java.lang.RuntimeException: error while
reading to file: java.io.FileNotFoundException, msg:cts/tools/vm-
tests/src/dot/junit/opcodes/aget_wide/Test_aget_wide.java (Too many open
files)
       at
util.build.BuildDalvikSuite.parseTestMethod(BuildDalvikSuite.java:743)
       at
util.build.BuildDalvikSuite.handleTests(BuildDalvikSuite.java:359)
       at util.build.BuildDalvikSuite.compose(BuildDalvikSuite.java:170)
       At util.build.BuildDalvikSuite.main(BuildDalvikSuite.java:136)
```

To set the maximum number of files that can be opened at the same time, perform the following steps:

```
$ sudo vi /etc/security/limits.conf
```

Add the following content in the last line:

```
*             -     nofile         8192
```

The following information is displayed in the last lines of **limits.conf**:

```
#<domain>     <type>  <item>        <value>
#


#*             soft    core          0
#root          hard    core          100000
#*             hard    rss           10000
#@student       hard    nproc         20
#@faculty       soft    nproc         20
#@faculty       hard    nproc         50
#ftp           hard    nproc         0
#ftp           -       chroot        /ftp
#@student       -       maxlogins     4
```

```
# End of file
*               -       nofile          8192
```

After logging in to the server again, run **ulimit –n** to check whether the setting is correct. If the setting is correct, **8192** is displayed.

Step 8 Check the umask value of the server.

Run the **umask** command in the compilation server shell and check whether **0022** is returned.

```
$ umask
```

If **0022** is not returned, recompile the **/etc/profile** file.

```
$ sudo vi /etc/profile
```

Add the following statement:

```
umask 022
```

Log in to the server again and verify that the umask value is correct.

Step 9 Create the fast compilation cache directory.

Create the **/run/shm** directory in the root directory of the server by running the following command:

```
$ sudo mkdir -p /run/shm/
```

Assign permission for the directory to all users by running the following command:

```
$ sudo chmod 777 -R /run/shm/
```

CAUTION

The fast compilation function is provided by Android officially. For details, visit http://source.android.com/source/initializing.html#setting-up-ccache.

Skip step 9 for Ubuntu 12.04 or later.

----End

2.3 Development Compilation

2.3.1 Obtaining the Source Code

The source code **HiSTBAndroidV600R001CXXSPCXXX.tar.gz** is stored in the **Software** directory of the SDK.

Copy the source code package to the server and untar the package by running **tar xzf HiSTBAndroidV600R001CXXSPCXXX.tar.gz**.

2.3.2 Configuring the Environment

Run the following command in the code root directory:

```
source build/envsetup.sh
lunch Hi3798MV100-eng
```

CAUTION

Running the preceding commands each time you log in to the server or switch the compilation directory.

Run all the commands in this section (except those in section 2.3.13) in the code root directory.

When the commands described in sections 2.3.3, 2.3.4, 2.3.8, and 2.3.13are executed, both the images to be burnt to the NAND flash and the images to be burnt to the eMMC are compiled. The output directory for the images to be burnt to the NAND flash is **out/target/product/Hi379XXVXXX/Nand**, and that for the images to be burnt to the eMMC is **out/target/product/Hi379XXVXXX/Emmc**.

The commands in this section are independent of each other and can be separately executed.

The parameter **Hi379XXVXXX-eng** in the **lunch** command is described as follows: **Hi379XXVXXX** indicates the product name based on which the structure of the **out** directory for the compilation result is generated. For example, the product code directory corresponding to **out/target/product/**produce name/ (for example, **out/target/product/Hi379XXVXXX/**) is **device/hisilicon/**product name/ (for example, **device/hisilicon/Hi379XXVXXX/eng**). The compilation mode is the eng mode by default.

To obtain the names of all products supported in the current version, run **lunch** after running **source buid/envsetup.sh**.

The parameter varies according to products. This section takes Hi379XX VXXX as an example. The environment configuration of other products is similar.

In the code compilation file **Android.mk**, the default value of **LOCAL_MODULE_TAGS** is **optional**. In this case, the files generated after compilation are not automatically copied to the compilation result directory. If you require that the files generated after compilation be automatically copied to the compilation result directory, and the generated files are dynamic libraries, .jar files, executable files, and precompiled files, add **ALL_DEFAULT_INSTALLED_MODULES += $(LOCAL_MODULE)** to **Android.mk**; if the generated files are .apk files, add **ALL_DEFAULT_INSTALLED_MODULES += $(LOCAL_PACKAGE_NAME)** to **Android.mk**.

2.3.3 Compiling All Images

Run the following command in the code root directory:

```
make bigfish -j32 2>&1 | tee bigfish.log
```

When this command is executed, the following images are compiled:

- In the out/target/product/Hi379XXVXXX/Nand directory:

  ```
  Android system kernel image:              kernel.img
  Android system partition image:       system_[x]_[y].ubi
  ```

```
Android userdata partition image:      userdata_[x]_[y].ubi
Android cache partition image:         cache_[x]_[y].ubi
Android sdcard partition image:        sdcard_[x]_[y].ubi
bootargs partition image:              bootargs_Hi379XXVXXX-nand.bin
Partition table:                           Hi3716CV200-nand.xml
Recovery small system kernel image:            recovery.img
Recovery upgrade package:              update.zip
Fastboot image:         fastboot.bin
Default precompiled baseparam partition image: baseparam.img
Default precompiled logo partition image:          logo.img
```

In the .ubi images, **[x]** indicates the page size, and **[y]** indicates the block size. For example, **system_4K_1M.ubi**. You can burn the images as required.

- In the out/target/product/Hi3798MV100/Emmc directory:

```
Android system kernel image:           kernel.img
Android system partition image:        system.ext4
Android userdata partition image:          userdata.ext4
Android cache partition image:         cache.ext4
Android sdcard partition image:        sdcard.ext4
Bootargs partition image:              bootargs.bin
Partition table:                       Hi379XXVXXX-emmc.xml
Recovery small system kernel image:        recovery.img
Recovery upgrade package:              update.zip
Fastboot image:                fastboot.bin
Default precompiled baseparam partition image: baseparam.img
Default precompiled logo partition image:      logo.img
```

- In the out/target/product/Hi379XXVXXX/ directory:

```
Burning tool:                      HiTool-[version](STB).zip
```

[version] indicates the release version of the HiTool in the SDK.

CAUTION

The parameter **-j32** indicates the number of threads for running commands. You can change the number after **-j** as required.

bigfish.log is stored in the code root directory. It stores the displayed information during compilation.

The preceding descriptions apply to other compilation commands.

2.3.4 Compiling the Android Kernel Image

Run the following command in the code root directory:

```
make kernel -j32 2>&1 | tee kernel.log
```

When this command is executed, the following image is compiled:

```
kernel.img
```

This image is copied to the **Nand** and **Emmc** directories of **out/target/product/Hi379XXVXXX/**. That is, the Android kernel partition image used in the NAND flash is the same as that used in the eMMC.

The process files generated during compilation are stored in the following directory:

```
out/target/product/Hi379XXVXXX/obj/KERNEL_OBJ/
```

2.3.5 Modifying Android Kernel Configuration

Run the following command in the code root directory:

```
make kernel_menuconfig
```

CAUTION

The default Android kernel configuration file is recorded in the **ANDROID_KERNEL_CONFIG** variable in **device/hisilicon/Hi379XXVXXX/BoardConfig.mk**.

The kernel configuration file modified by running the preceding command is stored in the temporary directory **out/target/product/Hi379XXVXXX/obj/KERNEL_OBJ/.config**. You need to store the modified file to **device/hisilicon/bigfish/sdk/source/kernel/linux-3.4.y/arch/arm/configs/**.

After modifying the kernel configuration, you can run the command in section 2.3.4 "Compiling the Android Kernel Image" to compile the new kernel image.

If you want to change the name of the kernel configuration file used during kernel compilation of the Android system, change the value of the **ANDROID_KERNEL_CONFIG** variable in **device/hisilicon/Hi379XXVXXX/BoardConfig.mk** and the value of the **CFG_HI_KERNEL_CFG** variable in the corresponding SDK configuration file in **device/hisilicon/bigfish/sdk/configs/**. The name of the corresponding SDK configuration file is defined in the **HISI_SDK_ANDROID_CFG** variable in **device/hisilicon/Hi379XXVXXX/BoardConfig.mk**. You need to store the new kernel configuration file to **device/hisilicon/bigfish/sdk/source/kernel/linux-3.4.y/arch/arm/configs/**.

The kernel configuration file must end with _defconfig.

You are advised to delete the kernel temporary file storage directory **out/target/product/Hi379XXVXXX/obj/KERNEL_OBJ/** before running the preceding command.

2.3.6 Compiling the Android System Image for the NAND Flash

Run the following command in the code root directory:

```
make ubifs -j32 2>&1 | tee ubifs.log
```

When this command is executed, the following images are compiled in the **out/target/product/Hi379XXVXXX/Nand** directory:

```
system_[x]_[y].ubi
userdata_[x]_[y].ubi
cache_[x]_[y].ubi
sdcard_[x]_[y].ubi
kernel.img
```

In the .ubi images, **[x]** indicates the page size, and **[y]** indicates the block size. For example:

```
system_4K_1M.ubi
data_4K_1M.ubi
cache_4K_1M.ubi
sdcard_4K_1M.ubi
```

In the preceding examples, **4K** indicates the page size, and **1M** indicates the block size.

2.3.7 Compiling the Android System Image for the eMMC

Run the following command in the code root directory:

```
make ext4fs -j32 2>&1 | tee ext4fs.log
```

When this command is executed, the following images are compiled in the **out/target/product/Hi379XXVXXX/Emmc** directory:

```
system.ext4
userdata.ext4
cache.ext4
sdcard.ext4
kernel.img
```

2.3.8 Compiling the Kernel Image for the Android Recovery Small System

Run the following command in the code root directory:

```
make recoveryimg -j32 2>&1 | tee recovery.log
```

When this command is executed, the following image is compiled:

```
recovery.img
```

This image is copied to the **Nand** and **Emmc** directories of **out/target/product/Hi379XXVXXX/**. That is, the kernel partition image for the Android recovery small system used in the NAND flash is the same as that used in the eMMC.

The process files generated during compilation are stored in the following directory:

```
out/target/product/Hi379XXVXXX/obj/RECOVERY_OBJ/
```

2.3.9 Modifying Kernel Configuration for the Android Recovery Small System

Run the following command in the code root directory:

```
make recovery_menuconfig
```

CAUTION

The default kernel configuration file of the Android recovery small system is recorded in the **RECOVERY_KERNEL_CONFIG** variable in **device/hisilicon/Hi379XXVXXX/BoardConfig.mk**.

The kernel configuration file modified by running the preceding command is stored in the temporary directory **out/target/product/Hi379XXVXXX/obj/RECOVERY_OBJ/.config**. You need to store the modified file to **device/hisilicon/bigfish/sdk/source/kernel/linux-3.10.y/arch/arm/configs/**.

After modifying the kernel configuration, you can run the command in section 2.3.7 "Compiling the Android System Image for the eMMC" to compile the new kernel image for the Android recovery small system.

If you want to change the default name of the kernel configuration file for the Android recovery small system, change the value of the **RECOVERY_KERNEL_CONFIG** variable in **device/hisilicon/Hi379XXVXXX/BoardConfig.mk**. You need to store the new kernel configuration file to **device/hisilicon/bigfish/sdk/source/kernel/linux-3.10.y/arch/arm/configs/**.

The kernel configuration file must end with _defconfig.

You are advised to delete the kernel temporary file storage directory **out/target/product/Hi379XXVXXX/obj/RECOVERY_OBJ/** before running the preceding command.

2.3.10 Compiling the Android Recovery Upgrade Package

Run the following command in the code root directory:

```
make updatezip -j32 2>&1 | tee updatezip.log
```

When this command is executed, the following image is compiled in the **out/target/product/Hi379XXVXXX/Nand** directory:

```
update.zip
```

By using the eMMC, **update.zip** is generated in **out/target/product/Hi379XXVXXXEmmc**.

If the recovery upgrade package is not required, you do not need to run this command.

2.3.11 Deleting Compilation Results

Run the following command in the code root directory:

```
make clean
```

CAUTION

All the process files generated during compilation are stored in the **out** directory. Running this command deletes all these files.

If you want to separately delete the temporary files for the Android system kernel, Android recovery small system kernel, or fastboot, you can directly delete the corresponding directory.

2.3.12 Modifying the HiSilicon SDK Configuration File

Take Hi3798M V100 as an example. The SDK configuration file is stored in the following directory:

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

Go to the SDK directory by running the following command:

```
cd device/hisilicon/bigfish/sdk
```

Run the following command to modify the corresponding configuration file:

```
make menuconfig SDK_CFGFILE=configs/hi3798mv100 name of the configuration
file to be modified
```

For example:

```
make menuconfig
SDK_CFGFILE=configs/hi3798mv100/hi3798mdmo_hi3798mv100_android_cfg.mak
```

The modifications are saved to the corresponding configuration file in **device/hisilicon/bigfish/sdk/configs/**.

CAUTION

The name of the SDK configuration file is defined by the **HISI_SDK_ANDROID_CFG** variable in **BoardConfig.mk**, for example: **device/hisilicon/Hi3798MV100/BoardConfig.mk**; **HISI_SDK_ANDROID_CFG := hi3798mdmo_hi3798mv100_android_cfg.mak**.

Running this command allows you to modify the SDK configuration file in the displayed GUI. After you save the modification and exit the GUI, the modified configuration file is stored in the directory for the SDK configuration file, and the original configuration file is renamed configuration file name**.old** and stored in the same directory.

You can also manually modify the corresponding configuration file in the directory for storing SDK configuration files.

Verify the name of the SDK configuration file before modification.

2.3.13 Compiling the Fastboot Partition Image

Take Hi3798M V100 as an example. Run the following command in the code root directory:

```
make hiboot -j32 2>&1 | tee hiboot.log
```

When this command is executed, the following images are compiled:

- In the out/target/product/Hi3798MV100/Nand directory:
  fastboot-burn-nand.bin
  The process files generated during compilation are stored in the following directory:
  out/target/product/Hi3798MV100/obj/NAND_HIBOOT_OBJ/
- In the out/target/product/Hi3798MV100/Emmc directory:
  fastboot-burn-emmc.bin
  The process files generated during compilation are stored in the following directory:
  out/target/product/Hi3798MV100/obj/EMMC_HIBOOT_OBJ/

CAUTION

During complete compilation, the fastboot is compiled by calling this command.

The compilation of the fastboot is based on the fastboot configuration defined in the SDK configuration file **device/hisilicon/Hi3798MV100/BoardConfig.mk**.

```
# emmc fastboot configure
EMMC_BOOT_CFG_NAME :=
hi3798mdmo1a_hi3798mv100_ddr3_1gbyte_16bitx2_4layers_emmc.cfg
EMMC_BOOT_REG_NAME :=
hi3798mdmo1a_hi3798mv100_ddr3_1gbyte_16bitx2_4layers_emmc.reg
EMMC_BOOT_ENV_STARTADDR :=0x100000
EMMC_BOOT_ENV_SIZE=0x10000
EMMC_BOOT_ENV_RANGE=0x10000

# nand fastboot configure
NAND_BOOT_CFG_NAME :=
hi3798mdmo1a_hi3798mv100_ddr3_1gbyte_16bitx2_4layers_nand.cfg
NAND_BOOT_REG_NAME :=
hi3798mdmo1a_hi3798mv100_ddr3_1gbyte_16bitx2_4layers_nand.reg
NAND_BOOT_ENV_STARTADDR :=0x800000
NAND_BOOT_ENV_SIZE=0x10000
NAND_BOOT_ENV_RANGE=0x10000
```

If you want to modify the fastboot configurations, such as the .reg files and directory for storing bootargs, you need to modify the corresponding configuration file in **BoardConfig.mk** and then run this command.

This command can be directly executed, not necessarily after all images are compiled.

The fastboot image is related to the partition table and the bootargs partition image. If one of the three images is updated, the other two images also need to be updated.

You do not need to run this command if you do not want to update the fastboot image.

2.4 Burning Images

2.4.1 Flash Partition Table

Take the NAND flash as an example. You can allocate the NAND flash partitions based on the flash configuration of the board by complying with the following rules (for details about the eMMC partitions, see the default eMMC partition table):

- The size of each partition of the SPI flash must be an integral multiple of the page size.
- The size of each partition of the NAND flash must be an integral multiple of the block size.
- There must be certain spare space in each partition of the NAND flash, and at least two blocks are reserved for redundancy.

Table 2-2 describes the examples of flash partitions.

Table 2-2 Flash partition table example

No.	Partition Name	Function	Required Capacity (Byte)	Allocated Capacity (Byte)	Remarks	Clipping Supported
1	fastboot	Bootloader for booting the system The system enters the recovery mode or boot mode based on the key or content of the MISC partition.	512 KB	512 KB	SPI flash	No
2	bootargs	Partition information, memory information, and kernel boot information	64 KB	64 KB	SPI flash	No
3	recovery	Recovery mode It includes the kernel, recovery rootfs, and recovery app.	3456 KB	3456 KB	SPI flash	No
4	deviceinfo	Device information including the MAC address, device ID, and product ID	64 KB	64 KB	SPI flash	No
5	baseparam	Startup picture display and fastplay parameters generated	1 KB	8 MB	NAND flash	No

No.	Partition Name	Function	Required Capacity (Byte)	Allocated Capacity (Byte)	Remarks	Clipping Supported
		by the HiTool. The parameters include the startup picture aspect ratio, DAC coefficient, and standard.				
6	pqparam	Image quality parameter	200 KB	8 MB	NAND flash	No
7	logo	Startup picture generated by the HiTool	1 MB It varies according to the actual requirement.	20 MB	NAND flash	No
8	logobak	Startup picture backup, generated by the HiTool. This partition is used when the logo partition is invalid.	1 MB It varies according to the actual requirement.	20 MB	NAND flash	No
9	fastplay	Startup animation generated by the HiTool	10 MB It varies according to the actual requirement.	40 MB	NAND flash	No
10	fastplayback	Startup animation backup, generated by the HiTool. This partition is used when the fastplay partition is invalid.	10 MB It varies according to the actual requirement.	40 MB	NAND flash	No
11	kernel	Linux kernel+rootfs	10 MB	40 MB	NAND flash	No
12	misc	Bootloader control block for storing the recovery bootloader message	512 KB	20 MB	NAND flash	No
13	system	Android system partition	200 MB It varies according to the actual requirement.	500 MB	NAND flash	No
14	userdata	User data	100 MB It varies according to the actual requirement.	1024 MB	NAND flash	No
15	cache	Upgrade packages (fastboot and kernel)	50 MB	100 MB	NAND	No

No.	Partition Name	Function	Required Capacity (Byte)	Allocated Capacity (Byte)	Remarks	Clipping Supported
		and the patch packages (system) This partition is used to exchange the commands, logs, and intent information during recovery.	If the system uses the upgrade package, this partition must be extended.		flash	The size can be set based on the upgrade requirements.
16	sdcard	Embedded SD card partition	2350 MB	2284 MB	NAND flash	No

CAUTION

This partition table is a demo, which may be inconsistent with the released code. The actual partition table prevails.

2.4.2 Burning Images to the Flash Memory

NOTE

This section uses Hi3798M V100 as an example.

2.4.2.1 Preparations

This section takes the NAND flash as an example to describe how to burn the images.

Prepare the compiled images. The directory for storing the compilation results is as follows:

```
out/target/product/Hi3798MV100/Nand
```

- fastboot image: **fastboot.bin**
- bootargs image: **bootargs.bin**
- baseparam image: **baseparam.img**
- logo image: **logo.img**
- kernel image: **kernel.img**
- recovery image: **recovery.img**
- recovery update package: **update.zip**
- System image: system_[x]_[x].ubi
- Userdata image: userdata_[x]_[x].ubi
- Cache image: cache_[x]_[x].ubi
- Sdcard image: sdcard_[x]_[x].ubi
- Burning tool: out/target/product/Hi3798MV100/HiTool-[version](STB).zip

2.4.2.2 Burning Method

To burn images to the flash, perform the following steps:

Step 1 Connect the network port cable, serial cable, and power cable to the board.

Step 2 Start the burning tool, configure the PC and the board, and set the transmission mode to **By Net (Recommend)**.

Step 3 Select **Burn by Partition**, and select the partition table file, for example, the demo partition table **Hi3798MV100-nand.xml**.

Step 4 Select the image to be burnt.

Step 5 Click **Burn** and power on the board.

"Burn success!" is displayed if the image is successfully burnt.

----End

3 Memory Configuration

NOTE

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

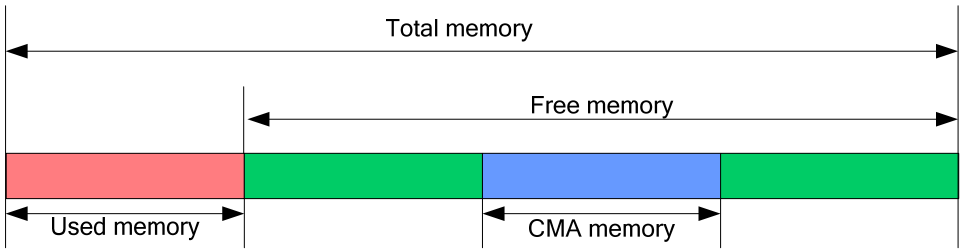
3.1 Memory Allocation

The Android solution allocates a large continuous physical memory for drivers to use in continuous memory allocator (CMA) mode. A continuous physical memory is allocated from the available memory based on the configured CMA memory size when the system boots.

The size of the CMA memory can be configured in the bootargs environment variable or the kernel configuration file, and the configuration of the bootargs environment variable has higher priority.

Figure 3-1 shows the CMA memory in the total memory.

Figure 3-1 CMA memory



An advantage of the CMA memory is that the kernel can use the CMA memory when there is CMA memory available. Compared with the memory allocation mode in which a memory is reserved for the kernel, memory can be used more efficiently in CMA mode. The CMA memory is physically continuous and mainly used by modules such as the encoding, decoding, transcoding, and graphics display modules.

3.2 Modifying CMA Memory Configuration

The same image can be burnt to boards with the same hardware but different DDR sizes (which can be 512 MB, 1 GB, and 2 GB respectively). The bootargs parameters for different DDR sizes are configured by using different environment variables in the bootargs partition. For example:

```
512MB DDR: bootargs_512M=mem=512M mmz=ddr,0,0,235M
1GB DDR: bootargs_1G=mem=1G mmz=ddr,0,0,400M
2GB DDR: bootargs_2G=mem=2G mmz=ddr,0,0,600M
```

The CMA memory size can be changed by configuring the corresponding DDR size environment variable or the bootargs environment variable. However, if the CMA memory size is configured in the bootargs environment variables, the configuration of the corresponding DDR size environment variable does not take effect.

NOTE

Use this method only during debugging because you are advised not to update the bootargs partition frequently.

To change the size of the CMA memory by setting the bootargs environment variable, perform the following steps:

Step 1 Press **Ctrl+C** to restart the board to enter the fastboot command line.

Step 2 Read the values of the bootargs environment variables by running the **printenv** command.

```
printenv
```

Step 3 Modify the DDR and CMA configurations in the environment variables.

- Method 1: Change the default CMA configuration in the DDR size environment variable. To be specific, change *XXX* in **mmz=ddr,0,0,*XXX*M** to the required value.
- Method 2: Add **mem=*YYY* mmz=ddr,0,0,*XXX*M** to the end of the bootargs environment variables. *YYY* is the total DDR size, and *XXX* is the CMA size. For example, if you want to set the DDR size to 512 MB and CMA size to 220 MB, add **mem=512M mmz=ddr,0,0,220M** to the end of the bootargs variables.

Before modification:

```
console=ttyAMA0,115200
blkdevparts=mmcblk0:1M(fastboot),1M(bootargs),10M(recovery),2M(deviceinfo
),8M(baseparam),8M(pqparam),20M(logo),20M(logobak),40M(fastplay),40M(fast
playbak),40M(kernel),20M(misc),8M(userapi),8M(hibdrv),8M(qbflag),300M(qbd
ata),800M(system),1024M(userdata),100M(cache),-(sdcard)
```

After modification:

```
console=ttyAMA0,115200
blkdevparts=mmcblk0:1M(fastboot),1M(bootargs),10M(recovery),2M(deviceinfo
),8M(baseparam),8M(pqparam),20M(logo),20M(logobak),40M(fastplay),40M(fast
playbak),40M(kernel),20M(misc),8M(userapi),8M(hibdrv),8M(qbflag),300M(qbd
ata),800M(system),1024M(userdata),100M(cache),-(sdcard) mem=512M
mmz=ddr,0,0,220M
```

CAUTION

The added data must be separated from the preceding character strings by using a space. If you use method 2, the environment variables corresponding to the three default DDR sizes do not take effect, and the DDR size and CMA size configured in the bootargs environment variables take priority. If you want to use the environment variables corresponding to the three default DDR sizes, restore the bootargs environment variables to default values.

Step 4 Save the environment variables and restart the system by running the following commands over the serial port.

```
saveenv
reset
```

----End

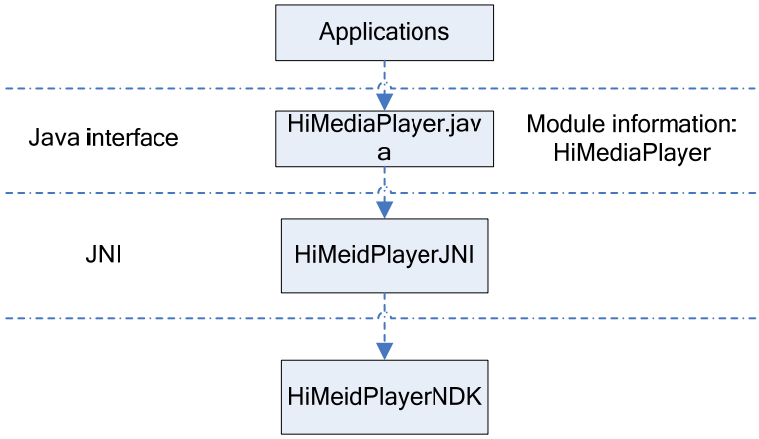
4 HiMediaPlayer

4.1 Overview

The HiMediaPlayer provides a framework for multimedia playback. By using the APIs of this framework, upper-layer applications can play media files in various formats in Android. In addition, the HiSilicon media playback extended APIs and bottom-layer HiSilicon media playback modules implement higher playback performance and more functions in subtitles and audio tracks.

Figure 4-1 shows the HiMediaPlayer framework.

Figure 4-1 HiMediaPlayer framework



- Java interface

  The Java interface layer consists of the following two parts:

  - HiMediaPlayer.java

    Provides media playback interfaces for the application layer.

  - HiBDInfo.java

    Implements the playback navigation for blu-ray disc (BD) files or ISO, and obtains blu-ray media information such as the title.

- JNI

The JNI layer provides media playback interfaces and blu-ray information for upper-layer applications and calls the native development kit (NDK) and libbluray to implement media playback.

4.2 Important Concepts

[HiPlayer]

The HiPlayer is a HiSilicon media playback module.

[Surface]

A surface is a display region for video playback.

[ISO]

The ISO is an image file format. Blu-ray streams are typically stored as ISO files.

4.3 Function Description

4.3.1 Features

The HiMediaPlayer has the following features relative to the Android inherent architecture:

- More supported audio and video formats

  The HiSilicon media playback module HiPlayer supports more audio and video formats than the Android inherent architecture.

  The Android inherent architecture supports the following formats: MP4, M4A, MKV, MP3, WAV, FLAC, OGG, AAC, MPEG-TS, 3GP, MIDI-MID, MIDI-XMF, MIDI-MXMF, MIDI-RTTTL, MIDI-RTX, MIDI-OTA, MIDI-IMY, WEBM, and M3U8.

  Extended formats: ASF, M1V, M2V, MPG/DATA, VOB, TS, MTS/M2TS, AVI, MKV, RM, RMVB, WMV, MOV, M4V, F4V, 3GP, 3G2, TP, TRP, M2P, CUE, APE, M3U8, M3U9, PLS, ISO, AND BDMV DIR
- More API functions

  Functions such as fast forwarding, rewinding, and switching the subtitle or audio track are supported, which are more suitable for the TV and STB players.
- Higher performance

  The HiPlayer plays HD streams and blu-ray streams more smoothly.

4.3.2 APIs

Basic playback functions:

- HiMediaPlayer: Initializes the HiMediaPlayer.
- setOnPreparedListener: Sets the callback interface for the prepare function.
- setOnCompletionListener: Sets the callback interface for playback completion.
- setOnBufferingUpdateListener: Sets the callback interface for buffer status listening.
- setOnSeekCompleteListener: Sets the callback interface for seek completion.
- setOnVideoSizeChangedListener: Sets the callback interface for window status changing.

- setDataSource: Sets the data source for playback.
- setDisplay: Sets the application layer surface for playback.
- setVideoRange: Sets the video display region for playback.
- prepare: Prepares for playback (sync).
- prepareAsync: Prepares for playback (async).
- start: Starts the playback.
- pause: Pauses the playback.
- stop: Stops the playback.
- reset: Resets the player.
- release: Releases the HiMediaPlayer.

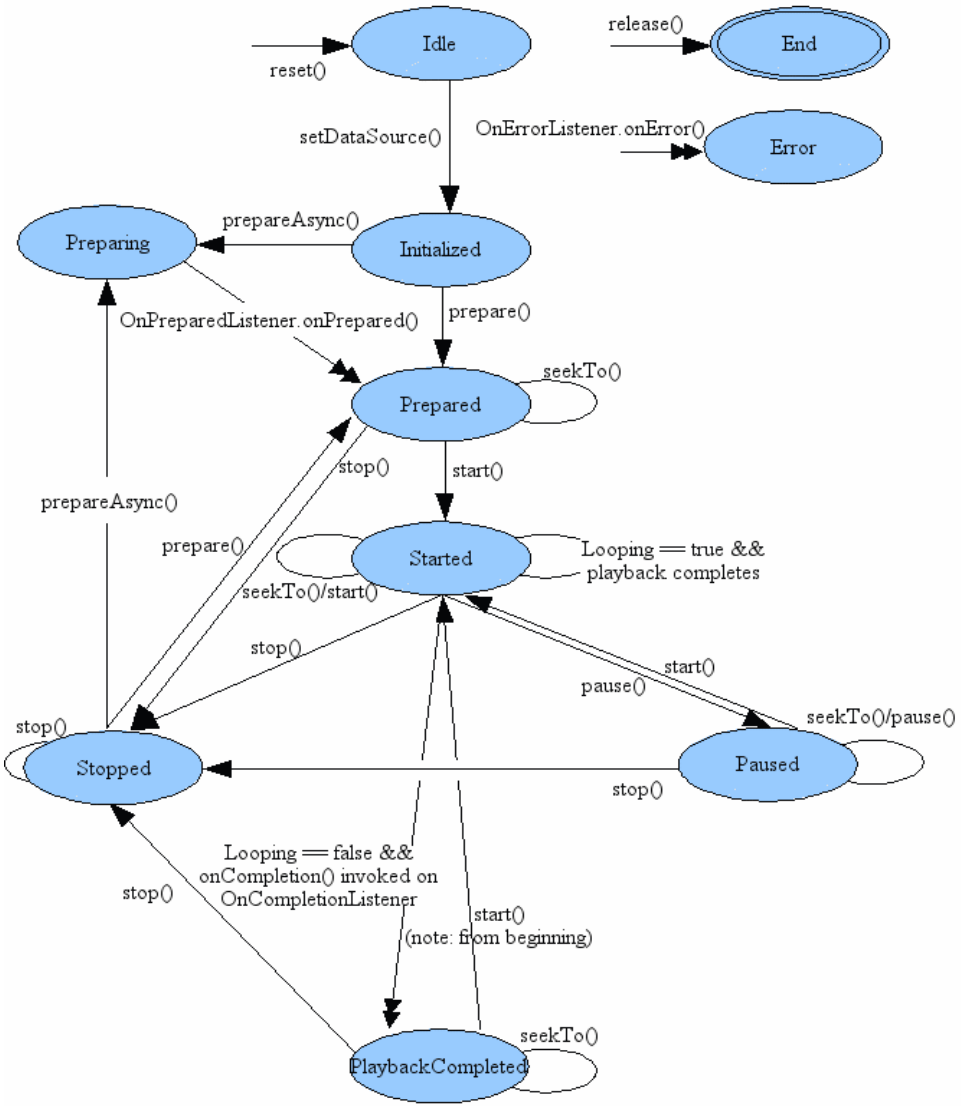
HiSilicon extended functions:

- setAudioTrack: Sets the ID of the audio streams to be played.
- setAudioChannel: Sets the audio track mode.
- setSubFontColor: Sets the subtitle color.
- setSubFontStyle: Sets the subtitle font.
- setSubFontSize: Sets the subtitle font size.
- setSubFontSpace: Sets the space between subtitle characters.
- setSubFontLineSpace: Sets the space between subtitle lines.
- setSubEncode: Sets the subtitle encoding format.
- setSubTrack: Sets the subtitle stream ID.
- setSubTimeOffset: Sets the subtitle sync time offset.
- setSubVertical: Sets the vertical position of the subtitle.
- setSubPath: Exports external subtitles.
- setBufferSizeConfig: Sets the threshold for the buffer data size.
- getBufferSizeConfig: Obtains the threshold for the buffer data size.
- setBufferTimeConfig: Sets the time threshold for the buffer.
- getBufferTimeConfig: Obtains the time threshold for the buffer.
- setFreezeMode: Sets the freeze mode of the current player.
- getFreezeMode: Obtains the freeze mode of the current player.
- setStereoVideoFmt: Sets the video input format.
- setStereoStrategy: Sets the video output format.

4.3.3 Function Implementation

Figure 4-2 shows the function implementation diagram of the HiMediaPlayer.

Figure 4-2 Function implementation diagram of the HiMediaPlayer



The working process of the HiMediaPlayer is as follows:

Step 1 Make preparations.

1. Create a HiMediaPlayer instance.
2. Configure various callback functions.
3. Set the directory for the file to be played.
4. Set the surface for subtitle and video display.
5. Set the coordinates (X, Y) for the upper left corner of the video display region and the video aspect ratio (W, H).
6. Identify the file and initialize the decoder by calling the prepare or prepareAsync function.
7. Listen to the OnPrepare callback function. The player status is changed from idle to prepared.

Step 2 Control the playback.

Control the playback by using commands such as start, stop, pause, setSpeed, and getXXXInfo.

Step 3 Perform the seek operation.

- The seek operation is an async operation because it requires various operations at the bottom layer, such as redirecting the file position, adjusting the data buffer, and clearing the decoder.
- After the seek function is called, the OnSeekComplete() callback function needs to be listened to complete the seek operation.
- The system is in the started status before the seek operation is complete, and operations such as resetting, releasing, and seeking again are not allowed.

Step 4 Process events.

When the player is running, various information events (such as buffer status changing) and error events (such as unsupported file) are generated. The player needs to listen to the error events, information events, and playback completion events as required.

Step 5 End the playback.

- The application can close the files opened by the HiPlayer and release various resources occupied by the HiPlayer any time by calling the reset function, or end the playback by calling the release function (the reset function is automatically called internally).
- After reset, all playback information is cleared. If you want to play the file again, start the entire playback process from the beginning.

----End

4.4 Development Guide

The following are the typical HiMediaPlayer development scenarios:

- HiMediaPlayer JNI playback
- Open-source customization
- SmoothStraming

4.4.1 HiMediaPlayer JNI Playback

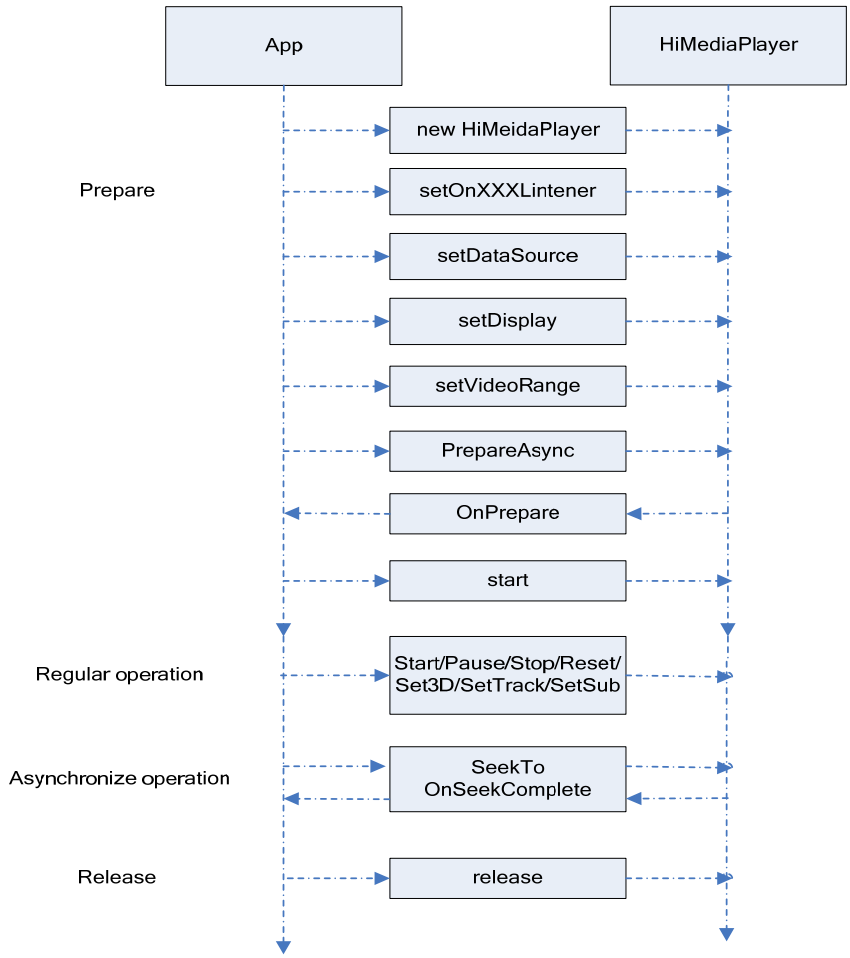
Scenario

In the source code of the Android solution in the HiSTBAndroid VX00R001 project, the HiMediaPlayer interfaces are used to develop Android video playback applications.

Working Process

Figure 4-3 shows the working process of the HiMediaPlayer.

Figure 4-3 Working process of the HiMediaPlayer



The process is as follows:

Step 1 Add the **HiMediaPlayer .jar** package to **Android.mk** in static link mode.

```
LOCAL_STATIC_JAVA_LIBRARIES += HiMediaPlayer
```

Step 2 Import the HiMediaPlayer packages in the code.

```
import com.hisilicon.android.mediaplayer.HiMediaPlayer;
import
com.hisilicon.android.mediaplayer.HiMediaPlayer.OnCompletionListener;
import com.hisilicon.android.mediaplayer.HiMediaPlayer.OnErrorListener;
import
com.hisilicon.android.mediaplayer.HiMediaPlayer.OnFastBackwordCompleteLis
tener;
import com.hisilicon.android.mediaplayer.HiMediaPlayer.OnInfoListener;
import  com.hisilicon.android.mediaplayer.HiMediaPlayer.OnPreparedListener;
```

Step 3 Initialize the HiMediaPlayer object.

```
HiMediaPlayer  mMediaPlayer = new HiMediaPlayer();
```

Step 4 Set listening (callback).

- Set the prepare or prepareAsync callback interface.
  ```
  mMediaPlayer.setOnPreparedListener(mPreparedListener);
  ```
- Set the callback interface for video size changing.
  ```
  mMediaPlayer.setOnVideoSizeChangedListener(mSizeChangedListener);
  ```
- Set the callback interface for playback completion.
  ```
  mMediaPlayer.setOnCompletionListener(mCompletionListener);
  ```
- Set the callback interface for fast forwarding or rewinding completion.
  ```
  mMediaPlayer.setOnFastBackwordCompleteListener(mFastBackwordCompleteL
  istener);
  ```
- Set the error event callback interface.
  ```
  mMediaPlayer.setOnErrorListener(mErrorListener);
  ```
- Set the buffer size changing callback interface.
  ```
  mMediaPlayer.setOnBufferingUpdateListener(mBufferingUpdateListener);
  ```

Step 5 Set the directory for the file to be played.

```
mMediaPlayer.setDataSource(mContext, mUri);
```

Step 6 Set the surface for playback.

```
mMediaPlayer.setDisplay(mSurfaceHolder);
```

Step 7 Set the video playback region.

```
mMediaPlayer.setVideoRange(mX, mY, mW, mH);
```

Step 8 Prepares for playback.

```
mMediaPlayer.prepareAsync();
```

Step 9 Call onPrepared.

```
mMediaPlayer.start();
```

----End

Notes

- For details about the APIs of the HiMediaPlayer, see the *HiMediaPlayer API Development Reference*.
- The HiSilicon extended interfaces, except setStereoVideoFmt and setStereoStrategy, can be called only when the player is in the prepared status.

Sample

```
mMediaPlayer = new HiMediaPlayer();
mMediaPlayer.setOnPreparedListener(mPreparedListener);
mMediaPlayer.setOnVideoSizeChangedListener(mSizeChangedListener);
mMediaPlayer.setOnCompletionListener(mCompletionListener);
```

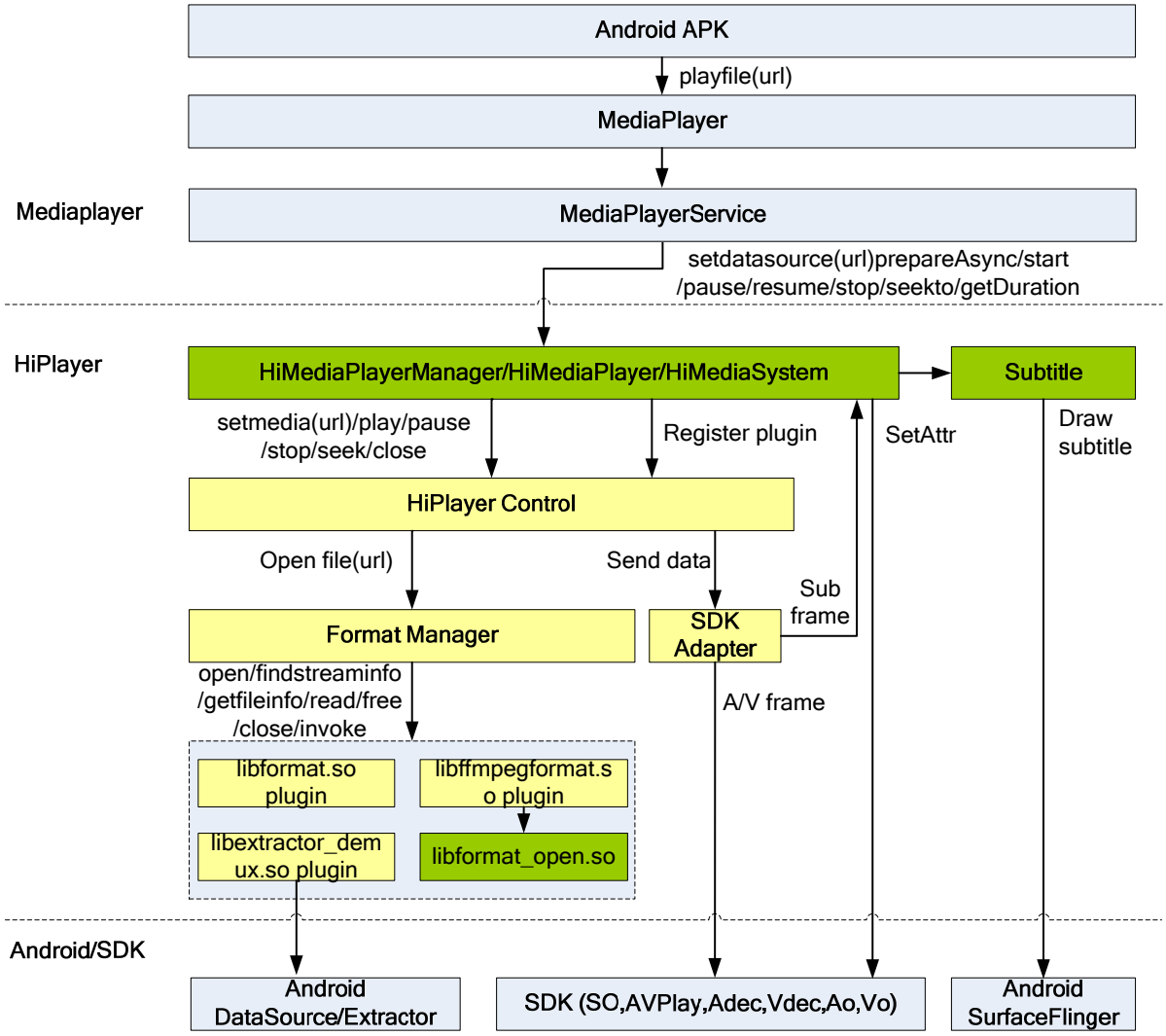
```
mMediaPlayer.setOnFastBackwordCompleteListener(mFastBackwordCompleteListe
ner);
mMediaPlayer.setOnInfoListener(m3DModeReceivedListener);
mMediaPlayer.setOnErrorListener(mErrorListener);
mMediaPlayer.setOnBufferingUpdateListener(mBufferingUpdateListener);
mCurrentBufferPercentage = 0;
mMediaPlayer.setDataSource(mContext, mUri);
mSurfaceHolder.setFixedSize(getVideoWidth(), getVideoHeight());
mMediaPlayer.setDisplay(mSurfaceHolder);
Surface mSurface;
mSurface = mSurfaceHolder.getSurface();

int[] location = new int[2];
getLocationOnScreen(location);

int mX, mY, mW, mH;
if (null != mSurface)
{
mX = location[0];
mY = location[1];
mW = mVideoWidth;
mH = mVideoHeight;
mMediaPlayer.setVideoRange(mX, mY, mW, mH);
}

mMediaPlayer.prepareAsync();
```

4.4.2 Open-Source Customization

4.4.2.1 Function Implementation Diagram

Except providing a set of NDK JNI interfaces, the HiMediaPlayer also connects to the Android standard MediaPlayer, and it is at the same level with StageFright and NuPlayer. The MediaPlayer selects the player based on the input file in routing mode. Figure 4-4 shows the diagram of connecting the HiMediaPlayer to the MediaPlayer.

Figure 4-4 HiMediaPlayer playback process



NOTE

In the preceding figure, the green parts of the HiPlayer are provided by the source code, and the yellow parts are provided by the binary libraries.

The diagram is divided into three layers:

- MediaPlayer layer
- HiPlayer layer
- Android/SDK layer

The HiPlayer layer provides the following components and functions:

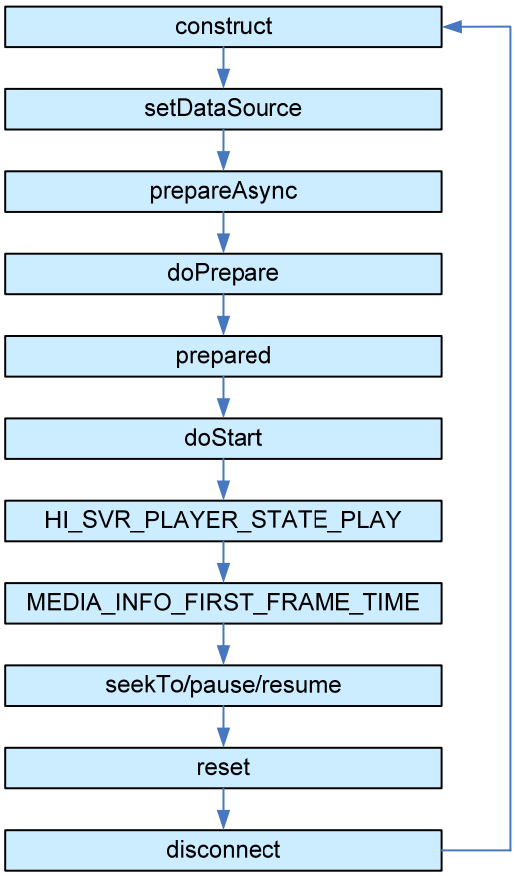
- HiMediaPlayer: adaptation layer from the HiPlayer APIs to MediaPlayer APIs
- HiMediaSystem: device initialization module, for initializing devices such as the AVPLAY and HiPlayer
- Subtitle: subtitle output module, for drawing the subtitles parsed by the HiPlayer or UTF-8 coded teletexts to the Android surface

- HiPlayer Control: player control module, for receiving playback control commands issued by the HiMediaPlayer, performing playback control operations, and reporting playback status information to the HiMediaPlayer as events
- Format Manager: player parser plug-in management module, for managing all parser plug-ins registered for the HiPlayer in the HiMediaSystem. When the APK specifies a media file for playback, the format manager traverses the parsers one by one until it finds the appropriate parser. The parser is then used for parsing the media file during playback.
- SDK Adapter: SDK adaptation layer, for connecting to the AVPLAY, SO, audio track, and window modules of the SDK
- libffmpegformat.so plugin: media file parser, for parsing the MP4, MKV, FLV, TS, and AVI media files
- libformat.so plugin: subtitle and protocol component, including the parser for the SRT, SMI, SSA/ASS, and IDX+SUB subtitles and the FILE protocol
- libextractor_demux.so plugin: Smooth Streaming Demux parser, for implementing the Smooth Streaming+PlayReady playback functions by calling the Android extractor (Android media file parser) and Datasource
- libformat_open.so: HTTP Live Streaming (HLS), HTTP, and TCP protocols, for providing protocols for the HLS, HTTP, and TCP stream media. It is the open-source part of the current HiSilicon player stream media protocols.

4.4.2.2 Playback Process

Figure 4-5 shows the key procedures of calling the MediaPlayer to play a media file by the Android APK.

Figure 4-5 Key procedures of media playback



The HiPlayer-related operations in the preceding key procedures are described as follows:

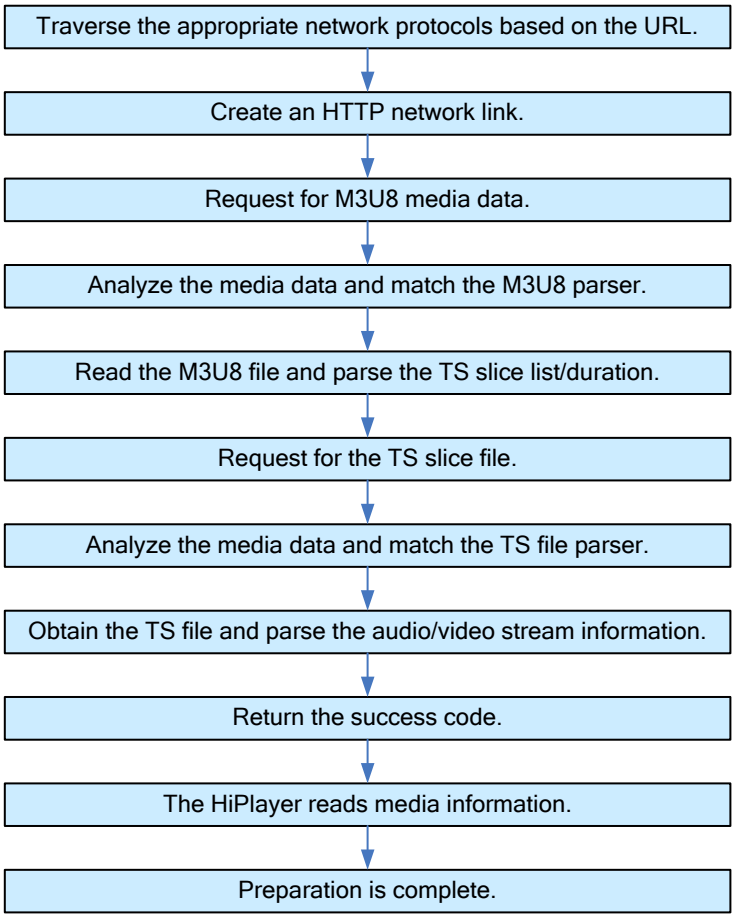
- prepareAsync

  Creates a HiPlayer. If the HiPlayer fails to be created, the playback fails. If the HiPlayer is created successfully, the user-configured network playback parameters are transferred to the HiPlayer, and an event is transmitted to trigger the doPrepare operation. prepareAsync is an asynchronous operation, and the called HiPlayer API is HI_SVR_PLAYER_Create.

- doPrepare

  Prepares for the playback, including opening the media file and parsing stream information in the media file, such as the number of audio/video streams and type of the audio/video CODEC. This operation is the most time-consuming one. The called HiPlayer API is HI_SVR_PLAYER_SetMedia, and the parser APIs called by the HiPlayer are fmt_find, fmt_open, fmt_find_stream, and fmt_getinfo (for details about the APIs, see **hi_svr_format.h**). Figure 4-6 shows the SetMedia process for the network stream media such as the HLS streams.

Figure 4-6 SetMedia process



NOTE

- For HLS live streams, there is no duration information. The M3U8 file is updated regularly during playback, and the seek operation is not supported. For VOD streams, the playback duration of the media file equals the total duration of segments. The seek operation is supported during playback, and the playback starts from the segment nearest to the sought position.
- For TS live streams, there is no duration information, and the seek operation is not supported. For VOD streams, the file duration is obtained by subtracting the time stamp of the start of file (SOF) from that of the end of file (EOF). After the seek operation is performed, the playback starts from the offset position calculated based on the sought time point, file duration, and file size.

- doStart

  Starts the playback. The procedures are as follows:

1. The HiPlayer initializes the audio/video decoder based on the media information obtained from the parser.

   If an error occurs, the player reports the HI_SVR_PLAYER_ERROR_PLAY_FAIL, HI_SVR_PLAYER_ERROR_AUD_PLAY_FAIL, or HI_SVR_PLAYER_ERROR_VID_PLAY_FAIL event based on the result. After an audio/video channel is enabled successfully, the HiPlayer state is changed from STOP to PLAY. (The playback can start if one audio/video channel is enabled successfully.)

2. The HiPlayer reads audio/video frame data by calling the parser APIs fmt_read and fmt_free (for details about the API definitions, see **hi_svr_format.h**) and transmits the frame data to the AVPLAY for decoding and playback.
3. After the first frame is output, the HiPlayer reports the HI_SVR_PLAYER_EVENT_FIRST_FRAME_TIME event, notifying the HiMediaPlayer of the output of the first frame.

   Therefore, the time duration for starting the playback is calculated from the time when setDataSource is called to the time when HI_SVR_PLAYER_EVENT_FIRST_FRAME_TIME is reported.

   Media files on the network cannot be played smoothly if the network condition is poor. Therefore, to avoid network buffering even before the playback starts, the HiPlayer plays for a short while and then determines the current network condition.

   For the HLS live play, if the network is abnormal (for example, the network is disconnected or data request times out), and the client does not obtain the segment file in time, the player will obtain the last segment saved on the current server after the network is restored. In this case, the frames may be skipped. That is, when the network is abnormal, the segment information is lost. This issue does not occur in VOD scenarios.

   Figure 4-7 shows an M3U8 list sample in the live play scenario. The left part is the segment information of time point 1, and the right part is that of time point 2. Time point 2 minus time point 1 is about 100s. Within this 100s, the network is disconnected and not restored. The player starts to play from **20121026T025514-174321.ts** after time point 2 is reached.

Figure 4-7 M3U8 list sample of HLS live play

#EXTM3U	#EXTM3U
#EXT-X-VERSION:3	#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:11	#EXT-X-TARGETDURATION:11
#EXT-X-MEDIA-SEQUENCE:174310	#EXT-X-MEDIA-SEQUENCE:174320
#EXTINF:10,	#EXTINF:10,
20121026T025514-174311.ts	20121026T025514-174321.ts
#EXTINF:10,	#EXTINF:10,
20121026T025514-174312.ts	20121026T025514-174322.ts
#EXTINF:10,	#EXTINF:10,
20121026T025514-174313.ts	20121026T025514-174323.ts
#EXTINF:10,	#EXTINF:10,
20121026T025514-174314.ts	20121026T025514-174324.ts
#EXTINF:10,	#EXTINF:10,
20121026T025514-174315.ts	20121026T025514-174325.ts

NOTE

During playback, the HiPlayer reads data from the parser by frame. The data must contain information such as the frame length (length), time stamp (pts), stream index (streamidx), and key frame flag (keyflag). When parsing raw streams, the parser needs to perform framing. For example, for the H.264 video with the MP4 encapsulation, the encapsulation unit of the file container may be NAL. The parser must group NALs into frames when parsing video streams.

- seekTo

  Seeks to the specified time point for playback. The HiPlayer calls the parser API fmt_seek_pts, specifies the stream index, sought time point, and seek direction, and asks the parser to seek the corresponding position and start to parse frame data from the

position. After fmt_seek_pts returns a success code, the HiPlayer clears the AVPLAY audio/video buffer to ensure that the stream data read after the seek operation is played.

If the seek operation is performed during the HLS network VOD, the HLS parser is called. The HLS parser finds the segment nearest to the sought position based on the sought time point and segment information, ends the previous network link, creates a network link, requests for new segment data, and starts to parse media frame data from the segment. If the sought time point exceeds the HLS duration, the playback is exited.

The seek operation is asynchronous. After the application issues the seek command, the MediaPlayer notifies the application of seek completion by reporting the MEDIA_SEEK_COMPLETE event.

- reset

  Resets the player. Performing this operation destroys the MediaPlayer instance, and the corresponding HiMediaPlayer operation destroys the HiPlayer. The HiPlayer asks the parser to terminate the current operation and exits quickly.

4.4.2.3 Open-Source Code

Currently the open-source code includes the HLS, HTTP, and TCP protocols, all of which are developed based on the FFMPEG framework and provide the HLS playback functions. When the playback starts, the three protocols are registered into the FFMPEG Demux/protocol list by calling the ffmpeg register interface. When HLS or HTTP network stream media files are played, the protocols are traversed until the appropriate protocol is found, and the protocol is used to create the network link and play HLS streams.

Directory Structure

The directory structure of the open-source code is as follows:

```
device/hisilicon/bigfish/external/libformat_open/
├── Android.mk
├── hls.c
├── hls_key_decrypt.c
├── hls_read.c
├── http.c
├── svr_allformat.c
├── svr_iso_dec.c
├── svr_udf_protocol.c
└── tcp.c
```

The path for storing the open-source code is **device/hisilicon/bigfish/external/libformat_open**. **hls.c**, **hls_key_decrypt.c**, and **hls_read.c** implement playback of HLS streams. **http.c** and **tcp.c** implement transmission over the network for HTTP clients. **svr_allformat.c** can register the HLS, HTTP, and TCP protocols into the FFMPEG Demux/protocol list. **svr_iso_dec.c** and **svr_udf_protocol.c** can be ignored.

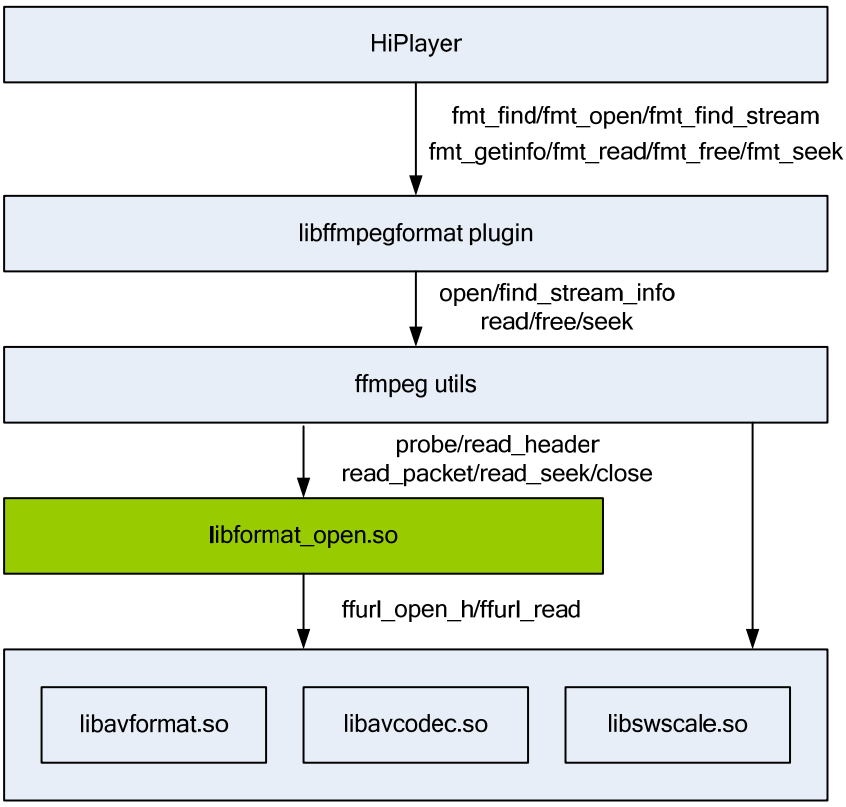
Compilation

libformat_open is an independent library. After modification, you can run the **mm** command in the **libformat_open** directory to generate **libformat_open.so**, and push the library to the **/system/lib** directory of the board by using the Android debug bridge (ADB). Then you can restart the media service and perform tests.

HLS Framework

Figure 4-8 shows the relationship between the libformat_open plug-in in the HLS framework and the FFMPEG/libffmpegformat plug-in.

Figure 4-8 Relationship between the libformat_open plug-in and FFMPEG/libffmpegformat plug-in



The implementation of libformat_open depends on the libavformat, libavcodec, and libswscale libraries. ffmpeg utils is an FFMPEG API, which provides the seek and packet framing functions.

As shown in Figure 4-8, an HLS network stream is played as follows:

Step 1 The HiPlayer calls the libffmpegformat plug-in to open the media file.

Step 2 The libffmpegformat plug-in calls ffmpeg utils to search for the appropriate protocol and parser, such as the HLS and HTTP in **libformat_open.so**, and then creates a network link and requests for the media file.

Step 3 ffmpeg utils calls the libformat_open HLS protocol to parse the M3U8 file to obtain the TS segment information.

Step 4 libformat_open creates a link, requests for the TS segment, and calls libavformat to parse TS data.

Step 5 During the playback, the libffmpegformat plug-in calls ffmpeg utils to read audio/video frame data, and ffmpeg utils calls libformat_open to read packet data and perform framing.

----End

HLS Internal Functions

The HLS internal functions are described as follows:

- HLS_probe

  Identifies the M3U8 format. This function checks whether the input data complies with the M3U8 file format and returns a grade, which determines whether to use the HLS protocol for playback.

- HLS_read_header

  Calls the M3U8 file to analyze HLS streams, obtains the HLS bit rate information and information about TS segments corresponding to each bit rate, and identifies live streams or VOD streams. After all segment information is obtained, this function opens the segment with the highest bit rate by default, and calls libavformat to parse the media.

- HLS_read_packet

  Reads frame data. ffmpeg utils calls this function to read packet data and perform framing, and this function calls libavformat to read packet data. Before reading data, this function checks whether the user has specified a bit rate for playback. If yes, the current bit rate file is closed, and a new bit rate file is opened and read.

- HLS_read_seek

  Seeks the segment nearest to the specified time point for playback. This function traverses the HLS list, finds the segment nearest to the time point specified by the user, and starts to play from the segment. The seek operation is not supported during the playback of live streams, and therefore a failure code is returned after this function is called. For VOD streams, if the sought time point exceeds the total duration of the HLS streams, the playback is terminated.

- HLS_close

  Exits the playback, closes the file, ends the network link, and releases the allocated resources.

- hlsParseM3U8

  Reads the M3U8 file by row, and parses the M3U8 file based on the HLS standard to obtain information such as the segment URL, target duration, segment duration, ENDLIST, and whether the stream is encrypted.

- hlsReadDataCb

  Data callback function. This function is called when HLS_read_packet is called. The logic of this function is complex, which involves switching the bit rate, requesting a new segment, and updating the M3U8 file in the live play scenario. The main logic of the HLS function is implemented in this function. In normal cases, this function returns the raw media data read from the HTTP protocol for the libavformat parser.

- hlsOpenSegment

  Requests for new segment data. For the HLS AES-128 encrypted streams, considering the obtaining method of the customer-extended AES-128 keys, this function first calls the adaptation API CLIENT_Int/CLIENT_GetKey to obtain the key. If the operation fails, the key is then obtained by using a common method.

- hlsReadLine

  Returns a row in the text file to facilitate the parsing of M3U8 files. This function is dedicated to M3U8 text files.

- hlsNewHlsStream

  Creates a new HLS stream. This function is dedicated for the HLS streams with multiple bit rates.

HLS Internal Data Structure

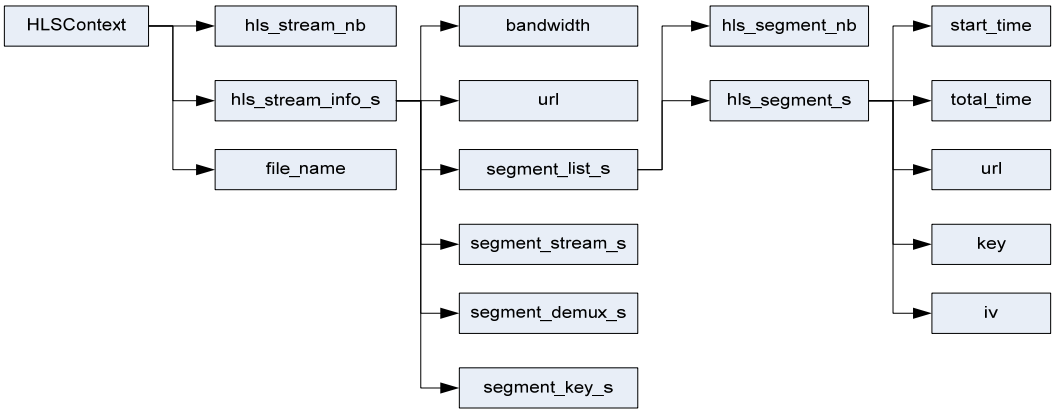
Table 4-1 describes the attributes of the HLS internal data structure.

Table 4-1 HLS internal data structure

Attribute	Definition
HLSContext	HLS internal data structure, invisible to other parsers and protocols
hls_stream_nb	Number of bit rates of the HLS streams. For example, if there are three bit rates, this attribute is 3.
hls_stream_info_s	Information about the segment and bandwidth corresponding to each bit rate in the HLS streams, that is, the number of level 2 M3U8 files
file_name	Level 1 M3U8 file name+path, that is, the URL
bandwidth	Bit rate information
url	URL of the level 2 M3U8 file
segment_list_s	Information about the segments contained in the level 2 M3U8 file
segment_stream_s	Current opened segment IO
segment_demux_s	Information about the parser of the current opened segment
segment_key_s	Key IO corresponding to the current opened segment, valid only for AES-128 encrypted streams
hls_segment_nb	Number of segments in streams with a specific bit rate
hls_segment_s	Information about segments in streams with a specific bit rate
start_time	Start time of a segment, which equals the total duration of the previous segments
total_time	Total segment duration, obtained by parsing the M3U8 file
url	Segment URL
key	URL of the decryption key of a segment
iv	IV vector of the decryption key of a segment

Figure 4-9 shows the HLS internal data structure.

Figure 4-9 HLS internal data structure



HLS Sample

The directory structure of the sample is as follows:

```
hls_sample/
├── hisilicon_key.txt
└── hls_multi_rate
    ├── main_playlist.m3u8
    ├── stream-128000
    │   ├── stream-128000-1.ts
    │   ├── stream-128000-2.ts
    │   ├── stream-128000-3.ts
    │   └── stream-128000.m3u8
    ├── stream-1708000
    │   ├── stream-1708000-1.ts
    │   ├── stream-1708000-2.ts
    │   ├── stream-1708000-3.ts
    │   └── stream-1708000.m3u8
    └── stream-764000
        ├── stream-764000-1.ts
        ├── stream-764000-2.ts
        ├── stream-764000-3.ts
        └── stream-764000.m3u8
```

The level 1 M3U8 file **main_playlist.m3u8** is as follows:

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=128000
stream-128000/stream-128000.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=764000
stream-764000/stream-764000.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1708000
```

```
stream-1708000/stream-1708000.m3u8
```

- This HLS stream has three bit rates, that is, three level 2 M3U8 files. The bit rates are as follows:
  - 128000 bit/s
  - 764000 bit/s
  - 1708000 bit/s
- The value of **hls_stream_nb** is 3, that is, there are three groups of **hls_stream_info_s** information.
- **file_name** is the *stream root directory*+**hls_sample/hls_multi_rate/main_playlist.m3u8**.

  For example, if an apache server is installed on a machine, and hls_sample is stored in the root directory of the apache server, **file_name** is as follows:

  http://10.67.225.7/htdocs/hls_sample/hls_multi_rate/main_playlist.m3u8

The attribute values in the follow-up HLS data structures can be parsed based on the contents of the level 2 M3U8 file. The level 2 M3U8 file is as follows:

```
#EXTM3U
#EXT-X-KEY:METHOD=AES-128,
URI="https://10.67.225.27/htdocs/hls_sample/hisilicon_key.txt",IV=0x00000
00000000000000000000000000
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXTINF:10,
stream-128000-1.ts
#EXTINF:10,
stream-128000-2.ts
#EXTINF:10,
stream-128000-3.ts
#EXT-X-ENDLIST

#EXTM3U
#EXT-X-KEY:METHOD=AES-128,
URI="https://10.67.225.27/htdocs/hls_sample/hisilicon_key.txt",IV=0x00000
00000000000000000000000000
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXTINF:10,
stream-764000-1.ts
#EXTINF:10,
stream-764000-2.ts
#EXTINF:10,
stream-764000-3.ts
#EXT-X-ENDLIST

#EXTM3U
```

```
#EXT-X-KEY:METHOD=AES-128,
URI="https://10.67.225.27/htdocs/hls_sample/hisilicon_key.txt",IV=0x00000
000000000000000000000000
#EXT-X-TARGETDURATION:10
#EXT-X-VERSION:3
#EXTINF:10,
stream-1708000-1.ts
#EXTINF:10,
stream-1708000-2.ts
#EXTINF:10,
stream-1708000-3.ts
#EXT-X-ENDLIST
```

4.4.2.4 HLS Customization

Take the live video of the China Mobile video base as an example. Except for the live play and VOD functions, the time shifting function is also defined (for example, if the current time is 12:00, users can watch the 8:00 news by using the time shifting function). Because the HLS does not define the time shifting standard, the video base implements the time shifting service by adding key fields to the URL and defining the standard.

Time Shifting

The following is the time shifting URL of the video base:

```
"http://ips.itv.cmvideo.cn/140_0_3736680_0.m3u8?id=3736680&scId=&isChanne
l=true&time=300&serviceRegionIds=140&codec=ALL&quality=100&vSiteCode=&off
set=0&livemode=2&starttime=20141024T095821.00Z"
```

where

- The value **2** of **livemode** indicates time shifting, and the value **1** indicates live play.
- **starttime** is the time shifting start time. **20141024T095821** indicates 09:58:21 on October 24, 2014.

A time shifting program has its duration, for example, two hours. According to the standard of the video base, if the playback of a time shifting program is complete, the next time shifting streams need to be played immediately. To play the next time shifting streams, the URL needs to be updated, that is, the **starttime** field in the previous URL needs to be updated as follows:

New **starttime** = **starttime** in the previously specified URL + Total duration of segments that have been played

Take the preceding URL as an example. If the obtained total segment duration is 1 hour and 50 minutes based on the URL, after the 1 hour and 50 minutes segments are downloaded, **starttime** in the URL should be updated as follows:

20141024T095821.00Z + 110 minutes = 20141024T114821.00Z

Then the new URL is used to update the M3U8 list.

For the time shifting function of the video base, the new URL needs to be calculated after the first obtained segments are downloaded, and the new URL is used to request for a segment.

In the open **hls.c** file:

- In the live play scenario, the M3U8 list is updated in the hlsReadDataCb function. **if (v->seg_list.hls_seg_cur >= v->seg_list.hls_seg_start + v->seg_list.hls_segment_nb)** in the function indicates whether all segments are downloaded.
- In the time shifting scenario, **if (v->seg_list.hls_seg_cur >= v->seg_list.hls_seg_start + v->seg_list.hls_segment_nb)** is executed after all the first obtained segments are downloaded, and updating the URL can be implemented in this case. For standard HLS live streams, the initial playback URL is used to update the M3U8 list after the existing segments are downloaded or a specific time period elapses.

In the time shifting scenario of the video base, the URL needs to be updated first after the segments are downloaded, and the new URL is used to request for new time shifting segments. For details, see the implementation of hlsReadDataCb in **hls.c**.

Secondary Development

As the HLS standards are evolving, and the HLS is widely used, new functions can be extended easily by adding private definitions. The implementation of **hls.c** is independent, and therefore it can be modified easily to adapt to the functions of private definitions. A private data structure HLSContext (for details, see **avformat.h** in **device/hisilicon/bigfish/external/ffmpeg/libavformat**) is defined within the HLS. If you want to add a new attribute, define it at the end of HLSContext. You cannot modify other data structures or add one among defined data structures.

4.4.3 Smooth Streaming

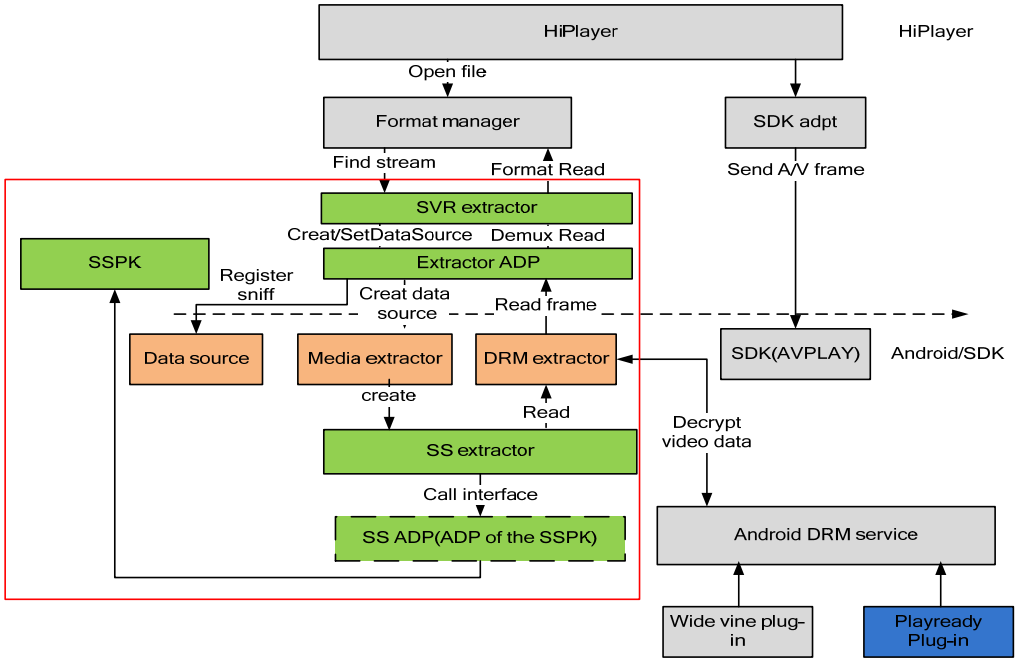
Scenario

The Smooth Streaming porting kit (SSPK) is an adaptive streaming solution provided by Microsoft. It is used to enable the stream quality to dynamically adapt to the user network conditions, bringing best user experience.

Working Process

Figure 4-4 shows the software architecture of the current HiSilicon solution. **libextractor_demux.so** is the Smooth Streaming parser. Figure 4-10 shows its internal working process.

Figure 4-10 Working process of the Smooth Streaming modules



The green parts in the red rectangle are the **libextractor_demux.so** parser.

- SVR extractor and extractor ADP

  This part is the open-source code and is stored in **device\hisilicon\bigfish\hidolphin\component\plugin_extractor**. The directory structure is as follows:

  ```
  ├── Android.mk
  ├── demux
  │   └── svr_extractor.c
  ├── extrator_adp
  │   ├── svr_extrator_adp.cpp
  │   ├── svr_extrator_adp.h
  │   └── svr_extrator_intf.h
  ├── hi_type.h
  ├── project.config
  └── sample_test.c
  ```

  The source file of the SVR extractor is **svr_extractor.c**. The SVR extractor is used to define parser APIs and connects to the HiPlayer format manager.

  The source file of the extractor ADP is **svr_extractor_adp.cpp**. This module has the following functions:

  - Provides C interface for the upper-layer module (SVR extractor).
  - Creates data sources and reads data. (This part is the open-source code. For details, see **svr_extrator_adp.cpp**.)

- SS extractor

This module implements adaptation of Android inherent components to the SSPK. The main process is as follows:

- Creates an SSPK parser.

```
s32Ret = HI_SVR_SMOOTHSTREAMING_Create(&struInitParam,
&mHiSSPlayer);
```

- Opens the file source.

```
s32Ret = HI_SVR_SMOOTHSTREAMING_Open(mHiSSPlayer, &mMediaParam);
```

- Obtains the encryption attribute of the stream.

```
s32Ret = HI_SVR_SMOOTHSTREAMING_GetParam(mHiSSPlayer,
HI_SVR_SMOOTHSTREAMING_ATTR_DRM_DOMAIN,&stDrmHeaderInfo);
```

- Starts Smooth Streaming for parsing the stream.

```
s32Ret = HI_SVR_SMOOTHSTREAMING_Play(mHiSSPlayer);
```

- Creates a data read thread.

```
createThreadEtc(SSExtractor::sched_thread, this,
"ssextrator_thread", ANDROID_PRIORITY_DEFAULT, 0, NULL);
```

- Reads data parsed by the SSPK.

```
status = HI_SVR_SMOOTHSTREAMING_ReadFrame(mHiSSPlayer,
&pstFmtFrame);
```

- SS ADP

  SS ADP is the adaptation layer of the SSPK. It provides APIs for external components. For details about the APIs, see **hi_svr_smoothstreaming.h** and **hi_svr_smoothstreaming_adp.h**.

- SSPK

  The SSPK is provided by Microsoft.

- DRM extractor

  The DRM extractor is an inherent Android component. It is used to transmit data and decrypt encrypted streams by calling the decryption interface. The details are as follows:

  - Obtains raw data.

```
originalMediaSource = mOriginalExtractor->getTrack(index);
```

  - Checks whether the stream is encrypted.

    No: `return originalMediaSource;`

    Yes:

```
MediaSource = new DRMSource(originalMediaSource,…);
   return MediaSource;
```

  - Reads data.

    The unscrambled streams are read directly.

    For encrypted streams, mOriginalMediaSource->read(buffer, options) is called to read the original data, and then mDrmManagerClient->decrypt(...) is called to decrypt the data.

- PlayReady

  PlayReady is the decryption library. For details, see the *Android PlayReady Development Guide*.

Notes

- Fees

  End-user product suppliers must purchase licenses for the SSPK from Microsoft. For details about the fees, go to http://www.microsoft.com/en-us/mediaplatform/sspk.aspx.

- Compilation

  Before compilation, ensure the following configurations in **device/hisilicon/Hi3798MV100/device.mk**:

  ```
  PRODUCT_PROPERTY_FOR_DRM_CLIENT=true
  PRODUCT_PROPERTY_FOR_DRM_SERVICE=true
  ```

- Supported functions
  - Adaptive switchover among multiple bit rates

    Videos with different bit rates can be switched based on the current network bandwidth. The minimum switch interval is the duration of a segment (typically 2s, depending on the stream segment duration provided by the server). The quality level of the video to be switched to is provided by the server. For example, the official Smooth Streaming server of Microsoft (http://playready.directtaps.net/smoothstreaming/) provides the following quality levels:

    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_230.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_331.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_477.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_688.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_991.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_1427.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_2056.ismv
    http://playready.directtaps.net/smoothstreaming/SSWSS720H264/SuperSpeedway_720_2962.ismv

    The optimal stream for the current network conditions is selected from the preceding listed streams (with the same contents but different quality or bit rates).
  - Supported formats

    Currently the supported audio/video encoding formats include VC1/WMA and H.264/AAC.
  - Table 4-2 describes the supported playback control functions.

Table 4-2 Playback control

Function	Network Live Play	Network VOD
Play	√	√
Pause	√	√
Fast-forward	×	√
Rewind	×	√
Seek	×	√
Stop	√	√

Note that √ indicates supported and × indicates not supported.

- The playback of soft PlayReady encrypted streams is supported. For details, see the *Android PlayReady Development Guide*.

- APIs

  For details about the APIs, see the APIs of the Android inherent media player.

- Tests

  Preparations:

  - Ensure that **getprop drm.service.enabled** and **getprop drm.client.enabled** are **true**.
  - Ensure that the following certificates exist in **data/playready/prpd**:

    bgroupcert.dat

    zgpriv.dat

    devcerttemplate.dat

    priv.dat
  - Ensure that **DrmAssist-Recommended.apk** exists in **system/app**.
  - Ensure that the PlayReady library exists. For details, see section 3.4.2 in the *Android PlayReady Development Guide*.
  - Open the DRM (both the icon and name are DRM), and select **UrlPlay**. The following web page is opened by default:

    http://playready.directtaps.net/smoothstreaming/
  - Select the stream to be played, and click the corresponding manifest file to play the stream.

    Streams that are not listed on the web page (http://playready.directtaps.net/smoothstreaming/) can be played by running commands as follows (because DrmAssist-Recommended.apk is only a demo, some functions are not implemented, such as seek and play at multiple times of the normal speed. You are advised to play unencrypted streams by using commands. For encrypted stream, you can obtain the license by using the APK and then play the streams by running commands):

    ```
    am start -n
    com.hisilicon.android.videoplayer/.activity.VideoActivity -d
    http://116.199.4.50/CDN/101_GZTV.isml/manifest
    ```

Sample

Step 1 Create a player.

```
mVideoView = (VideoView) findViewById(R.id.videoView1);
```

Step 2 Set the error listener.

```
mVideoView.setOnErrorListener(this);
```

Step 3 Set the media controller.

```
mVideoView.setMediaController(ctlr);
```

Step 4 Set the path of the video to be played.

```
mVideoView.setVideoPath(path);
```

Step 5 Obtain the license (for details, see chapter 4 "Application Development" in the *Android PlayReady Development Guide*).

```
response = drmClient.acquireDrmInfo(request);
```

Step 6 Start to play the file.

```
mVideoView.start();
```

----End

5 HiDLNA

5.1 Overview

The Digital Living Network Alliance (DLNA) is an organization intended for the home device interoperability and multimedia interaction. The HiDLNA is the standard implementation of the DLNA protocol.

The HiDLNA consists of the following three independent applications:

- Digital media server (DMS): This application is used for sharing media files on the STB, making the media files available to digital media players (DMPs) and digital media controllers (DMCs). The DMPs and DMCs can also play those shared files.
- Digital media renderer (DMR): This application provides media control and connection management services for receiving media files pushed by the DMC and controlling the playing.
- DMP: This application searches for media-shared servers, browses media files on servers, and plays selected files.

The DMC is a control point and standard implementation of the DLNA. It is used to detect DMSs and DMRs, access resources in DMSs, and control DMRs for playback. The DMC is not described here. This chapter describes the usage of HiDLNA APIs on Android and Linux to guide the development of HiDLNA functions based on the HiSilicon chip platforms. Section 5.2 "Android AIDL APIs" describes APIs on Android, and section 5.3 "Linux APIs" describes APIs on Linux.

5.2 Android AIDL APIs

The Android interface definition language (AIDL) is an interprocess communication (IPC) interface definition language for Android internal processes.

On Android, each application runs in its own process space, and one process cannot normally access the memory of another process. However, in the development of Android applications, objects must be transferred across processes. In this case, the AIDL API mechanism is provided to implement and regulate the communication between Android internal processes.

When the AIDL APIs are used, the communication message on one process is converted into the AIDL protocol message, transmitted to another process, and then converted into the corresponding object. The bidirectional conversion of messages is implemented by the agent class, which is generated by the Android compiler and transparent to R&D engineers.

To use the AIDL, perform the following steps:

Step 1 Create an AIDL file, which defines the methods and domains that apply to clients. You can quote interfaces that are defined in other AIDL files.

Step 2 Add the AIDL file to the compiler.

Step 3 Implement the internal abstract class of the defined AIDL API.

Step 4 Provide the API to clients and describe how to call the AIDL API.

Android has detailed the procedure for using the AIDL. For details, see http://developer.android.com/guide/components/aidl.html.

Before using the HiDLNA AIDL APIs, you have to know about the concept of the Android service.

A service is an application component that can perform long-running operations in the background and does not provide a user interface. Other application component can start a service, and the service continues to run in the background even if the user switches to another application. Additionally, a component can bind to a service to interact with it and even perform IPC.

HiDLNA DMSs and DMRs exist in the form of services on Android and provide AIDL APIs. The following sections describe the usage of HiDLNA AIDL APIs. Before using the AIDL APIs, you must get familiar with the definition of AIDL APIs.

----End

5.2.2 Usage of HiDLNA Android AIDL APIs

5.2.2.1 Overview

The following is the procedure for using HiDLNA AIDL APIs by taking **IDMRServer.aidl** provided by the DMR as an example.

Step 1 Create a dmrBinder instance in the service and register the dmrBinder with the Service Manager by calling the following methods:

```
IDMRServer.Stub dmrBinder = new DMRBinder(Context);
ServiceManager.addService("dmr", dmrBinder);
```

Step 2 Obtain the interface to which the DMR service is bound by using the registered name **dmr** in another component. You can call the following method:

```
IDMRServer dmrServer =
IDMRServer.Stub.asInterface(ServiceManager.getService("dmr"));
```

Step 3 To start the DMR service, call the following method:

```
dmrServer.create()
```

----End

5.2.2.2 Using DMP AIDL APIs

The classes provided by the DMP are stored in **com.hisilicon.dlna.file**. To use the DMP AIDL APIs, perform the following steps:

Step 1 Obtain the DMS device list by calling the following method:

```
List<DMSDevice> searchedDeviceList = DMSDevice.searchDMSDevices()
```

List<DMSDevice> is returned, which is the obtained DMS device list.

Step 2 Obtain the DMS level-1 directory list by calling the following method:

```
DMSFile[] files = DMSFile.getTopFiles((DMSDevice)device)
```

DMSFile[] is returned, which is the obtained DMS level-1 directory list.

Step 3 Obtain the subdirectory list of the DMS level-1 directory by calling the following method:

```
DMSFile[] files = ((DMSFile)focusFile).listFiles()
```

DMSFile[] is returned, which is the obtained subdirectory list of the DMS level-1 directory.

----End

NOTE

The directory list is obtained in batches. That is, if a directory has 2000 subdirectories, the first 200 subdirectories are returned as the first batch, and the next 200 subdirectories are returned as the second batch. If you want to obtain from the first subdirectory, call DMSFile.reset().

5.2.2.3 Using DMR AIDL APIs

The classes provided by the DMR are stored in **com.hisilicon.dlna.file**. To use the DMR AIDL APIs, perform the following steps:

Step 1 Create a dmrBinder instance in the service and register the dmrBinder with the Service Manager by calling the following methods:

```
IDMRServer.Stub dmrBinder = new DMRBinder(Context);
ServiceManager.addService("dmr", dmrBinder);
```

The preceding methods are used to start the DMR service and register it with the system service so that the DMR service can be used in the client. The parameter **dmr** specifies the DMR to be started.

Step 2 Obtain the interface to which the DMR service is bound by using the registered name **dmr** in another component. You can call the following method:

```
IDMRServer dmrServer =
IDMRServer.Stub.asInterface(ServiceManager.getService("dmr"));
```

Before performing operations provided by the DMR, you must obtain the DMR service. The parameter **dmr** specifies the DMR service to be obtained.

Step 3 Call the following methods to control the DMR:

1. To start the DMR service, call the following method:
   dmrServer.create("dmrName")

dmrName specifies the DMR name.

2. To stop the DMR service, call the following method:

   dmrServer.destroy();

3. To notify the client of the pause status of the player, call the following callback method:

   dmrServer.getDMRCallback().pauseNotify()

   When the player pauses during playing, this method is used to notify the client of the pause status.

----End

5.2.2.4 Using DMS AIDL APIs

The classes provided by the DMS are stored in **com.hisilicon.dlna.file**. To use the DMS AIDL APIs, perform the following steps:

Step 1 Create a dmrBinder instance in the service and register the dmrBinder with the Service Manager by calling the following methods:

```
IDMSServer.Stub dmsBinder = new DMSBinder(Context);
ServiceManager.addService("dms", dmsBinder);
```

The preceding methods are used to start the DMS service and register it with the system service so that the DMS service can be used in the client. The parameter **dms** specifies the DMS to be started.

Step 2 Obtain the interface to which the DMR service is bound by using the registered name **dmr** in another component. You can call the following method:

```
IDMSServer dmsServer =
IDMSServer.Stub.asInterface(ServiceManager.getService("dms"));
```

Before performing operations provided by the DMS, you must obtain the DMS service. The parameter **dms** specifies the DMS service to be obtained.

Step 3 Call the following methods to control the DMS:

1. To start the DMS, call the following method:

   dmsBinder.create("dmsName", "/data");

   dmsName specifies the DMS name, and /data specifies the DMS shared directory.

2. To add a shared directory, call the following method:

   dmsBinder.addShareDir("dmsName", "/sdcard/");

   dmsName specifies the DMS name, and /sdcard/ specifies the DMS shared directory to be added.

----End

5.2.3 Definitions of HiDLNA Android AIDL APIs

5.2.3.1 Overview

The AIDL APIs provided by the HiDLNA are used for registering and starting the DMS and DMR services, and enabling the device searching and file browsing functions of the DMP, implementing the DLNA functions on HiSilicon chip platforms.

5.2.3.2 HiDLNA AIDL API Functions

The HiDLNA provides the following AIDL APIs:

- IDMRPlayerController: Obtains the DMR player information. This API is a constructor function.
- getDuration: Obtains the total DMR playback duration.
- getCurrentPosition: Obtains the current DMR playback duration.
- IDMRCallback: Returns the current DMR player status. This API is a constructor function.
- playNotify: Returns the playing status of the player.
- pauseNotify: Returns the pause status of the player.
- seekNotify: Reserved.
- stopNotify: Returns the stop status of the player.
- setCurTime: Reserved.
- getCurTime: Reserved.
- setAllTime: Reserved.
- getAllTime: Reserved.
- IDMRServer: Controls the DMR service. This API is a constructor function.
- create: Starts the DMR or DMS.
- destroy: Destroys the DMR or DMS.
- setName: Sets the DMR name upon the first startup.
- setDeviceName: Sets the DMR and DMS names after startup.
- startActivity: Starts an activity.
- sendBroadcast: Sends a broadcast.
- IDMRCallback: Obtains the DMR callback interface.
- registerPlayerController: Registers a player.
- unregisterPlayerController: Deregisters a player.
- IDMRPlayerController: Obtains a player.
- IDMSServer: Controls the DMS service. This API is a constructor function.
- addShareDir: Adds shared directories for the DMS.
- delShareDir: Deletes shared directories for the DMS.

interface IDMRPlayerController

[Purpose]

Obtains the DMR player information, including the total and current playback duration. This API is a constructor function.

[Syntax]

```
interface IDMRPlayerController
{
    int getDuration();
    int getCurrentPosition();
}
```

[Description]

The player implements this API and registers it with the DMR server by calling registerPlayerController. This API is used to obtain player information.

[Parameter]

None

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

int getDuration()

[Purpose]

Obtains the total playback duration. This API is the implementation of the constructor function IDMRPlayerController.

[Syntax]

int getDuration();

[Description]

This API is used to obtain the total playback duration.

[Parameter]

None

[Return Value]

Return Value	Description
A value greater than	Total playback duration, in ms

Return Value	Description
or equal to 0	
Other specifications	Error

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRPlayerController

[Note]

None

[Example]

None

int getCurrentPosition()

[Purpose]

Obtains the current playback duration. This API is the implementation of the constructor function IDMRPlayerController.

[Syntax]

```
int getCurrentPosition ();
```

[Description]

This API is used to obtain the current playback duration.

[Parameter]

None

[Return Value]

Return Value	Description
A value greater than or equal to 0	Current playback duration, in ms
Other specifications	Error

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRPlayerController

[Note]

None

[Example]

None

interface IDMRCallback

[Purpose]

Returns the current player status. This API is a constructor function.

[Syntax]

```
interface IDMRCallback
{
    void playNotify();
    void pauseNotify();
    void seekNotify(int pos);
    void stopNotify();
    void setCurTime(int time);
    int getCurTime();
    void setAllTime(int time);
    int getAllTime();
}
```

[Description]

This API is used to notify the client of the player status change.

[Parameter]

None

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

[Example]

None

void playNotify()

[Purpose]

Returns the player status to the DMC when the player starts playing. This API is the implementation of the constructor function IDMRCallback.

[Syntax]

void playNotify();

[Description]

This API is used to return the player status to the client when the player starts playing.

[Parameter]

None

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRCallback

[Note]

None

[Example]

None

void pauseNotify ()

[Purpose]

Returns the player status to the client when the player pauses. This API is the implementation of the constructor function IDMRCallback.

[Syntax]

void pauseNotify ();

[Description]

This API is used to return the player status to the client when the player pauses.

[Parameter]

None

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRCallback

[Note]

None

[Example]

None

void seekNotify (int pos)

[Purpose]

Returns the player status to the client when the player is seeking a time point. This API is the implementation of the constructor function IDMRCallback.

[Syntax]

void seekNotify (int pos);

[Description]

This API is used to return the player status to the client when the player is seeking a time point.

[Parameter]

Parameter	Description	I/O
pos	Sought time point	I

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRCallback

[Note]

This API is reserved and invalid.

[Example]

None

void stopNotify ()

[Purpose]

Returns the player status to the client when the player stops. This API is the implementation of the constructor function IDMRCallback.

[Syntax]

void stopNotify ();

[Description]

This API is used to return the player status to the client when the player stops.

[Parameter]

None

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRCallback

[Note]

None

[Example]

None

void setCurTime(int time)

[Purpose]

Sets the current playback duration. This API is the implementation of the constructor function IDMRCallback.

[Syntax]

void setCurTime(int time);

[Description]

This API is used to set the current playback duration.

[Parameter]

Parameter	Description	I/O
time	Current playback duration	I

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRCallback

[Note]

This API is reserved and invalid.

[Example]

None

int getCurTime ()

[Purpose]

Obtains the current playback duration. This API is the implementation of the constructor function IDMRCallback.

[Syntax]

int getCurTime ();

[Description]

This API is used to obtain the current playback duration.

[Parameter]

None

[Return Value]

Return Value	Description
A value greater than or equal to 0	Current playback duration
Other specifications	Error

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRCallback

[Note]

This API is reserved and invalid.

[Example]

None

void setAllTime (int time)

[Purpose]

Sets the total playback duration. This API is the implementation of the constructor function IDMRCallback.

[Syntax]

void setAllTime (int time);

[Description]

This API is used to set the total playback duration.

[Parameter]

Parameter	Description	I/O
time	Total playback duration	I

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRCallback

[Note]

This API is reserved and invalid.

[Example]

None

int getAllTime ()

[Purpose]

Obtains the total playback duration. This API is the implementation of the constructor function IDMRCallback.

[Syntax]

int getAllTime ();

[Description]

This API is used to obtain the total playback duration.

[Parameter]

None

[Return Value]

Return Value	Description
A value greater than or equal to 0	Total playback duration
Other specifications	Error

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file, IDMRCallback

[Note]

This API is reserved and invalid.

[Example]

None

interface IDMRServer

[Purpose]

Starts or destroys the DMR. This API is a constructor function.

[Syntax]

```
interface IDMRServer
{
    boolean create(String name);
    boolean destroy();
    void setName(String name);
    boolean setDeviceName(String dmrName, String name);
    void startActivity(in Intent intent);
    void sendBroadcast(in Intent intent);
    IDMRCallback getDMRCallback();
    void registerPlayerController(IDMRPlayerController playerController);
    void unregisterPlayerController();
    IDMRPlayerController getDMRPlayerController();
}
```

[Description]

This API is used to start or destroy the DMR.

[Parameter]

None

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

For details, see section 5.2.2.3 "Using DMR AIDL APIs."

boolean create(String name)

[Purpose]

Starts the DMR service. This API is the implementation of the constructor function IDMRServer.

[Syntax]

boolean create(String name);

[Description]

This API is used to start the DMR service.

[Parameter]

Parameter	Description	I/O
time	Total playback duration	I

[Return Value]

Return Value	Description
true	Success
false	Failure

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

For details, see section 5.2.2.3 "Using DMR AIDL APIs."

boolean destroy ()

[Purpose]

Destroys the DMR service. This API is the implementation of the constructor function IDMRServer.

[Syntax]

boolean destroy();

[Description]

This API is used to destroy the DMR service.

[Parameter]

None

[Return Value]

Return Value	Description
true	Success
false	Failure

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

void setName (String name)

[Purpose]

Sets the DMR name. This API is the implementation of the constructor function IDMRServer.

[Syntax]

void setName (String name);

[Description]

This API is used to change the DMS name.

[Parameter]

Parameter	Description	I/O
name	Specifies the DMR name.	I

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

void setDeviceName (String name)

[Purpose]

Changes the DMR name after startup. This API is the implementation of the constructor function IDMRServer.

[Syntax]

boolean setDeviceName(String dmrName, String name);

[Description]

This API is used to change the DMR name.

[Parameter]

Parameter	Description	I/O
dmrName	New DMR name	I
name	Original DMR name	I

[Return Value]

Return Value	Description
true	Success
False	Failure

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

void startActivity(in Intent intent)

[Purpose]

Starts an activity. This API is the implementation of the constructor function IDMRServer.

[Syntax]

void startActivity(in Intent intent);

[Description]

This API is used to change the DMS name.

[Parameter]

Parameter	Description	I/O
Intent intent	Activity name	I

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

void sendBroadcast(in Intent intent)

[Purpose]

Sends a broadcast. This API is the implementation of the constructor function IDMRServer.

[Syntax]

void sendBroadcast(in Intent intent);

[Description]

This API is used to send a broadcast.

[Parameter]

Parameter	Description	I/O
Intent intent	Activity name	I

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

IDMRCallback getDMRCallback()

[Purpose]

Obtains the DMR callback interface. This API is the implementation of the constructor function IDMRServer.

[Syntax]

IDMRCallback getDMRCallback();

[Description]

This API is used to obtain the DMR callback interface.

[Parameter]

None

[Return Value]

Return Value	Description
IDMRCallback	IDMRCallback constructor function.

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

For details, see the IDMRCallback interface.

void registerPlayerController(IDMRPlayerController playerController)

[Purpose]

Registers a player. This API is the implementation of the constructor function IDMRServer.

[Syntax]

void registerPlayerController(IDMRPlayerController playerController)

[Description]

This API is used to register a player.

[Parameter]

Parameter	Description	I/O
playerController	IDMRPlayerController constructor function	I

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

This API is used with unregisterPlayerController.

[Example]

For details, see the IDMRPlayerController constructor function.

void unregisterPlayerController ()

[Purpose]

Deregisters a player. This API is the implementation of the constructor function IDMRServer.

[Syntax]

void unregisterPlayerController ()

[Description]

This API is used to deregister a player. It must be called after the player is closed.

[Parameter]

None

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

- This API must be called after the player is closed.
- This API is used with registerPlayerController.

[Example]

None

IDMRPlayerController getDMRPlayerController()

[Purpose]

Obtains a player. This API is the implementation of the constructor function IDMRServer.

[Syntax]

IDMRPlayerController getDMRPlayerController()

[Description]

This API is used to obtain a player.

[Parameter]

None

[Return Value]

Return Value	Description
IDMRPlayerController	IDMRPlayerController constructor function

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

interface IDMSServer

[Purpose]

This API is used to control the DMS service.

[Syntax]

```
interface IDMSServer
{
    boolean create(String dmsName, String dir);
    boolean destroy();
    boolean setDeviceName(String dmsName, String name);
    boolean addShareDir(String dmsName, String dir);
boolean delShareDir(String dmsName, String dir);
}
```

[Description]

This API is used to control the DMS service.

[Parameter]

None

[Return Value]

None

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

For details, see section 5.2.2.4 "Using DMS AIDL APIs."

boolean create(StringdmsName, Stringdir)

[Purpose]

Starts the DMS service. This API is the implementation of the constructor function IDMSServer.

[Syntax]

boolean create(String dmsName, String dir);

[Description]

This API is used to start the DMS service.

[Parameter]

Parameter	Description	I/O
dmsName	DMS name	I
dir	DMS shared directory	I

[Return Value]

Return Value	Description
true	Success
False	Failure

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

For details, see section 5.2.2.3 "Using DMR AIDL APIs."

boolean destroy ()

[Purpose]

Stops the DMS service. This API is the implementation of the constructor function IDMSServer.

[Syntax]

boolean destroy();

[Description]

This API is used to stop the DMS service.

[Parameter]

None

[Return Value]

Return Value	Description
true	Success
False	Failure

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

boolean setDeviceName(String dmsName, String name)

[Purpose]

Changes the DMS name. This API is the implementation of the constructor function IDMSServer.

[Syntax]

boolean setDeviceName(String dmsName, String name);

[Description]

This API is used to change the DMS name.

[Parameter]

Parameter	Description	I/O
dmsName	New DMS name	I
name	Original DMS name	I

[Return Value]

Return Value	Description
true	Success
False	Failure

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

boolean addShareDir(String dmsName, String dir)

[Purpose]

Adds the DMS shared directory. This API is the implementation of the constructor function IDMSServer.

[Syntax]

boolean addShareDir(String dmsName, String dir)

[Description]

This API is used to add the DMS shared directory.

[Parameter]

Parameter	Description	I/O
dmsName	DMS name	I
dir	DMS shared directory to be added	I

[Return Value]

Return Value	Description
true	Success
False	Failure

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

boolean delShareDir(String dmsName, String dir)

[Purpose]

Deletes the DMS shared directory. This API is the implementation of the constructor function IDMSServer.

[Syntax]

boolean delShareDir(String dmsName, String dir)

[Description]

This API is used to delete the DMS shared directory.

[Parameter]

Parameter	Description	I/O
dmsName	DMS name	I
name	DMS shared directory to be deleted	I

[Return Value]

Return Value	Description
true	Success
False	Failure

[Error Code]

None

[Requirement]

com.hisilicon.dlna.file

[Note]

None

[Example]

None

5.3 Linux APIs

5.3.1 Overview

This section is a supplement to the *HiDLNA API Development Reference*. This section describes how to implement the DMR, DMP, and DMS functions by calling corresponding APIs.

5.3.1.1 Application Scenarios of the DMS

Scenario 1

Share one or more directories on a DMS. Perform the following steps:

Step 1 Initialize the DMS protocol stack by calling the following method:

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

Step 2 Add the directories to be shared by calling the following APIs:

```
HI_DLNA_AddDmsSharedDir (path1,strlen(path1),HI_NULL_PTR);
HI_DLNA_AddDmsSharedDir (path2,strlen(path2), HI_NULL_PTR);
? ? ?
HI_DLNA_AddDmsSharedDir (pathx,strlen(pathx), HI_NULL_PTR);
```

----End

Scenario 2

Delete one or more shared directories on a DMS. This scenario is based on scenario 1. Perform the following steps:

Step 1 Initialize the DMS protocol stack by calling the following API:

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

Step 2 Add the directories to be shared by calling the following APIs:

```
HI_DLNA_AddDmsSharedDir (path1,strlen(path1), HI _NULL_PTR);
HI_DLNA_AddDmsSharedDir (path2,strlen(path2), HI _NULL_PTR);
? ? ?
HI_DLNA_AddDmsSharedDir (pathx,strlen(pathx), HI _NULL_PTR);
```

Step 3 Delete the shared directories by calling the following APIs:

```
HI_DLNA_DelDmsSharedDir (path1,strlen(path1), HI _NULL_PTR);
HI_DLNA_DelDmsSharedDir (path2,strlen(path2), HI _NULL_PTR);
? ? ?
HI_DLNA_DelDmsSharedDir (pathx,strlen(pathx), HI _NULL_PTR);
```

----End

Scenario 3

Exit a DMS. This scenario is based on scenario 1. You do not need to exit the DMS after deleting the shared directories.

CAUTION

Shared directories are automatically deleted after HI_DLNA_DestoryStack is called.

Perform the following steps:

Step 1 Initialize the DMS protocol stack by calling the following API:

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

Step 2 Add shared directories as required. If no shared directories are added, the start and stop of the DMS protocol stack are not affected.

Step 3 Exit the DMS protocol stack.

```
HI_DLNA_DestoryStack();
```

----End

Scenario 4

Change the destination directory for uploading. You need to start the DMS can then call the following APIs to change the directory for uploading. You can also edit the **dlna.ini** file. The change takes effect after the DMS restarts. Perform the following steps:

Step 1 Initialize the DMS protocol stack by calling the following API:

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

Step 2 Change the destination directory for uploading by calling the following API:

```
HI_DLNA_SetDefaultUploadPath(Path_For_Upload);
```

The directory is changed successfully. However, the change does not take effect on the currently running DMS.

----End

Scenario 5

Change a DMS device name. The DMS device name needs to be changed upon startup. Otherwise, the default name is displayed. Perform the following steps:

Step 1 Initialize the DMS protocol stack by calling the following API:

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMS);
```

Step 2 Set the DMS device name by calling the following API:

```
HI_DLNA_SetDeviceName(Name_For_DMS);
```

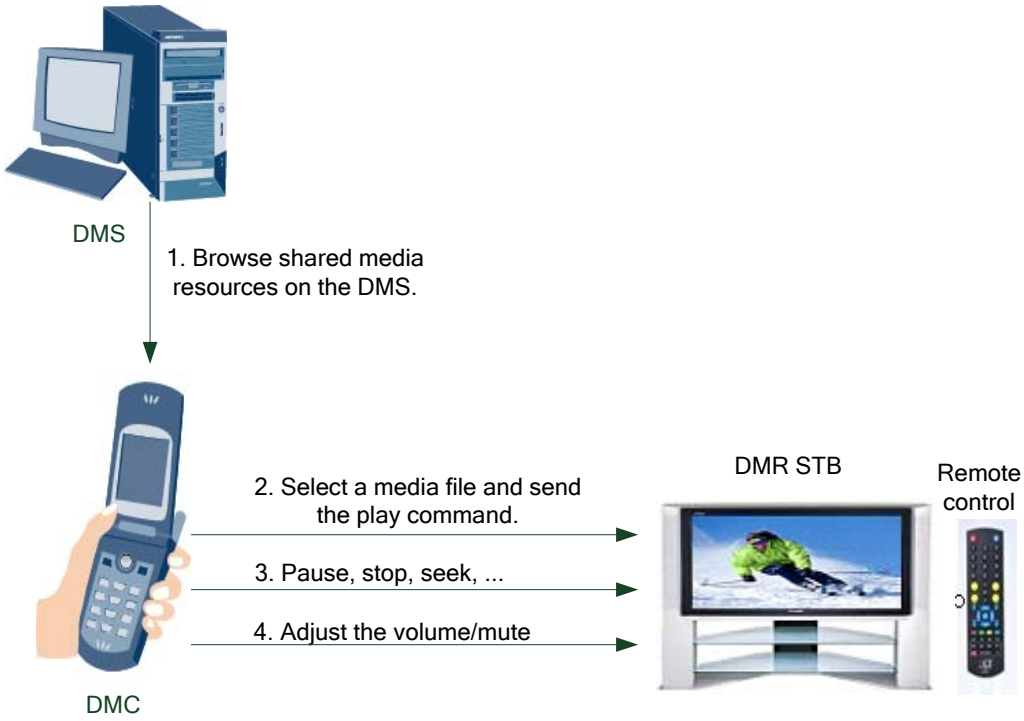
----End

5.3.1.2 Application Scenario of the DMR

Common Application Scenario

Figure 5-1 shows the common application scenario of the DMR.

Figure 5-1 Common application scenario of the DMR



The STB is a DMR, which is controlled by the DMC and responds to the commands from the DMC. For example, in Figure 5-1, the PC is a DMS device, which shares contents. The mobile phone is a DMC device, which browses shared contents on the DMS, views information about shared files, and pushes the files to the STB (DMR) for playing.

During the playback by using an STB, you can also use the remote control to perform related operations, such as exit or pause.

CAUTION

When the playback status of the STB changes, for example, the playback pauses, the DMR needs to notify the DMC and DMS by calling corresponding APIs while the DMP does not need to notify the DMC and DMS.

Process Analysis (sample_dmr.c)

Starting the DMR

This API must be called first. After this API is called, the DMR is started and can be found by clients on PCs or mobile phones (the default name is HiMedia Render). The DMR device name cannot be changed during initialization. You can call HI_DLNA_SetDeviceName to change the DMR device name. After that, the DMR protocol stack restarts.

```
SampleCode_DlnaApiStackInit();
```

Exiting the DMR

To exit the DMR, call the following API:

```
SampleCode_DlnaApiStackDeInit();
```

Changing the DMR Name

The default DMR name is HiMediaRender. After the DMR name is changed, you can view the new DMR name on clients on PCs or mobile phones.

```
HI_DLNA_SetDeviceName(HI_CONST HI_CHAR *pcNewName);
```

Restarting the DMR Protocol Stack When the STB IP Address Changes

If the IP address of the STB changes, call the following API to restart the DMR protocol stack:

```
HI_DLNA_IpChange();
```

Responding to the Playback Control

The DMR receives and responds to commands from the DMC. Start the DMR first. Register the events to which the DMR needs to respond with the protocol stack by referring to the following instance function:

```
SampleCode_DlnaApiRegisterDmrCallback();
```

After the preceding API is called, the DMR can receive the events controlled by the DMC, such as setting the URL, playing, and pausing, and performs corresponding operations according to the events. The following describes all the APIs that are used to respond to the DMC-controlled events.

- Callback_DlnaDmrSetMediaUri

```
HI_U32 Callback_DlnaDmrSetMediaUri
(
    HI_U32    ulInstanceID,
    HI_UCHAR  *pucUri,
    HI_U32     ulUrlLen,
    HI_U32     ulMediaId,
    HI_UCHAR  *pucMetaData,
    HI_U32     ulMetaDataLen,
    HI_VOID   *pvAuxData
)
```

Note the following:

- After the STB receives this event, it sets the media URL and meta-information (for example, the file name, size, and type).
- When the STB receives this event, it only sets the URL but does not start the player. The player is started when the play command is received.
- The parameters are described as follows:

  pucUri: URL of the media file. When the media file is played, the URL is sent to the player.

  ulUrlLen: length of the URL.

  pucMetaData: media meta-information. For example, the media name or supported protocol stack. This parameter can be empty, which depends on the DMS. pucMetaData is not clearly defined by a standard structure. For details about the media information, see the protocol definition.

  ulMetaDataLen: length of the media meta-information.

- Callback_DlnaDmrPlayInd

```
Callback_DlnaDmrPlayInd
(
    HI_U32    ulInstanceID,
    HI_UCHAR  *pucPlaySpeed,  /* string */
    HI_U32    ulStrLen,
    HI_VOID   *pvAuxData
)
```

Note the following:

- When the STB receives this event, it starts the player. The URL of the media to be played is the one configured by calling Callback_DlnaDmrSetMediaUri.
- This event is meaningless if the Callback_DlnaDmrSetMediaUri event is never received. The previously played media can be played again.
- The parameters are described as follows:

  pucPlaySpeed: play speed. The default value is **1**. Other parameters can be ignored.

- Callback_DlnaDmrPauseInd

  When the STB receives this event, it pauses the media playback. If no media is being played, this event is meaningless. The pause operation controlled by the DMC requires only the pause API of the player, and the DLNA is not involved.

- Callback_DlnaDmrStopInd

  When the STB receives this event, it stops the media playback. If the URL still exists after the media playback is stopped, the media with the URL can be played again.

- Callback_DlnaDmrSeekInd

```
HI_U32 Callback_DlnaDmrSeekInd
(
   HI_U32     ulInstanceID,
   HI_DLNA_SEEKMODE_E enSeekMode,
   HI_DLNA_PARATYPE_E enSeekDataType,
   HI_VOID      *pvSeekTarget,
   HI_VOID      *pvAuxData
)
```

Note the following:

- When the STB receives this event, it performs the corresponding seek operation based on the seek type.

- Callback_DlnaDmrGetAllTime and Callback_DlnaDmrGetCurrTime

  These two APIs are used to request the total duration and the current time (the format is 00:00:00) of the media that is being played.

- Callback_DlnaDmrSetVol

```
HI_UCHAR * Callback_DlnaDmrSetVol
(
    HI_S32      ulInstanceID,
    HI_U32      ulDesiredVol,
    HI_CHAR      *pcChannel
)
```

  This API is used to set the volume. The parameter **ulDesiredVol** specifies the volume value, ranging from 0 to 100. When this API is called, the player volume is set to the value of **ulDesiredVol**.

- Callback_DlnaDmrSetMute

```
HI_UCHAR * Callback_DlnaDmrSetMute
(
    HI_S32    ulInstanceID,
    HI_BOOL   bDesiredMute,
    HI_CHAR   *pcChannel
)
```

  This API is used to set the mute attribute. The parameter **bDesiredMute** specifies the mute attribute. When **bDesiredMute** is set to **0**, the mute mode is cancelled; when it is set to **1**, the player is muted.

Controlling the Playback by Using a Remote Control

To control the playback by using a remote control, for example, stop the playback using a remote control, call the following API:

```
HI_DLNA_SetDmrCurrPlayState(HI_DLNA_DMR_ACTION_E dmrState);
typedef enum hiDLNA_DMR_ACTION_E
{
    HI_DLNA_DMR_ACTION_INVALID = -1,
    HI_DLNA_DMR_ACTION_PLAY,
    HI_DLNA_DMR_ACTION_STOP,
    HI_DLNA_DMR_ACTION_PAUSE,
    HI_DLNA_DMR_ACTION_FINISH,
    HI_DLNA_DMR_ACTION_BUTT = HI_ENUM_END
}HI_DLNA_DMR_ACTION_E
```

5.3.1.3 Application Scenario of the DMP

Browse files on a DMS device. Perform the following steps:

Step 1 Initialize the DMP protocol stack by calling the following API:

```
HI_DLNA_InitStack(HI_DLNA_INIT_MODE_DMP);
```

Step 2 Obtain the DMS device list by calling the following API:

```
HI_DLNA_DMP_GetDmsList(ppDmsList);
```

Step 3 Browse the shared directories by calling the following API:

```
HI_DLNA_DMP_BrowseDmsCD(devnum, ObjectID, in_beg, in_num);
```

Step 4 Check whether the search is complete by calling the following API:

```
HI_DLNA_DMP_GetBrowseRspFlag();
```

Step 5 Obtain the lists of directories and media files by calling the following APIs:

```
HI_DLNA_DMP_GetDmsContainList();
HI_DLNA_DMP_GetDmsMediaList();
```

Step 6 Play a specified media file by calling the local playback API after obtaining the URL of the file.

----End

5.3.2 HiDLNA Linux Sample Code

This section describes the installation and usage of the HiDLNA sample code on Linux. The sample files described in this section include **sample_dms.c**, **sample_dmr.c**, **sample_dms_dmr_dmp.c**, and **sample_dmp.c**.

5.3.2.1 Compiling the Sample Code

Decompress the Linux version component package to obtain the **hidlna** directory. Table 5-1 describes the contained directories.

Table 5-1 HiDLNA directories

Directory	Content
include	HiDLNA head files required for compiling samples
lib	HiDLNA library files required for compiling and running samples
Makefile	Makefile for compiling samples
sample	Sample source code file and other related files

In addition, the package contains a **component_common** directory, which stores all public library files required for the HiDLNA compilation.

To compile the HiDLNA sample code in the Linux SDK, perform the following steps:

Step 1 Decompress the SDK package. For example, for Hi3716X V100R001C00SPC0A1, the decompressed SDK directory is **Hi3716XV100R001C00SPC0A1**. If a patch package exists,

integrate the patch package first and run **make build** to compile the SDK package based on the instructions.

Step 2 Copy the **hidlna** directory to the **sample** directory of the SDK. For example, forHiSTBAnroidVXXXRXXXCXXSPC0XX, copy the **hidlna** directory to **HiSTBAnroidVXXXRXXXCXXSPC0XX/sample/**.

Step 3 Copy the header files in **hidlna/include** to the **pub/include** directory of the SDK and copy the dynamic library files in **hidlna/lib/arm-hisiv200-linux-debug** to the **pub/lib/share** directory of the SDK, for example, **HiSTBAnroidVXXXRXXXCXXSPC0XX/pub/include** and **HiSTBAnroidVXXXRXXXCXXSPC0XX/pub/lib/share**.

Step 4 Copy the library file **libcomponent_common.so** in the **component_common** directory to **pub/lib/share** of the SDK.

Step 5 Modify the **Makefile** file in the **sample** directory of the SDK, for example, for**HiSTBAnroidVXXXRXXXCXXSPC0XX**, modify **HiSTBAnroidVXXXRXXXCXXSPC0XX/sample/Makefile**. Add **make –C hidlna** under **all:**.

Step 6 Run **make sample** in the SDK root directory to compile the HiDLNA sample.

----End

5.3.2.2 Configurations Before Running Sample Code

To run the HiDLNA sample code on the development board, perform the following steps:

Step 1 Burn the running environment of the Linux SDK to the board and enter the shell command line interface.

Step 2 Configure the IP address for the board. The Linux system is not connected to the network by default, you need to configure an IP address for the system by running the following command:

```
ifconfig eth0 xxx.xxx.xxx.xxx
```

In addition, obtain lo (127.0.0.1), which is required by sample_dms, by running the following command:

```
ifconfig lo 127.0.0.1
```

Step 3 Copy the dynamic libraries that the sample code depends on in the **hidlna/lib/arm-hisiv200-linux-debug** directory to the /**usr/lib** directory, or add the dependent dynamic libraries (**hidlna/lib/arm-hisiv200-linux-debug**)to LD_LIBRARY_PATH by using NFS mounting. For example, if the shared library directory after the NFS mounting is **/mnt/rootfs_full/dlnalib**, run the following command:

```
export LD_LIBRARY_PATH="/mnt/rootfs_full/dlnalib:$LD_LIBRARY_PATH"
```

Copy the library file **libcomponent_common.so** in the **component_common** directory to **usr/lib/** of the SDK, or share the library file by using NFS mounting. The sample file of the HiDLNA depends on this library.

Step 4 Transfer the sample file to any directory on the development board and grant the executable permission on the file.

```
chmod u+x sample_dms sample_dmr sample_dmp sample_dms_dmr_dmp
```

----End

5.3.2.3 Testing the Sample DMS

Add the **data** and **uploaddir** directories to the directory when the executable file **sample_dms** is located, and put media files to be shared to the **data** directory. Run **sample_dms**, and the following information is displayed:

```
# ./sample_dms
*****************************
* DMS Help Info *
*****************************
a:dms init
b:change dms friendly name
c:add "./data" as shared directory
h:add shared directory by hand, such as an Absolute Dir Path from SD card
u:set "./uploaddir" as new upload directory
p:set "/usr/share/dlna/dlnaUploadDir as upload directory
r:remove "./data"
q:dms deinit
>>
```

The information is described as follows:

- a

  Initializes the DMS protocol stack.
- b

  Sets the DMS name. In the example, the DMS name is set to **This is a Test**.
- c

  Shares the **data** directory under the current directory.
- h

  Enables the user to specify a directory to be shared. For example, if you input **/mnt**, the DMS shares the contents in the **/mnt** directory to those with the access permission.
- u

  Sets the **/uploaddir** directory under the current directory as the uploading directory. This setting takes effect after the DMS restarts.
- p

  Restores the default uploading directory. This setting takes effect after the DMS restarts.
- r

  Removes the **data** directory from the shared directory list.
- q

  Exits the DMS.

To test sample_dms, perform the following steps:

Step 1 Test the following functions by running the **a**, **b**, and **c** commands in sequence: initialize the DMS, change the DMS name, and add shared directories.

The DMS device name (This is a Test) can be viewed from the DMP or DMC device.

Step 2 Run the **h** command to manually add a directory as a shared directory. The system displays the following information:

```
Please input the directory path or "quit" to back:
```

Enter the directory to be shared, for example, **/mnt**, and press **Enter**.

Step 3 Run the **r** command. The **data** directory is removed from the shared directory list, but other shared directories added by running the **h** command are not removed.

Step 4 Run the **u** command. The uploading directory is changed to the **uploaddir** directory under the current directory and the setting takes effect after sample_dms restarts.

Step 5 Run the **p** command. The uploading directory is restored to the default uploading directory set by the protocol stack (**/usr/share/dlna/dlnaUploadDir**)and the setting takes effect after sample_dms restarts.

You can also modify the configuration file **/usr/share/dlna/dlna.ini** to change the uploading directory.

Step 6 Run the **q** command to exit sample_dms. Resources occupied by the protocol stack are released.

----End

5.3.2.4 Testing the Sample DMR

Run **sample_dmr**, and the following information is displayed:

```
# ./sample_dmr
*****************************
* DMR Help Info *
*****************************
a:dmr init
b:set dmr friendly name
c:set dmr  state
d:dmr ip change
q:dmr deinit
```

The information is described as follows:

- a
  Initializes the DMR. Starts a DMR named HiMediaRender.
- b
  Sets the DMR name, for example, **This is a Test**.
- c
  Sets the DMR state.
- d
  Restarts the DMR if the IP address changes.
- q
  Exits the DMR.

After the DMR is started, you can test only some of the parameters and display the result through the print information.

You can use the UPnP certification test tool to perform separate tests. Some functions cannot be tested and some test cases cannot be performed.

5.3.2.5 Testing the Sample DMP

Run **sample_dmp**, and the following information is displayed:

```
# ./sample_dmp
*****************************
* DMP Help Info *
a:dmp start
b:Get dms list
c:browse directory
d:exit the device
q:quit
```

The information is described as follows:

- a

  Starts the DMP.
- b

  Obtains the DMS device list.
- c

  Browses the shared directories. You can select a device from the obtained DMS list, enter **0** to browse the contents of the top directory on the device, and run the **c** command to browse the directories or media files based on the IDs.
- d

  Exits the DMS device. After exiting the DMS device, you can select another DMS device and browse the contents.
- q

  Exits the DMP.

To test sample_dmp, perform the following steps:

Step 1 Run the **a** command to start the DMP.

Step 2 Run the **b** command. The DMS device list is displayed. Record the ID of the device to be browsed.

```
DMS:
   1 -- uuid:bb5e21ce-2222-11b2-f918-8E2D1302F7BE
FriendlyName HiMediaServer(jiang)
   2 -- uuid:55076f6e-6b79-4d65-6469-0022a10453e2
FriendlyName TwonkyServer [TestServer_NAS]
   3 -- uuid:0751b700-b25a-4998-9e98-50838396b62e
FriendlyName W00215319-PC: w00215319:
   4 -- uuid:624e5ba7-5871-4fd3-8e15-64eff64a3f71
FriendlyName TIANJIA-HP: t00204177:
```

```
   5 -- uuid:818b429c-9fc6-4e66-994b-3e6337550918
FriendlyName Z002096281A: z00209628:
   6 -- uuid:832a616e-cc8a-43c4-8122-e3f4c7170135
FriendlyName J00209609-VM1A: j00209609:
```

Step 3 Run the **c** command. The system asks you to enter the device ID and directory ID. If you enter a device for the first time, the ID of the top directory is 0, and you can obtain the IDs of other directories according to the displayed information.

```
please choose one dms device: 1
Input dir id: 0
Input  id:0
DlnaParseActionResp:u:BrowseResponse
<OBJID>0$2</OBJID><NAME>Photos</NAME>
<OBJID>0$3</OBJID><NAME>Videos</NAME>
Browse Result
TotalMatches:2
NumberReturned:2
RealNumReturned:2
```

Step 4 Run the **d** command to exit the DMS device. Then you can select another DMS device.

Step 5 Run the **q** command to exit the DMP.

----End

5.3.2.6 Testing the Sample DMS_DMR_DMP

Run **sample_dms_dmr_dmp**, and the following information is displayed:

```
/*****************************/
a:dms_dmr_dmp start
q:dms_dmr_dmp quit
/** DMP Help Info **/
b:Get dms list
c:browse directory
d:exit the device
/** DMR Help Info **/
h:set dmr&dms friendly name
i:set dmr  state
j:dmr ip change
/** DMS Help Info **/
r:add "./data" as shared directory
s:add shared directory by hand, such as an Absolute Dir Path from SD card
t:set "./uploaddir" as new upload directory
u:set "/usr/share/dlna/dlnaUploadDir as upload directory
v:remove "./data"
/*****************************/
```

sample_dms_dmr_dmp starts the DMP, DMR, and DMS services in the same process, and sample_dmp, sample_dmr, and sample_dms each starts only one service in a process. The testing method of sample_dms_dmr_dmp is similar to that of sample_dmp, sample_dmr, or sample_dms. Note the following:

- a: dms_dmr_dmp start

  Starts the DMS, DMR, and DMP services simultaneously.
- q: dms_dmr_dmp quit

  Stops the DMS, DMR, and DMP services simultaneously.
- h: set dmr&dms friendly name

  Changes the DMR and DMS device names simultaneously.

5.4 Solutions to Common Development Issues

5.4.1 Android

5.4.1.1 How Do I Modify the DMR or DMS Device Description?

Problem Description

How do I modify the device information?

Cause Analysis

The displayed device manufacturer information of the DMR and DMS is Hisilicon Technologies Co.,Ltd on the DMC end. If you need to customize this kind of information, do as follows.

Solutions

Modify the character strings **dmrdescription** and **dmsdescription** in **HiDLNASettings/res/values/strings.xml** as follows:

```
<manufacturer>Hisilicon Technologies Co.,Ltd</manufacturer>
       <manufacturerURL>http://www.Hisilicon.com</manufacturerURL>
       <modelName>Hisilicon MediaXXXX</modelName>
       <modelNumber>1.1</modelNumber>
       <modelURL>http://www.Hisilicon.com</modelURL>
```

After modification, you need to recompile the HiDLNA settings and update **HiDLNASettings.apk**. The modification takes effect after the system is restarted.

5.4.1.2 How Do I Locate a DLNA Push Error?

Problem Description

When an exception occurs during the DLNA push, how do I locate the issue?

Cause Analysis

For the preliminary issues, you can analyze the problems based on the proc and log information. For the major problems, you can diagnose the issues through packet-capture analysis by using the tcpdump.

Solutions

[Solution 1] Locate the issues by using the DLNA proc.

Before using this function, ensure that the DMR push has been completed. You can obtain the following proc information:

- URI: Run **cat /proc/hisi/hidlna/URI** over the serial port.

  The URL information similar to that shown in Figure 5-2 is displayed. This URL is the URI information carried by the **SetAVTransportURI** information by default. You can open this URI using a browser or using the video LAN client (VLC) to check the validity of the resource.

Figure 5-2 URI displayed in the proc

```
root@Hi3751AV500:/ # cat /proc/hisi/hidlna/URI
############################URI#########################
http://10.161.179.59:10243/WMPNSSv4/1372349108/ezlDRTU2NTMwLUEzM0EtNDdERC04MEI0LUI4ODgzMEJDM0E0M30uMC44.mpg?formatID=28
```

- URIMetaData: Run **cat /proc/hisi/hidlna/URIMetaData** over the serial port.

  The MetaData information similar to that shown in Figure 5-3 is displayed. This MetaData information is carried by **SetAVTransportURI** by default. You can check the push type and whether the seek operation is supported from this MetaData information.

Figure 5-3 URIMetaData displayed in the proc

```
root@Hi3751AV500:/ # cat /proc/hisi/hidlna/URIMetaData
########################URIMetaData###################
<DIDL-Lite xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/" xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/" xm
PNSS-1-0/"><item id="{9CE56530-A33A-47DD-80B4-B88830BC3A43}.0.8" parentID="8" restricted="1"><upnp:albumArtURI dlna:pro
C04MEI0LUI4ODgzMEJDM0E0M30uMC44.jpg?albumArt=true</upnp:albumArtURI><upnp:albumArtURI dlna:profileID="JPEG_TN">http://
MC44.jpg?albumArt=true,formatID=13</upnp:albumArtURI><upnp:albumArtURI dlna:profileID="JPEG_SM">http://[fe80::c0ee:5564
0M30uMC44.jpg?albumArt=true</upnp:albumArtURI><upnp:albumArtURI dlna:profileID="JPEG_TN">http://[fe80::c0ee:5564:49a8:e
.jpg?albumArt=true,formatID=13</upnp:albumArtURI><upnp:albumArtURI dlna:profileID="JPEG_SM">http://10.161.179.59:10243/
mArt=true</upnp:albumArtURI><upnp:albumArtURI dlna:profileID="JPEG_TN">http://10.161.179.59:10243/WMPNSSv4/1372349108/e
upnp:albumArtURI><upnp:albumArtURI dlna:profileID="JPEG_SM">http://[fe80::dd28:ba3e:7437:97a3]:10243/WMPNSSv4/137234910
bumArtURI><upnp:albumArtURI dlna:profileID="JPEG_TN">http://[fe80::dd28:ba3e:7437:97a3]:10243/WMPNSSv4/1372349108/ezlDR
:albumArtURI><upnp:albumArtURI dlna:profileID="JPEG_SM">http://10.141.136.44:10243/WMPNSSv4/1372349108/0_ezlDRTU2NTMwLU
bumArtURI dlna:profileID="JPEG_TN">http://10.141.136.44:10243/WMPNSSv4/1372349108/ezlDRTU2NTMwLUEzM0EtNDdERC04MEI0LUI4O
lna:profileID="JPEG_SM">http://[::1]:10243/WMPNSSv4/1372349108/0_ezlDRTU2NTMwLUEzM0EtNDdERC04MEI0LUI4ODgzMEJDM0E0M30uMC
```

- Action: Run **cat /proc/hisi/hidlna/Action** over the serial port.

  A simple log of recent 10 pushes is displayed as shown in Figure 5-4. This information is used to check the playback status. When a playback error occurs, the log information helps to identify the operations that cause the error.

Figure 5-4 Action displayed in the proc

```
root@Hi3751AV500:/ # cat /proc/hisi/hidlna/Action
##########################DMC Action######################
00:02:30.479,Seek:00:01:10
00:02:30.631,Play:1
00:02:32.394,Pause
00:02:33.205,Seek:00:00:17
00:02:33.522,Seek:00:00:15
00:02:33.720,Play:1
00:02:40.310,Stop
18:29:30.906,GetCurrentTransportActions
18:29:53.349,SetTransportURI
18:29:54.880,Play:1
```

- PlayerTimer: Run **cat /proc/hisi/hidlna/PlayerTimer** over the serial port.
  A log showing the time of calling the MediaPlayer is displayed as shown in Figure 5-5.

Figure 5-5 PlayerTimer displayed in the proc

```
root@Hi3751AV500:/ # cat /proc/hisi/hidlna/PlayerTimer
##########################HiPlayer Time####################
#old:
00:01:56.917,playurl:http://10.161.179.101:51121/74366056-0cb7-402e-9e3a-f32d94207a68.mp4
00:02:30.364,before seek
00:02:30.488,after seek
00:02:30.491,before seek
00:02:30.671,after seek
00:02:33.230,before seek
00:02:33.599,after seek
00:02:33.604,before seek
00:02:33.853,after seek
18:29:55.481,before reset
18:29:55.482,after reset
#new:
18:29:55.486,playurl:http://127.0.0.1:10243/WMPNSSv4/1372349108/0_ezlDRTU2NTMwLUEzM0EtNDdERC04MEI0LUI4ODgzMEJDM0E0M30uMC44.mp4
,
```

[Solution 2] Locate the issues by using the DLNA log.

The DLNA log system adopts the standard Android Logcat protocol, and displays the key procedures of the protocol and the received network commands. You can capture logs by running the standard Logcat command over the serial port or by connecting to the Android Debug Bridge (ADB) at the computer end.

[Solution 3] Locate the issue by capturing network packets.

The board provides the network packet capturing commands. You can locate the compatibility issues by capturing network packets.

For example, if the DLNA works on eth0, run the following command over the serial port:

```
tcpdump -p -vv -s 0 -i eth0 -w /sdcard/1.pcap
```

You can capture the status files of the network packets when the issue happens, and provide the files to HiSilicon R & D engineers for analysis.

5.4.1.3 How Do I Determine Whether the DLNA Service is Started Normally?

Problem Description

After the board is started, how do I determine whether the DLNA service is started normally?

Cause Analysis

View the log to check the information about the startup of the UPnP protocol stack, where you can check whether the DLNA service is started normally.

Solution

Step 1 Run **ps | grep dlna** over the serial port to query the DLNA process IDs.

Figure 5-6 Querying the DLNA process IDs

```
130|root@Hi3798MV100:/ # ps | grep dlna
system    2194  1364  539756 20336 ffffffff 2b1387f4 S com.hisilicon.dlna.dms
system    2207  1364  534664 20268 ffffffff 2b1387f4 S com.hisilicon.dlna.dmr
```

Step 2 Run **logcat –v time | grep PID** over the serial port based on the DLNA process IDs.

In Figure 5-7, the process ID (PID) is **2207**. View the displayed information to check whether it contains the information in the red box as shown in Figure 5-7. If yes, the protocol stack is started properly.

Figure 5-7 Checking the DLNA log

```
root@Hi3798MV100:/ # logcat -v time | grep 2207
01-01 00:00:19.371 I/ActivityManager( 1866): Start proc com.hisilicon.dlna.dmr for added application com.hisilicon.dlna.dmr: pid=2207 uid=1000 gids={41000, 1028,
01-01 00:00:34.089 D/DMRNative( 2207): getInstance
01-01 00:00:34.089 D/DMRNative( 2207): getInstance
01-01 00:00:34.097 E/UpnpBootBroadcastServiceDMR( 2207): Save description
01-01 00:00:34.107 D/UpnpBootBroadcastServiceDMR( 2207): UpnpBootBroadcastService onStartCommand : null
01-01 00:00:34.148 D/UpnpBootBroadcastServiceDMR( 2207): readConfig
01-01 00:00:34.148 D/UPNP     ( 2207): com.hisilicon.dlna.dmr.UpnpBootBroadcastServiceDMR.readConfig()  Line:426  (UpnpBootBroadcastServiceDMR.java)
01-01 00:00:34.161 D/UPNP     ( 2207): com.hisilicon.dlna.dmr.UpnpBootBroadcastServiceDMR.pirntConfig()  Line:481  (UpnpBootBroadcastServiceDMR.java)
01-01 00:00:34.161 D/UpnpBootBroadcastServiceDMR( 2207): hostNameValue=Living Room 1_7293
01-01 00:00:34.161 D/UpnpBootBroadcastServiceDMR( 2207): dmrStartFlag=1
01-01 00:00:34.161 D/UpnpBootBroadcastServiceDMR( 2207): dmsStartFlag=1
01-01 00:00:34.161 D/UpnpBootBroadcastServiceDMR( 2207): shareDirecoryValue=null
01-01 00:00:34.161 D/UpnpBootBroadcastServiceDMR( 2207): dmsNameValue=HisiMediaServer
01-01 00:00:34.163 D/UpnpBootBroadcastServiceDMR( 2207): dmrNameValue=HiMediaRender
01-01 00:00:34.164 D/UPNP     ( 2207): com.hisilicon.dlna.dmr.UpnpBootBroadcastServiceDMR.onStartCommand()  Line:216  (UpnpBootBroadcastServiceDMR.java)
01-01 00:00:34.165 E/UpnpBootBroadcastServiceDMR( 2207): will start dmr
01-01 00:00:34.175 D/UpnpBootBroadcastServiceDMR( 2207): sleep 1000 for net check
01-01 00:00:34.822 D/UPNP     ( 2207): com.hisilicon.dlna.dmr.UpnpBootBroadcastServiceDMR$NetStateChangeReceiver.onReceive()  Line:321  (CONNECTIVITY_CHANGE,IFI_A
_CHANGE)
01-01 00:00:34.834 D/UPNP     ( 2207): com.hisilicon.dlna.dmr.UpnpBootBroadcastServiceDMR$NetStateChangeReceiver.onReceive()  Line:350  (netOldState set to off)
01-01 00:00:35.176 E/UpnpBootBroadcastServiceDMR( 2207): timerExpire ==1
01-01 00:00:35.182 D/UpnpBootBroadcastServiceDMR( 2207): sleep 1000 for net check
01-01 00:00:36.183 E/UpnpBootBroadcastServiceDMR( 2207): timerExpire ==3
01-01 00:00:36.189 D/UpnpBootBroadcastServiceDMR( 2207): sleep 1000 for net check
01-01 00:00:37.189 E/UpnpBootBroadcastServiceDMR( 2207): timerExpire ==5
01-01 00:00:37.196 D/UpnpBootBroadcastServiceDMR( 2207): sleep 1000 for net check
01-01 00:00:38.196 E/UpnpBootBroadcastServiceDMR( 2207): timerExpire ==7
01-01 00:00:38.208 D/UpnpBootBroadcastServiceDMR( 2207): sleep 1000 for net check
01-01 00:00:39.209 E/UpnpBootBroadcastServiceDMR( 2207): timerExpire ==9
01-01 00:00:39.217 D/UpnpBootBroadcastServiceDMR( 2207): sleep 1000 for net check
01-01 00:00:40.217 E/UpnpBootBroadcastServiceDMR( 2207): timerExpire ==11
01-01 00:00:40.222 D/DMRBinder( 2207): create()
01-01 00:00:40.222 D/DMRManager( 2207): create
01-01 00:00:40.227 I/CommonDef( 2207): [getCurrentAdapterName] wifi is open
01-01 00:00:40.228 D/CommonDef( 2207): getCurrentAdapterName start strAdapterName=wlan0
01-01 00:00:40.239 E/VPP_DLNA_LOG( 2207): [VPPDLNA][UpnpInit3][:510] === the name is adapter_name: wlan0
01-01 00:00:40.239 E/VPP_DLNA_LOG( 2207): [VPPDLNA]:[getlocalhostname]:the param adapterName is wlan0!
01-01 00:00:40.240 E/VPP_DLNA_LOG( 2207): [VPPDLNA]:[getlocalhostname] find the adapter :wlan0,MAC is 2059A0BDB71A
01-01 00:00:40.240 E/VPP_DLNA_LOG( 2207): [VPPDLNA]:Inside getlocalhostname: after strncpy 10.161.179.19
```

----End

5.4.1.4 What Do I Do If the DLNA Device Cannot Be Detected Sometimes?

Problem Description

The DLNA device has been started properly, but the DLNA client on the mobile phone cannot detect the DLAN device sometimes.

Cause Analysis

The DLNA client on the mobile phone can detect the DLNA device only after the DLNA client completes the simple service discovery protocol (SSDP) interaction with the DLNA device and downloads and parses the device and service description files properly.

Solution

The device detection process involves the distribution of SSDP messages, and the downloading and parsing of device and service information. The distribution of SSDP messages adopts the User Datagram Protocol (DUP), which may cause packet loss. However, the HiSilicon service end has optimized the service by increasing the transmission frequencies of SSDP alive messages. Therefore, there is less occurrence of device detection failure due to UDP packet loss.

The most possible cause is that the unstable network service causes the client to fail to download the device and service description files, which results in the device detection failure. You can check the network status by using the ping tool and the Wi-Fi analyzer. You are advised to use the wired network or 5G network to rule out network congestion and adjacent-frequency interference.

5.4.1.5 What Do I Do If the DLNA Device Cannot Be Detected?

Problem Description

The DLNA service has been started properly, but the client cannot detect the DLNA device.

Cause Analysis

The HiSilicon DLNA supports working on only one network adapter. As described in section 5.4.1.4 " What Do I Do If the DLNA Device Cannot Be Detected Sometimes?", the device can be detected only after the proper interaction between the DLNA service end and the client end.

Solution

Step 1 Ensure that the client end and the service end are in the same LAN. Connect the STB and mobile phone to the same LAN if they are in different LANs.

Step 2 Check whether other DLNA devices can be detected on the mobile phone client.

- If all DLNA devices cannot be detected, the mobile phone client is faulty. You are advised to change the DLNA client.
- If only the HiSilicon DLNA device cannot be detected, view the log information and capture network packets for further analysis.

5.4.1.6 What Do I Do If the Mobile Phone Client Fails to Push Messages?

Problem Description

After a push message is sent from the mobile phone client, there is no response from the DLNA server or the playback fails.

Cause Analysis

The possible causes of this problem are classified as follows:

- Network and client problem: The DLNA server end fails to receive the message from the client.
- Client problem: The pushed resource is invalid, which causes the playback failure.
- Media playback failure or Android platform compatibility issue.

Solution

- The DLNA server fails to receive the message from the client.

  The DMR can implement the playing operation only after receiving the DMC message. When the network environment is poor, the third-party DLNA client performs poorly, or the HiSilicon DLNA server runs abnormally, the DMR may fail to receive the action message.

  You can check the recent messages received by the DMR in the proc information by running the following command over the serial port:

  ```
  cat /proc/hisi/hidlna/Action
  ```

  Check the displayed proc information as shown in Figure 5-4. If it is a push operation, check the following:

  - Whether the **SetTransportURI** and the play messages are received.
  - If the messages are not received, check the network connection and whether the client end is connected to the specified DMR device.

- The resource is invalid.

  Run the following command over the serial port:

  ```
  cat /proc/hisi/hidlna/URI
  ```

  The URL information similar to that shown in Figure 5-2 is displayed. This URL is the URI information carried by the **SetAVTransportURI** information by default.

  - If the pushed resource is an image, open this URL in a browser.
  - If the resource is an audio or video file, open this URL using the VLC on the computer to check whether the playback is normal. If the playback fails, the resource may not exist, and the problem comes from the third-party DMS or mobile phone client.

    If the playback of the picture and audio/video is normal, check the local playback on the Android platform.

- Media playback fails or the Android platform is not compatible.
  - If the picture is displayed normally in a browser but fails to be pushed, download the picture in the browser to the **sdcard** directory on the board. Play the picture using the local Android picture player to check whether the playback is normal. If the Android local playback fails too, the picture format is not supported on the Android platform.

If the playback succeeds, an error may occur in DLNA downloading decoding. Capture the log for further analysis.

- If the audio/video is played normally on the VLC but fails to be pushed, check the playback status by using the local video player. Run the following command over the serial port:

  ```
  am start -n
  com.hisilicon.android.videoplayer/.activity.VideoActivity URL
  ```

  The URL in the command is the displayed URL of **cat /proc/hisi/hidlna/URI** corresponding to the push.

  - If the playback by the video player fails, the issue may be caused by the media.
  - If the playback succeeds, capture the log for DLNA playback failure analysis.

5.4.1.7 What Do I Do If the Pushed Resources are Loaded Slowly or Intermittently on the Mobile Phone?

Problem Description

When the pictures and videos taken by a mobile phone are pushed to the board for playback, intermittence occurs or the picture is loaded slowly.

Cause Analysis

The resolution of most of the pictures taken by a mobile phone is 1080p or above, and the resolution of the videos is 720p or above. To play 720p videos smoothly on a mobile phone, the bandwidth should be broader than 1 Mbit/s. When the network status is unstable, typically the media downloading rate of the DMR from the DMS HTTP is less than 1 Mbit/s. Therefore, intermittence occurs during video playback and pictures are loaded slowly.

Solution

You are advised to rule out the network interference by using a stable network environment. For example, connect the board to the wired network, or connect both the board and the mobile phone to a stable wireless network. Switch the board and the mobile phone to the 5G frequency band. Analyze the used network using the Wi-Fi analyzer to avoid adjacent-frequency interference.

5.4.1.8 What Do I Do for the Splash Screen?

Problem Description

After the DMC client pushes a resource, it continues to push the next resource without exiting the current playback status. The playback page continues to play immediately after existing the current playback.

Cause Analysis

In the DLNA protocol, when the client stops playing, it sends a stop message to the service end. When the client pushes another resource while a pushed resource is being played, it also sends a stop message between the two pushes. The purpose of this stop message is to reset the DMR status machine, not to exit from the playing status. In the two cases, the stop messages sent from the client are the same, and the service end is unable to identify whether the stop message is used to stop playing or to reset the status.

- If the message is to stop playing, the DLNA should exit from the playback status.
- If the message is not to stop playing, the DLNA should wait for the next push and then continue with the playback. Otherwise, the DLNA playback will exit unexpectedly and then restart, which causes the screen flicker. The screen flicker affects the push performance and user experience.

Solution

To cope with the screen flicker issue, the HiSilicon DLNA delays three seconds before exiting the playback status when it receives a stop message. If another push is received during the delay time, the exit process is terminated and the playback continues. The 3-second delay prolongs the exit time. However, compared with the enhanced push performance and user experience, the cost is controllable.

5.4.1.9 How Do I Process the Special Escape Characters in the URL?

Problem Description

The resources pushed to the DMR using Skifta fails to be played. What should I do to solve this problem?

Cause Analysis

The URI in the pushed resource contains the characters "%20". To solve the previous compatibility issues, the characters "%20" are treated as escape characters (converted to spacing). In Skifta, the characters "%20" refer to the character string "%20".

Solution

Do not convert "%20" to spacing. The playback of the resources pushed by the Skifta client succeeds. However, this may cause push failures of other applications that consider "%20" as escape characters. There are many similar issues for the DLNA. A compromising solution is recommended to support the most frequently used clients.

5.4.1.10 How Do I Set the Wired/Wireless Network Bounding Strategies?

Problem Description

The DLNA device in the wireless network cannot be found when the wired network is set to be the first option for system network access.

Cause Analysis

The HiSilicon DLNA works only on one network adapter and follows the system network priority by default. Therefore, the DLNA devices in other networks cannot be found in multiple network environment.

Solution

Modify the network priority in the open-source codes of the upper-layer application. The DMS/DMR/DMP are modified separately in different files and the files to be modified are as follows:

- DMS:
  android/packages/apps/HiMediaShare/src/com/hisilicon/dlna/dms/MediaService.java
- DMR:
  android/packages/apps/HiMediaRender/src/com/hisilicon/dlna/dmr/UpnpBootBroadcastServiceDMR.java
- DMP:

  android/packages/apps/HiMediaCenter/src/com/hisilicon/dlna/mediacenter/MediaCenterActivity.java

  android/packages/apps/HiMediaPlayer/src/com/hisilicon/dlna/player/MainPlayerActivity.java

Modify the following command in corresponding files:

private final static String DEFAULT_USE_ADAPTER_WIFI_ETHERNET = "eth";

//wlan

eth indicates that the wired network is the first option for network access and **wlan** indicates that the wireless network is the first option for network access.

5.4.2 Linux

5.4.2.1 How Does the DMR Handle the Seek Event During Callback?

Problem Description

How does the DMR control the seek operation of the DMR? How is the seek-by-time function implemented?

Cause Analysis

When the DMC controls the playback progress, it requires the DMR to respond by calling the callback function so that the progress of the DMR player is the same as the DMC. The parameters sent from the DMC control how to respond.

Solutions

The seek callback function is defined as follows:

```
typedef HI_U32 (*HI_DLNA_DMR_SEEK_IND_PTR)
(
    HI_U32        ulInstanceID,
    HI_DLNA_SEEKMODE_E enSeekMode,
    HI_DLNA_PARATYPE_E enSeekDataType,
    HI_VOID      *pvSeekTarget,
    HI_VOID      *pvAuxData
)
```

The DMR specifies the seek type and parameters. The supported seek operations include HI_DLNA_SEEK_MODE_ABS_TIME and HI_DLNA_SEEK_MODE_REL_TIME, and the play list is not supported currently.

enSeekDataType is defined as follows:

```
typedef enum hiDLNA_PARATYPE_E
{
    HI_DLNA_PARA_TYPE_INVALID = -1,
    HI_DLNA_PARA_TYPE_STRING =0,
    HI_DLNA_PARA_TYPE_INT32,
    HI_DLNA_PARA_TYPE_UINT32,
    HI_DLNA_PARA_TYPE_FLOAT,
    HI_DLNA_PARA_TYPE_BUTT = HI_ENUM_END
}HI_DLNA_PARATYPE_E
```

When **enSeekMode** specifies the seek type, **enSeekDataType** specifies the data type corresponding to the seek type specified by **enSeekMode**. For details about the definition of data type specified by **enSeekDataType**, see the DlnaParaTypeEn structure.

For HI_DLNA_SEEK_MODE_ABS_TIME and HI_DLNA_SEEK_MODE_REL_TIME, the received **enSeekDataType** is **0** (HI_DLNA_PARA_TYPE_STRING).

See the following:

```
typedef union _DmrSeekTargetValUn
{
       HI_DLNA_STRING_S stStrVal;
       HI_U32 ulVal;
       HI_S32  iVal;
       HI_FLOAT  fVal;
}DmrSeekTargetValUn;


HI_U32 Callback_DlnaDmrSeekInd
(
    HI_U32     ulInstanceID,
    HI_DLNA_SEEKMODE_E enSeekMode,
    HI_DLNA_PARATYPE_E enSeekDataType,
    HI_VOID      *pvSeekTarget,
    HI_VOID      *pvAuxData
)
{
    Dlna_Print("Received the Seek Indication/n");
    int len=0;
    HI_S32 s32Ret = 0;
    DmrSeekTargetValUn uSeekTargetVal;
    HI_U32 mseconds;
    HI_U32 hh,mm,ss;
    if(!pvSeekTarget)
    {
        printf("no seek data/n");
        return -1;
```

```
    }
    memset(&uSeekTargetVal,0,sizeof(DmrSeekTargetValUn));
    memcpy(&uSeekTargetVal,(DmrSeekTargetValUn
*)pvSeekTarget,sizeof((DmrSeekTargetValUn *)pvSeekTarget));
    printf("Seek mode:%d,dataType:%d/n",enSeekMode,enSeekDataType);
    switch(enSeekDataType)
    {
        case HI_DLNA_PARA_TYPE_STRING:
            printf("Seek
value:%s/n",(HI_DLNA_STRING_S)uSeekTargetVal.stStrVal);
            sscanf(uSeekTargetVal.stStrVal.pucBuf,"%d:%d:%d",&hh,&mm,&ss);
            mseconds = (hh * 3600 + mm * 60 + ss) * 1000;
            printf("second:%d/n",mseconds);
            s32Ret = HI_SVR_PLAYER_Seek(hPlayer, mseconds);
            if (HI_SUCCESS != s32Ret)
                printf("/e[31m ERR: seek fail, ret = 0x%x /e[0m /n", s32Ret);
Dlna_Print("ERR: seek status is : %d/n",stPlayerInfo.eStatus);
     break;
        case HI_DLNA_PARA_TYPE_INT32:
            printf("Seek value:%d/n",uSeekTargetVal.iVal);
            //to do:
            break;
        case HI_DLNA_PARA_TYPE_UINT32:
            printf("Seek value:%u/n",uSeekTargetVal.ulVal);
            break;
        case HI_DLNA_PARA_TYPE_FLOAT:
            printf("Seek value:%f/n",uSeekTargetVal.fVal);
            break;
        default:
            printf("no SeekDataType/n");
            return -1;
    }
    //HI_SVR_PLAYER_Seek(hPlayer, u32SeekTime);
    return HI_DLNA_RET_SUCCESS;
}
```

5.4.2.2 How Does the DMP Returns to the Upper-Level Directory?

Problem Description

How does the DMP returns to the upper-level directory when it obtains the directory and file lists of a device?

Cause Analysis

The DMP browses the shared contents on the DMS based on directory IDs. Therefore, you have to record the directory IDs.

Solutions

Take the following directory structure as an example. Assume that the ID of the **video** directory is 1, the IDs of the two sub directories **all video** and **by folder** are 2 and 3 respectively.

```
- video
    ├-- all video
    └-- by folder
```

To enter the video directory, enter the ID **1**. The **all video** and **by folder** directories are displayed. To enter the all video directory, enter the ID **2**. The contents in the all video directory is returned.

If you want to return to the **video** directory, enter the ID **1**. The **all video** and **by folder** directories are displayed if other parameters are all correctly configured. See the following information:

```
HI_S32 HI_DLNA_DMP_BrowseDmsCD
(
    HI_S32  devnum,
    HI_CHAR *ObjectID,
    HI_S16  in_beg,
    HI_S16  in_num
);
```

5.4.2.3 How Do I Use the HI_DLNA_DEVICE_NODE_S Linked List?

Problem Description

When the DMP searches for DMS devices by calling HI_DLNA_DMP_GetDmsList, the HI_DLNA_DEVICE_NODE_S linked list is returned. Is the linked list required to be released immediately after it is obtained or when the user exits the DMP? Is it released when HI_DLNA_DMP_DeleteList () is called? It is tested that when a DMS device is added, its information can be obtained from the linked list; but when a DMS device exits, its information still can be found in the linked list.

Solutions

The linked list is not necessarily released immediately after it is obtained, and whether it is released depends on the DMP.

HI_DLNA_DMP_DeleteList() does not release the linked list.

When a DMS exits, the DMP receives the BYEBYE information from the DMS device and deletes the DMS information. However, you must call DlnaDmcGetDmsList() to refresh the linked list after the deletion.

5.4.2.4 How Do I Specify the Directory for Storing Logs of the HiDLNA Protocol Stack?

Problem Description

The log files for the HiDLNA protocol stack are stored in the current directory by default. Error logs are stored in **DlnaLogError.txt**, and Info logs are stored in **lnaLogInfo.txt**. How do I output the logs to other files, for example, to **/tmp/dlnalog.txt**?

Solutions

You can specify the directory for outputting logs of the HiDLNA protocol stack by configuring the **DLNA_LOGFILE_PATH** environment variable of the program. For example, if you want to output the logs to **/tmp/dlnalog.txt**, run the following command:

```
export DLNA_LOGFILE_PATH="/tmp/dlnalog.txt"
```

Run the program. Then the logs are output to **/tmp/dlnalog.txt**.

Note that after the log directory is specified, all logs are output to the same directory.

6 HiMultiScreen

6.1 Overview

The HiMultiScreen is a HiSilicon application component that enables handheld devices to display STB screen outputs in the local area network (LAN) and remotely control the STB. It consists of two parts: client software that runs on handheld devices and server software that runs on the STB. Figure 6-1 shows its networking. The audio control function (Speech) requires access to the Internet.

Figure 6-1 HiMultiScreen networking

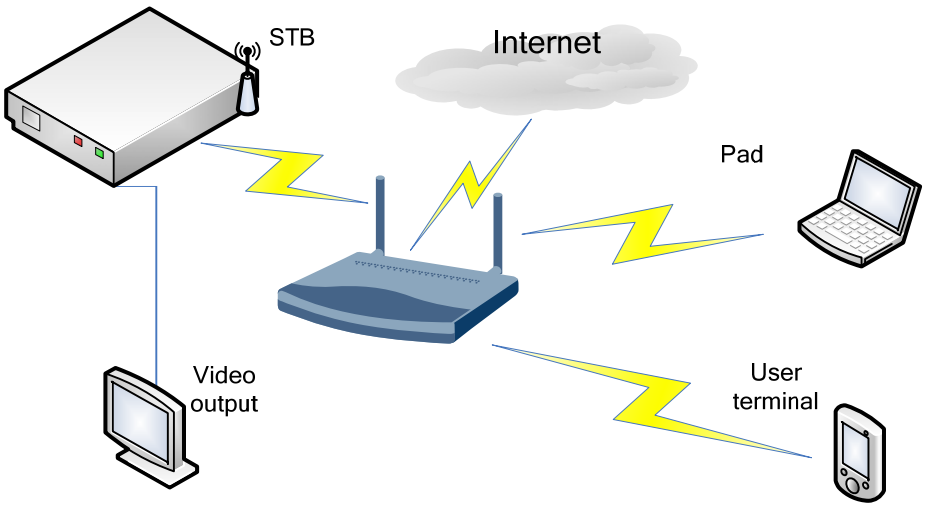


Figure 6-2 shows the architecture of the HiMultiScreen at the STB server end. The yellow parts indicate Java code delivered as source code, and the green parts indicate C/C++ code provided as .so libraries.

Figure 6-2 HiMultiScreen STB end architecture

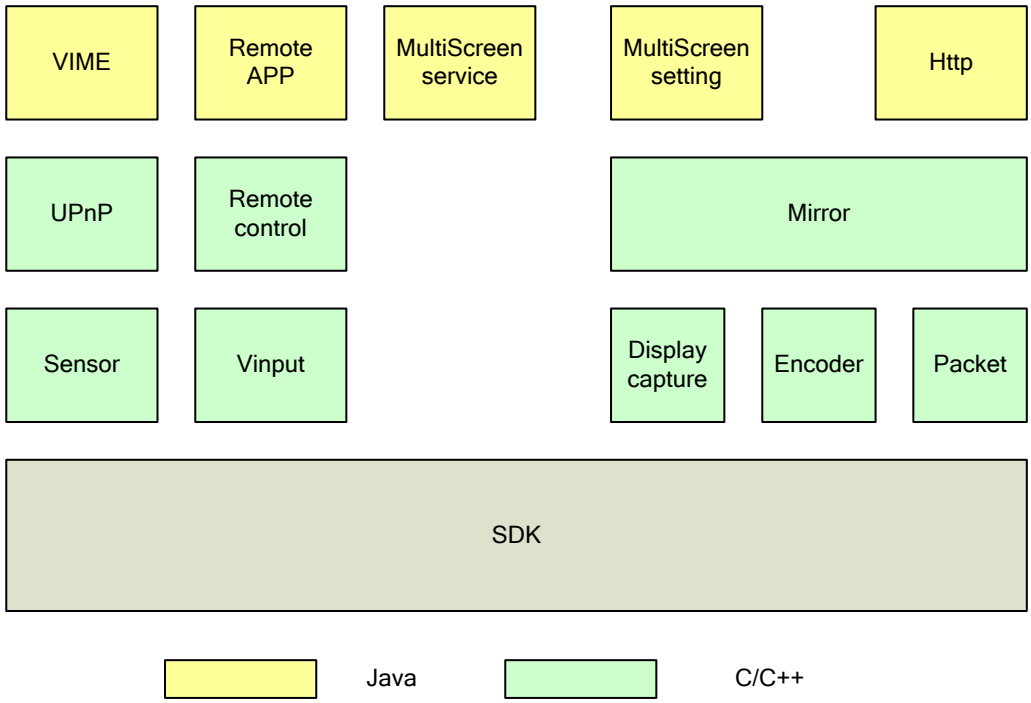
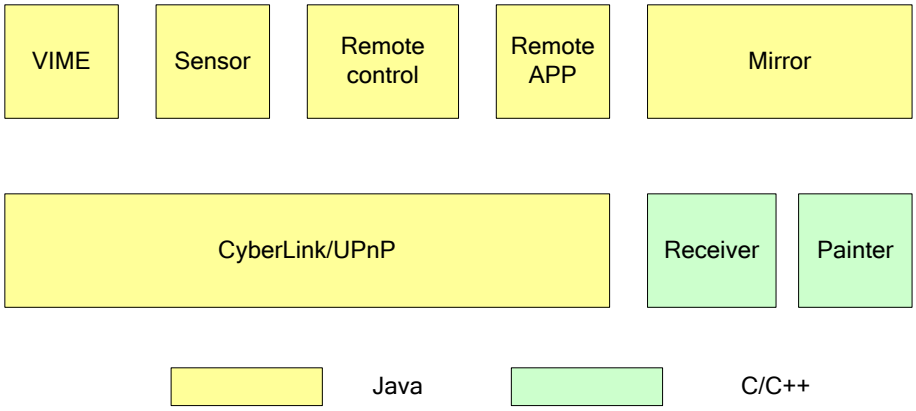


Figure 6-3 shows the architecture of the HiMultiScreen at the handheld device end. CyberLink/UPnP in yellow is the open-source library and is provided as source code. Other modules in yellow are code of the application layer and are provided as source code. The green parts are provided as .so libraries.

Figure 6-3 HiMultiScreen handheld device end architecture



6.2 Important Concepts

[UPnP]

Universal plug and play (UPnP) is a set of network protocols that allow various devices in the residential networks (data sharing, communications, and entertaining) and company networks to seamlessly interconnect with each other and simplify network implementation. For details, see http://www.upnp.org/.

[Mirroring]

Mirroring indicates transferring graphics or videos from the STB to the handheld device end for display.

6.3 Function Description

6.3.1 Features

Currently the HiMultiScreen consists of the device detection module, Mirror module, RemoteControl module, virtual input method editor (VIME), Sensor module, and Speech module. The functions of each module are described as follows:

- Device detection

  This module allows the handheld device client to detect STBs that provide multi-screen services in the LAN by using the UPnP protocol and describe a controlled STB in the format of *device name+IP address*.

- Mirror

  This module displays the same graphics and videos as those displayed on the STB, and sends the user controlling commands to the STB end. The controlling commands allow you to perform simulated touchscreen operations on the controlled STB.

- RemoteControl

  This module simulates most of the frequently used keys of the HiSilicon infrared remote control, which allows you to enjoy the experience similar to that of operating the hardware remote control.

- VIME

  The VIME allows you to enter texts to the STB end from the handheld device end. The client is deployed at the handheld device end, and the server software is deployed at the STB end. The client receives user input data and transmits the data to the server end, then the server end submits the data to the current textbox of the STB.

- Sensor

  This module uses the sensor (G-sensor) at the handheld device end to control the motion sensing games at the STB end.

- Speech

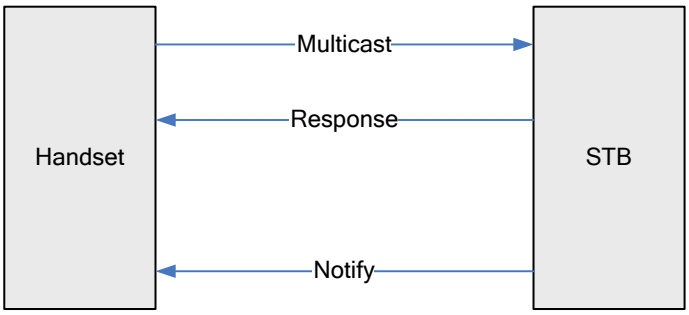
  This module uses the client MIC as the voice input source, obtains the speech recognition result through interaction between the Internet and iFly voice cloud, and submits the recognition result to the STB server end. The Speech module can implement various voice functions such as intelligent conversation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

6.3.2 Function Implementation

6.3.2.1 Device Detection

The device detection module complies with the UPnP protocol 1.0. The handheld device client uses the CyberLink UPnP development kit. Figure 6-4 shows the interaction between the handheld device and the STB during device detection.

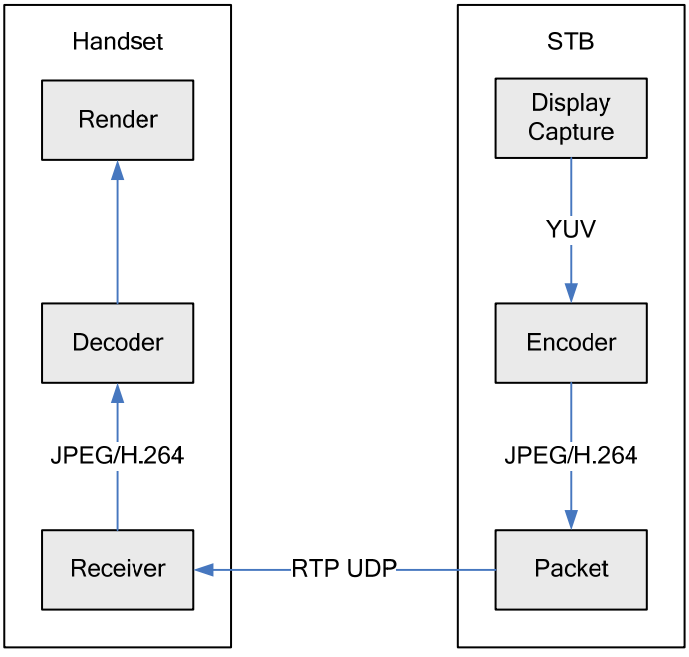
Figure 6-4 Device detecting interaction



6.3.2.2 Mirroring

Figure 6-5 shows the interaction between the handheld device and the STB during mirroring.

Figure 6-5 Mirroring interaction

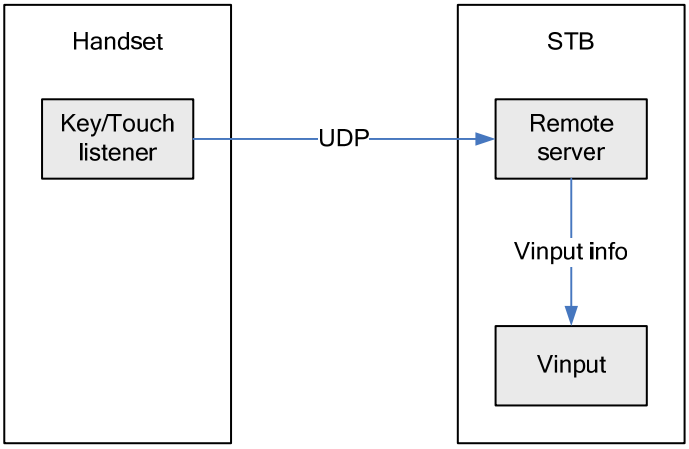


The code for encoding and transmitting at the STB end and decoding and receiving at the handheld device end is provided as .so files.

6.3.2.3 Remote Control

The RemoteControl module uses the User Datagram Protocol (UDP) packets to transmit the touched key information to the STB end. Figure 6-6 shows the RemoteControl interaction.

Figure 6-6 RemoteControl interaction



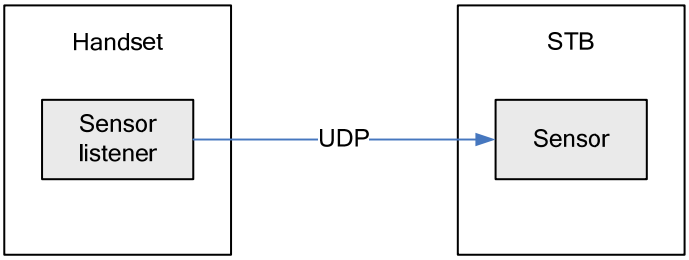
The RemoteControl interaction uses proprietary protocols.

The default port at the STB end for RemoteControl is port 8822.

6.3.2.4 Sensor

The Sensor module transmits data packets to the specific port over the UDP protocol. The motion sensing type in the data packets is selected based on the type supported by the handheld device. Figure 6-7 shows the interaction process.

Figure 6-7 Sensor interaction

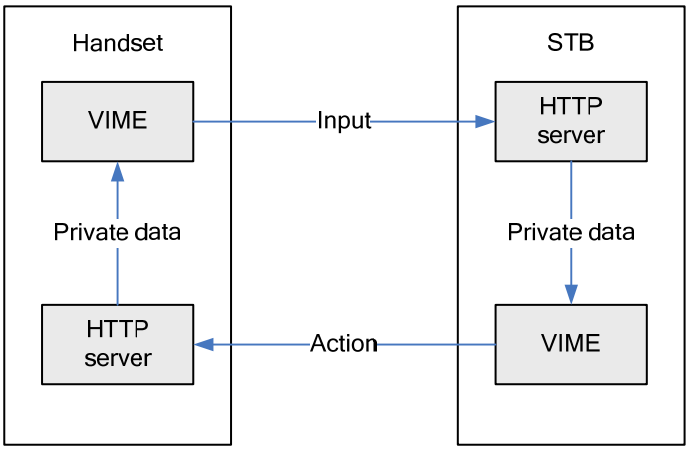


The default port at the STB end for the Sensor module is port 10021.

6.3.2.5 VIME

The VIME interaction is implemented by transmitting XML between the HTTP servers at the handheld device and STB ends. Figure 6-8 shows the interaction process.

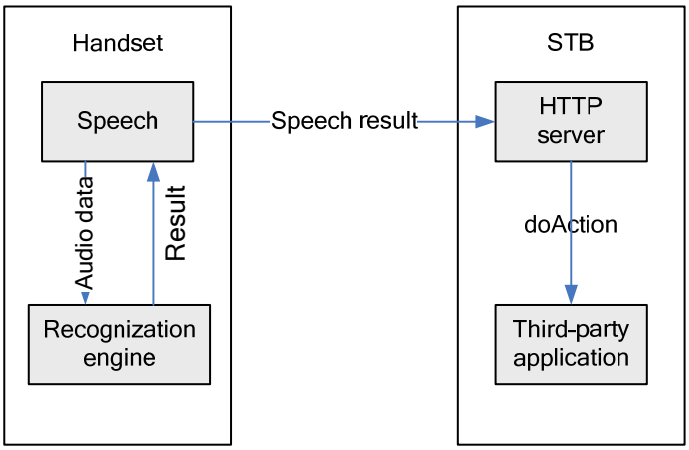
Figure 6-8 VIME interaction



6.3.2.6 Speech

The Speech module on the handheld device side interacts with the speech recognition engine over the Internet to obtain the speech identification result, and then sends the result to the HTTP server on the STB side. Figure 6-9 shows the interaction between the handheld device and the STB.

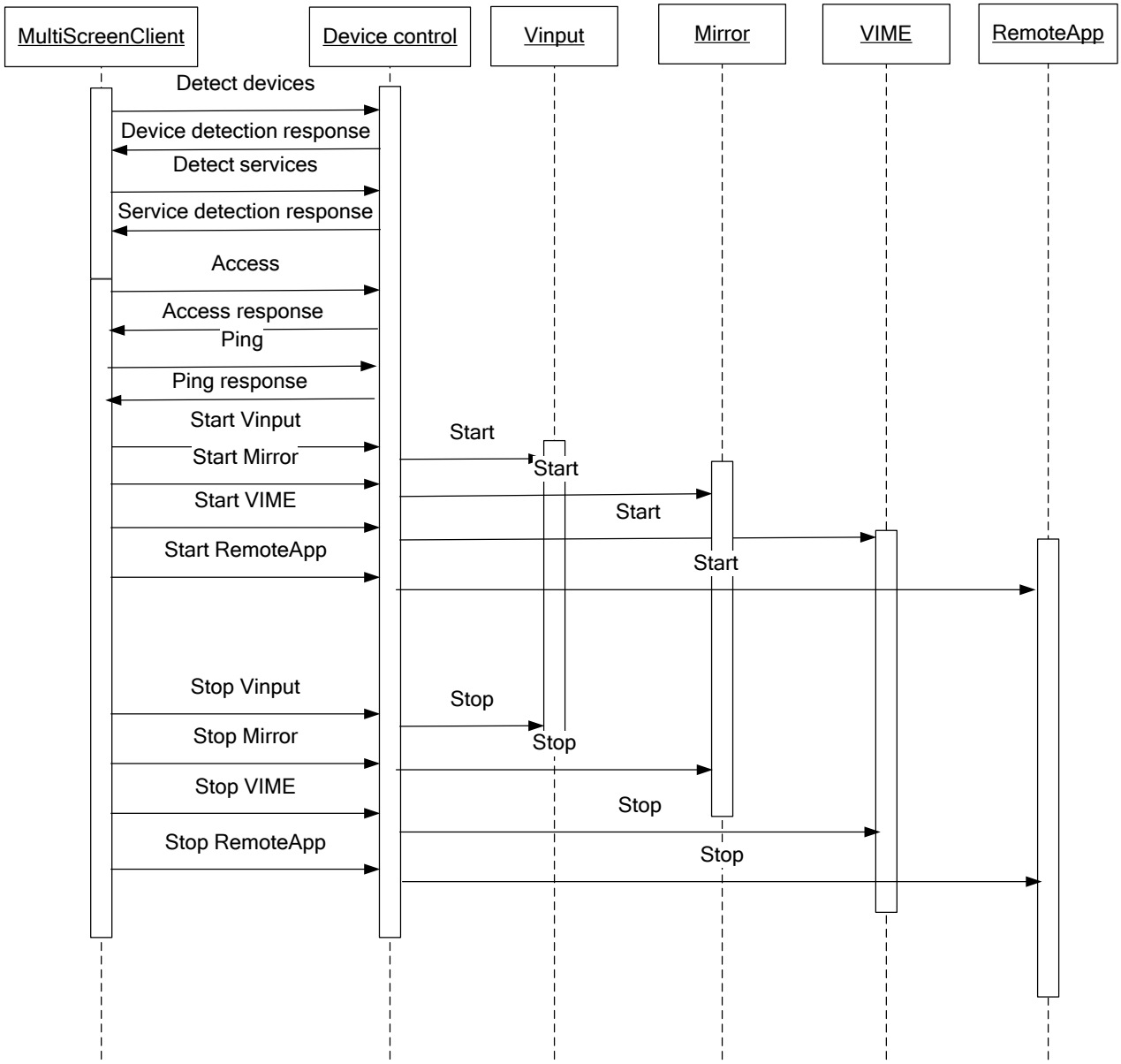
Figure 6-9 Speech interaction



6.3.2.7 Working Process

Figure 6-10 shows the overall process of interaction between the multi-screen client and the STB.

Figure 6-10 Overall process



6.4 Development Guide

NOTE

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

6.4.1 Overview

Unlike other components which are delivered as APIs, the HiMultiScreen is delivered as an application layer component, and its source code is open at the application layer. The directory for storing the source code is as follows:

- STB end:
  device/hisilicon/bigfish/hidolphin/component/himultiscreen/android/packages/apps
- Handheld device end:
  device/hisilicon/bigfish/hidolphin/component/himultiscreen/handset_software/MultiScreen

Because the implementation of protocol interaction (Mirror, VIME, RemoteControl, and Sensor) is delivered as .so libraries, you need to focus on customized UI development of the HiMultiScreen. The following sections describe the compilation and installation of multi-screen applications and UI-related code, familiarizing you with the open-source code and facilitating UI customization.

6.4.2 Compilation and Installation at the STB End

The HiMultiScreen at the STB end is compiled by default during complete compilation of the Android system. The HiMultiScreen component is included in the images by default.

6.4.2.1 Incremental Compilation

After modifying the application, you can implement incremental compilation. Ensure that the server compilation environment is set up as required, and complete compilation of the Android system is performed. Take the MultiScreenServer as an example. Run the **mm** command in **device/hisilicon/bigfish/hidolphin/component/himultiscreen/android/packages/apps/MultiScreenServer**. **MultiScreenServer.apk** is generated later in **/out/target/product/Hi3798MV100 system/app**.

6.4.2.2 Deployment

After the Android application package (APK) compilation, you can perform either of the following operation:

- Generate an image and directly burn it.
- Install the APK by running the **adb push** command.

Assume that the IP address of the STB is 10.161.179.2, and **MultiScreenServer.apk** is stored in **H:/** of the PC, run the following command:

```
adb connect 10.161.179.2
adb remount
adb push H:/MultiScreenServer.apk system/app
```

6.4.3 STB Applications

The HiMultiScreen applications at the STB end include MultiScreenServer and VIME.

6.4.3.1 MultiScreenServer

MultiScreenServer mainly includes MultiScreenService.

MultiScreenService starts when the system starts. It initializes the MultiScreen and adds the STB to the LAN as the UPnP root device. It also starts the VIME and Speech based on the callback request from the bottom layer. This part has no UI.

6.4.3.2 VIME

The VIME code registers a virtual input method with the STB end to enable the STB end to receive inputs from the client. This part involves only one message UI, and the message processing logic is fixed, therefore you are advised not to modify it.

6.4.4 Customized Development at the STB End

6.4.4.1 UI Customization

The UI customization at the STB end mainly involves the speech display UI SpeechDialog. SpeechDialog is inherited from Dialog. You can customize the UI by replacing the resources. For details about the Android UI development, see http://developer.android.com/guide/components/index.html.

6.4.4.2 Configuring the Mirror Transmission Format

Mirror transmission in JPEG and H.264 formats is supported. You can configure the format by modifying the mirror transmission format configuration file **compath264list.xml** in the **himultiscreen/android/packages/apps/MultiScreenServer/res/raw** directory. When a handheld device contained in **compath264list.xml** accesses the multiscreen service, the H.264 format is used during mirror transmission. When a handheld device that is not listed in **compath264list.xml** accesses the multiscreen service, the JPEG format is used during mirror transmission. For example, to set the mirror transmission format to H.264 for the handheld device Huawei Mate 1, add information about Huawei Mate 1 as a device node in **compath264list.xml**.

- Device mode

  In **compath264list.xml**, information of each handheld device is contained in a device node as follows:

  ```
  <device>
  <buildModel>HUAWEI MT1-U06</buildModel>
  <sdkVersion>4.1.2</sdkVersion>
  </device>
  ```

  where:

  - **buidlModel** indicates the device model. For example, the model of Huawei Mate 1 is HUAWEI MT1-U06.
  - **sdkVersion** indicates the Android version of the handheld device.
- Configuration modes of the mirror transmission format
  - Modify the compatibility configuration file **compath264list.xml** and push it to the **data/data/com.hisilicon.multiscreen.server/cache/** directory by running the **adb push** command.
  - Modify **compath264list.xml**, and compile and burn the image.

6.4.5 Compilation and Installation of the Client

6.4.5.1 Overall Requirements

For details about how to set up the client compilation environment, see Chapter 2 "Development Environment Configuration".

- OS

  The client program must be installed on handheld devices that run Android 2.3 or later.
- Compilation requirement

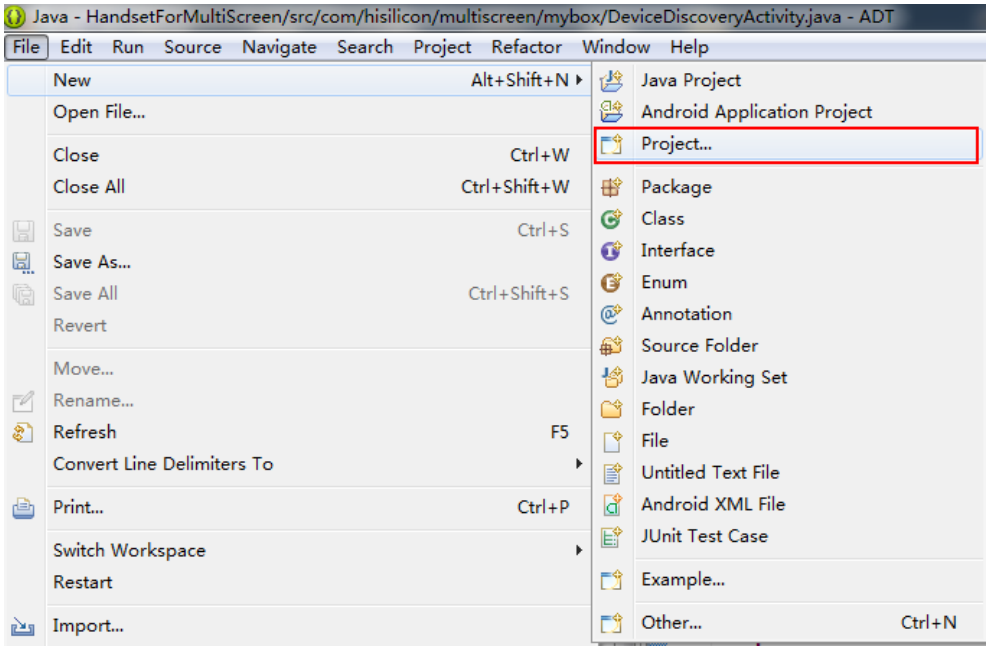
  The compilation of the client program is based on the Android 4.0 SDK.

6.4.5.2 Procedure

Step 1 Find the **device/hisilicon/bigfish/hidolphin/component/himultiscreen/handset_software/MultiScreen** directory in the SDK source code directory.

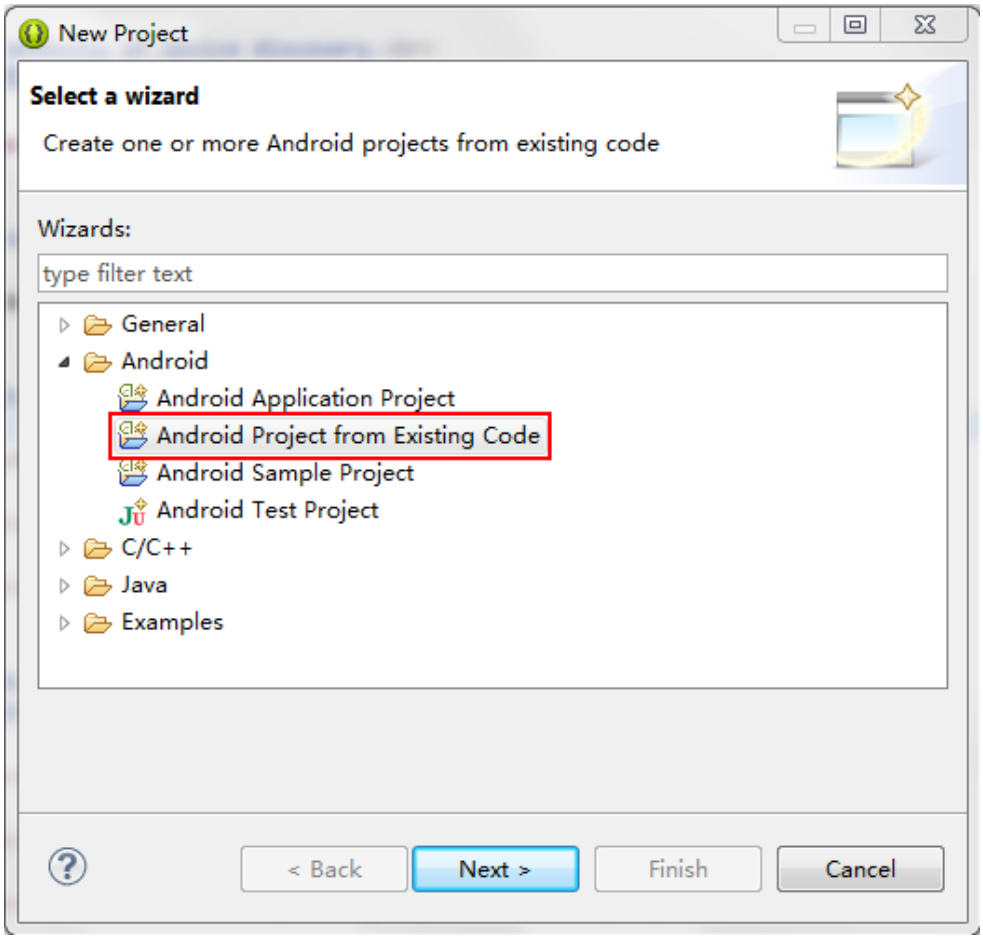
Step 2 Choose **File** > **New** > **Project** to create an Android project, as shown in Figure 6-11.

Figure 6-11 Creating an Android project



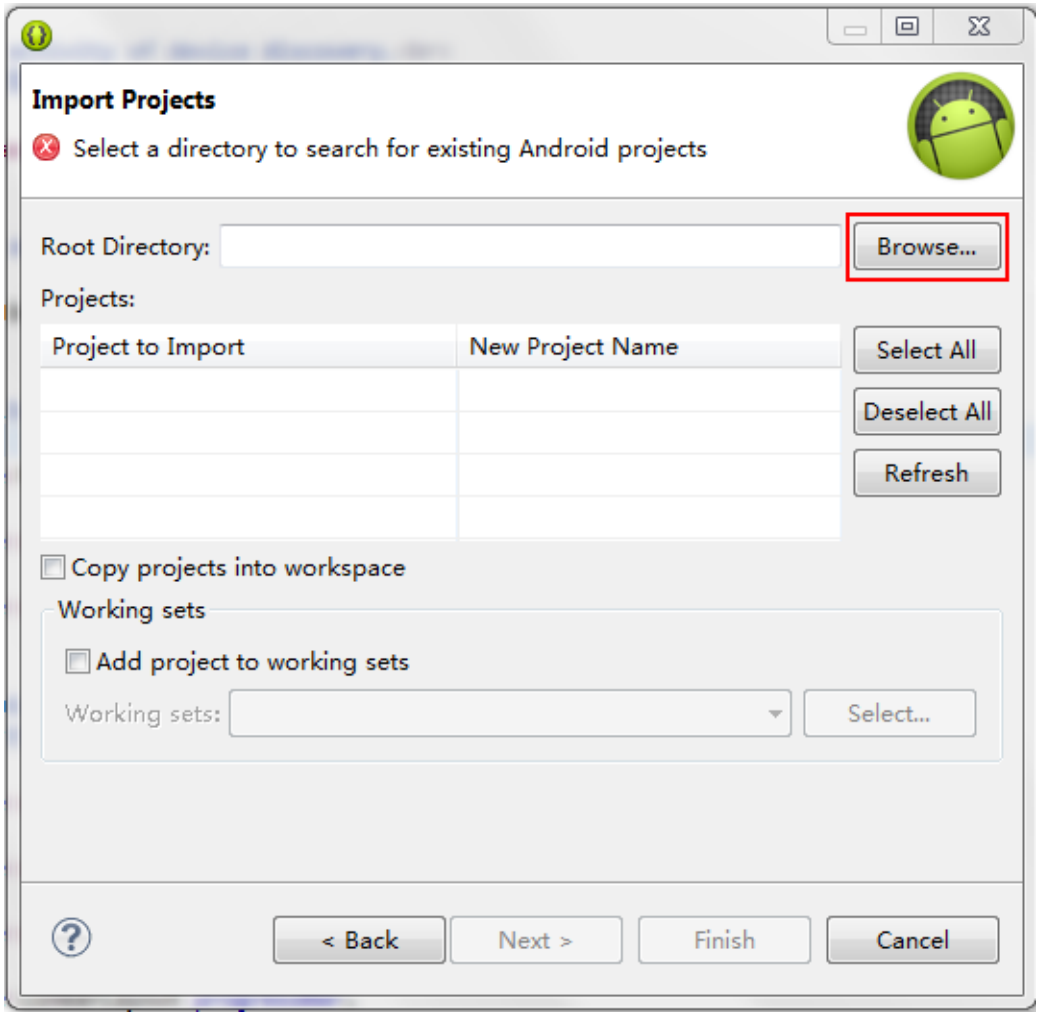
Step 3 Choose **Android** > **Android Project from Existing Code**, and click **Next**, as shown in Figure 6-12.

Figure 6-12 Selecting Android Project from Existing Code



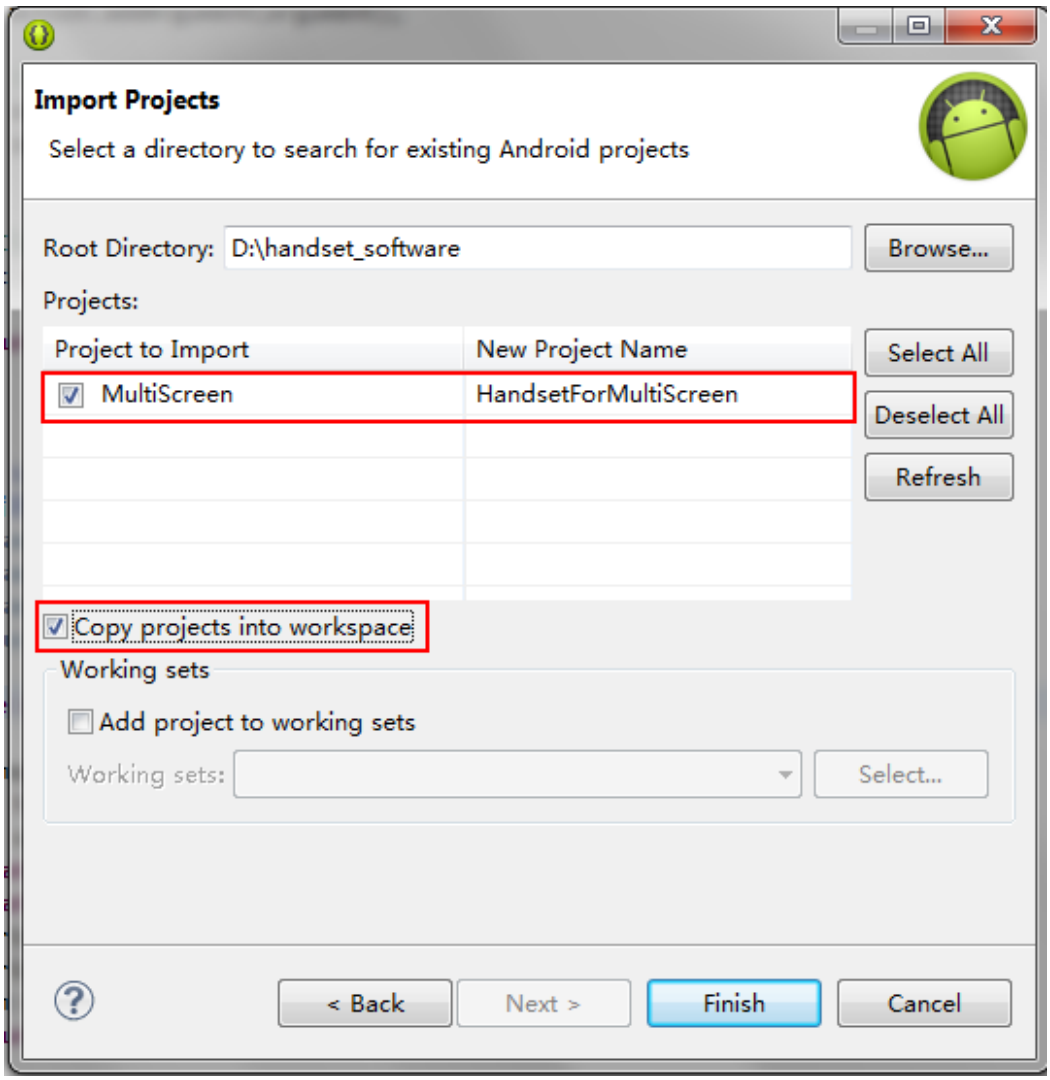
Step 4 Click **Browse**, and select the decompressed MultiScreen project, as shown in Figure 6-13.

Figure 6-13 Importing the project file



Step 5 Select the project to be imported, and select **Copy projects into workspace**, as shown in Figure 6-14.

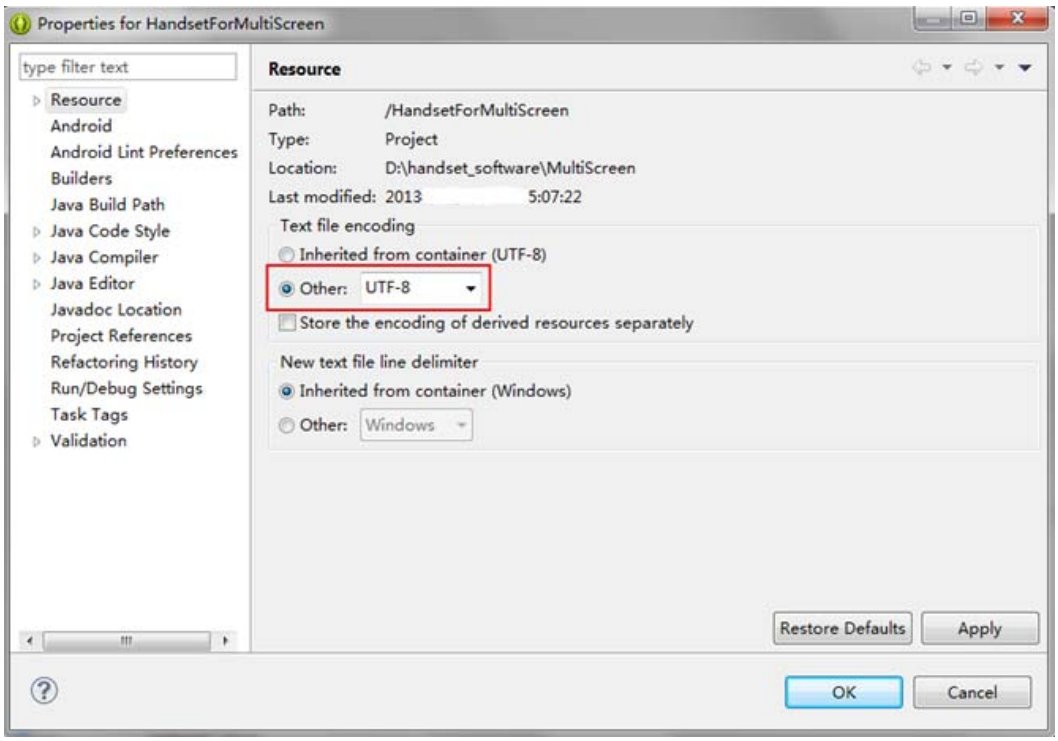
Figure 6-14 Setting the project to be imported



Step 6 Set the project encoding format.

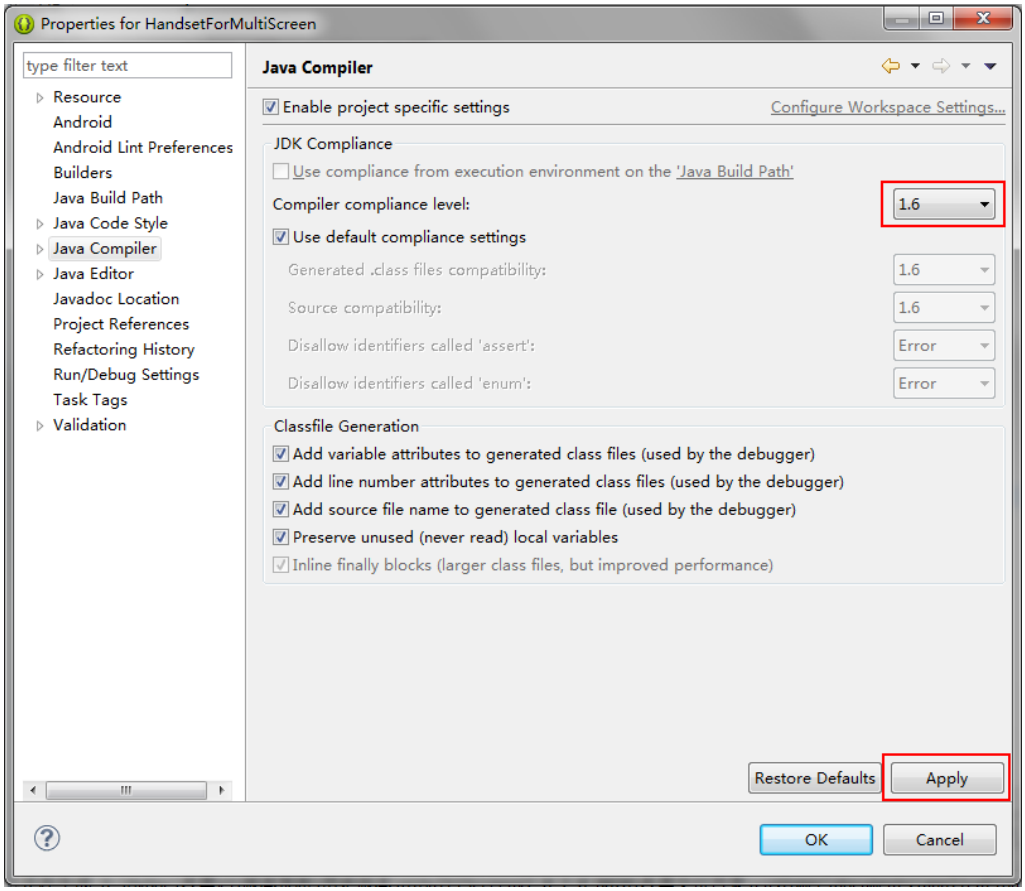
1. Choose **Project** > **Properties**.
2. Click **Resource**, and select **UTF-8** from the **Other** drop-down list of **Text file encoding**.
3. Click **Apply** to enable the setting to take effect.
4. Click **OK** and exit.

Figure 6-15 Setting the encoding format



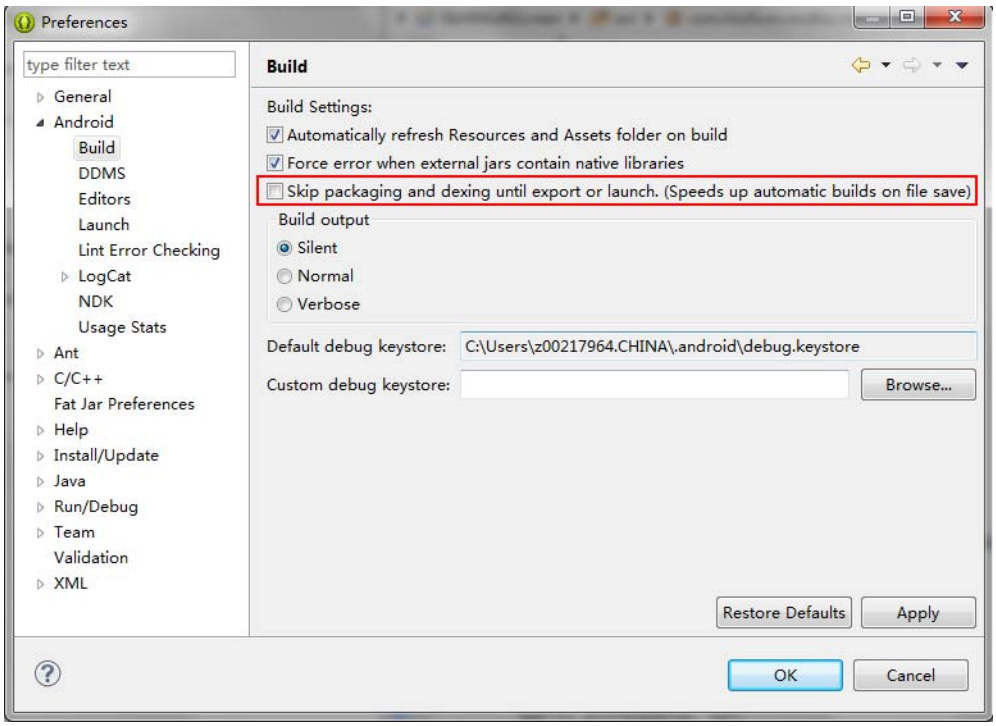
Step 7 Choose **Project** > **Properties**, click **Java Compiler**, and set **Compiler compliance level** to **1.6**, as shown in Figure 6-16.

Figure 6-16 Setting the JDK version



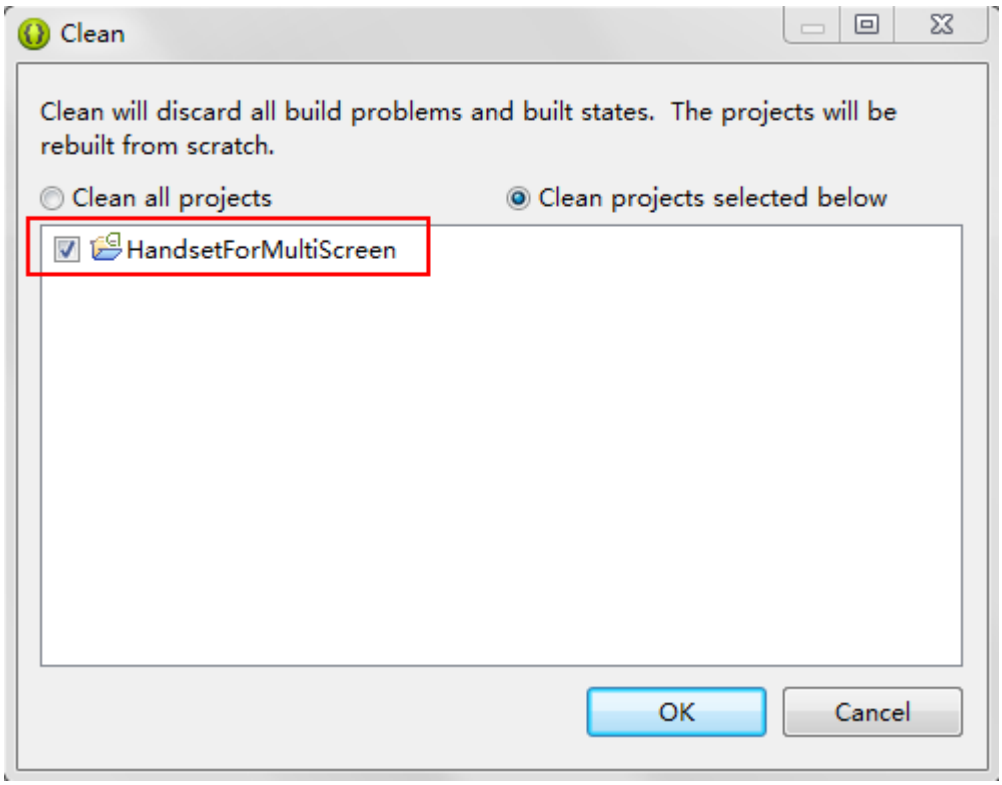
Step 8 Choose **Window** > **Preferences**, choose **Android** > **Build**, and deselect **Skip packaging and dexing until export or launch**, as shown in Figure 6-17.

Figure 6-17 Setting APK generation



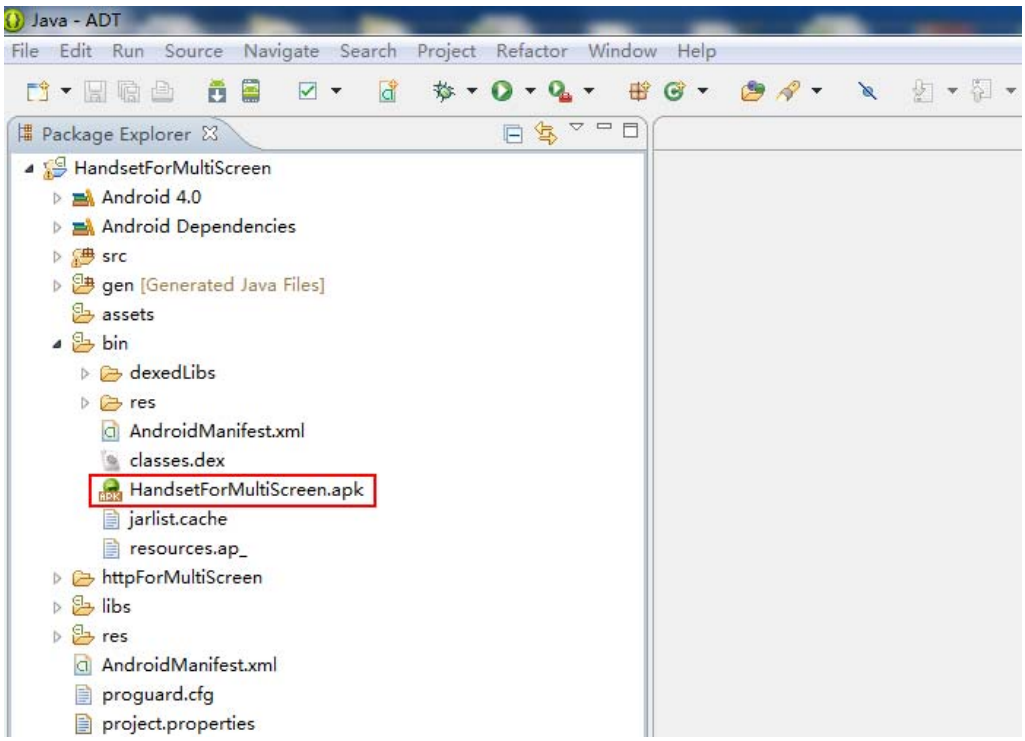
Step 9 Choose **Project** > **Clean**, select **Clean projects selected below**, select **HandsetForMultiScreen**, and click **OK** to implement the compilation, as shown in Figure 6-18.

Figure 6-18 Compiling the project



After compilation is complete, the file **HandsetForMultiScreen.apk** is generated in the directory as shown in Figure 6-19.

Figure 6-19 Generating the APK file



Step 10 If **HandsetForMultiScreen.apk** is stored in the root directory of drive H, run the following command:

```
adb install H:/HandsetForMultiScreen.apk
adb uninstall com.hisilicon.multiscreen.mybox
```

----End

6.4.6 Client Applications

The client is named **MyBox** after installation. It consists of the welcome UI, device detection UI, Mirror UI, settings UI, RemoteControl UI, and RemoteAPP UI.

6.4.6.1 UI

The UI code is stored in src/com/hisilicon/multiscreen/mybox.

- Welcome UI

  The code files are **GuideActivity.java** and **GuideGallery.java**. As a pop-up window, the welcome UI is opened with the device detection UI when the client starts for the first time. It provides instructions on how to use the multi-screen application. You can read the instructions by sliding the screen.

- Device detection UI

  The code file is **DeviceDiscoveryActivity.java**. The device detection UI is automatically displayed by default after the client starts. You can update the available STBs periodically on this page, or manually search for STBs. After the search is complete or a

notification from the STB end is received, the list of detected devices is displayed for selection or connection.

- Mirror UI

  The code file is **MultiScreenActivity.java**. The Mirror UI is displayed after you select a specific STB multi-screen service end on the device detection UI. It is the main UI. You can check the network status, initialize the Mirror, Touch, Sensor, and VIME modules, and listen to and transmit the touch events of the Mirror and RemoteControl modules on the Mirror UI. In addition, it provides jumping links to the setting UI, RemoteControl UI, and RemoteAPP UI.

- RemoteAPP UI

  The code file is **RemoteAppActivity.java**. On the RemoteAPP UI, you can obtain the icons and names of STB applications by list and start a specific application.

- RemoteControl UI

  The code file is **RemoteControlActivity.java**. The RemoteControl UI displays the keys of the remote control.

- Setting UI

  The code file is **SettingActivity.java**. The Setting UI allows you to enable or disable the G-sensor and virtual input method.

- VIME UI

  The code file is **ContentInputActivity.java**. It is stored in **src/com/hisilicon/multiscreen/vime/**. The VIME UI is the remote input UI, which can enable the input method of the handheld device for input.

- BaseActivity

  The code file is **BaseActivity.java**. BaseActivity consists of basic activities of the multi-screen applications. It implements common features of the UIs.

6.4.6.2 Services

The client services are the implementation of service subclasses which implement background service logic of the client. You are advised not to modify the code.

- MultiScreenControlService

  The code file **MultiScreenControlService.java** is stored in **src/com/hisilicon/multiscreen/mybox/**. MultiScreenControlService is inherited from the controlPoint class of the UPnP. It implements the logic of device detection, access control, network status check, and mirror control.

- SensorService

  The code file **SensorService.java** is stored in **src/com/hisilicon/multiscreen/gsensor/**. SensorService starts the motion sensing service of the handheld device, obtains the motion sensing data of the client, and sends the data to the STB end.

- VIMEClientControlService

  The code file **VIMEClientControlService.java** is stored in **src/com/hisilicon/multiscreen/vime/**. VIMEClientControlService is the virtual input method control service. It receives, sends, and processes control messages of the virtual input method and synchronizes with the STB status.

6.4.7 Customized Development of the Client

6.4.7.1 Customizing the UI

You can customize the UIs based on the preceding sections. For details about the Android UI development, see http://developer.android.com/guide/components/index.html.

Note that the welcome UI needs to be updated if you modify other UIs.

6.4.7.2 Customizing Remote Control Keys

You are advised to customize the RemoteControl UI based on your infrared remote control. The original UI corresponds to the common HiSilicon remote control. The related keys used in the common HiSilicon remote control are provided in the code. If there are extended keys at the STB end, you can modify the code based on the following description.

Add the required keys to the KeyInfo class in **src/com/hisilicon/multiscreen/protocol/message/KeyInfo.java**.

```
public static final int KEYCODE_XX = 0xXX;
```

Add the button image to **RemoteControlActivity.java**, and add processing in the click event listening.

Processing of only short press operations is supported:

```
if (event.getAction() == MotionEvent.ACTION_DOWN)
{
   //Modify the Button image
}
else if (event.getAction() == MotionEvent.ACTION_UP)
{
    //Modify the Button image
    mKeyboard.sendDownAndUpKeyCode(KEYCODE_XX);
}
```

Processing of long press operations is supported:

```
if (event.getAction() == MotionEvent.ACTION_DOWN)
{
if (mStartTime == 0)
    {
        ?
        mStartTime = SystemClock.elapsedRealtime();
        if (mStartTime != 0)
        {
sendLongPressKeyRequest(KeyInfo.KEYCODE_XX,
                                KeyInfo.KEY_EVENT_DOWN);
        }
    }
    else
    {
```

```
            LogTool.e("wrong order");
            mStartTime = 0;
sendLongPressKeyRequest(KeyInfo.KEYCODE_XX,
                                KeyInfo.KEY_EVENT_UP);
        }
}
else if (event.getAction() == MotionEvent.ACTION_UP)
{
    ?
    mStartTime = 0;
sendLongPressKeyRequest(KeyInfo.KEYCODE_XX,
KeyInfo.KEY_EVENT_UP);
}
else if (event.getAction() == MotionEvent.ACTION_MOVE)
{
if (mStartTime != 0)
    {
        if (mCount < 20)
        {
            mCount++;
        }
        else
        {
            sendLongPressKeyRequest(KeyInfo.KEYCODE_XX,
                                KeyInfo.KEY_EVENT_DOWN);
            mCount = 0;
        }
     }
}
```

6.4.7.3 Changing the Application Icon and Name

Replace **icon.png** in **res/drawable**, **res/drawable-hdpi**, **res/ drawable-ldpi**, and **res/drawable-mdpi** with the customized **ICON.png**.

Change the application names in **res/values/string.xml**, **res/values-en-rUS/string.xml**, and **res/values-zh-rCH/string.xml**.

```
<string name="app_name">XXXX</string>
```

6.5 Debugging Guide

6.5.1 Logs

The Android inherent log system Logcat is used. You can filter logs by using the class name as the tag in DDMS during debugging.

6.5.2 Connection

If the client cannot connect to the STB end, you are advised to check whether the STB properly connects to the handheld device by running the **ping** command at the STB end.

7 HiTranscoder

7.1 Overview

Based on the SDK hardware capability, the HiTranscoder provides various modules such as the encoding, synchronization, buffer, encapsulation, and protocol modules to ensure that media data on the board can be obtained and played on the PC or handheld device. The HiTranscoder module allows accesses from various clients and provides data in multiple encapsulation formats.

The occupied memory varies according to the video and audio encoding rates, mostly the video encoding rate. Table 7-1 describes the relationship between the video encoding rate and occupied memory.

Table 7-1 Relationship between the video encoding rate and occupied memory

Video Encoding Rate	Occupied Memory
256 kbit/s	560 KB
512 kbit/s	560 KB
1 Mbit/s	1060 KB
2 Mbit/s	2 MB

The occupied network bandwidth is also closely related to the audio and video bit rates. The occupied network bandwidth is about the sum of the video bit rate and the audio bit rate. Table 7-2 describes the relationship.

Table 7-2 Relationship between the video and audio encoding rates and occupied bandwidth

Video Bit Rate	Audio Sampling Rate	Occupied Bandwidth
256 kbit/s	48 kHz	280 Kbyte/s
512 kbit/s	48 kHz	536 Kbyte/s
1 Mbit/s	48 kHz	1 Mbyte/s
2 Mbit/s	48 kHz	2 Mbyte/s

The CPU usage is stable. For example, on the Hi379M V100 board, the CPU usage is 3% to 8% when all the services are running, and the main frequency is 30 MHz to 80 MHz.

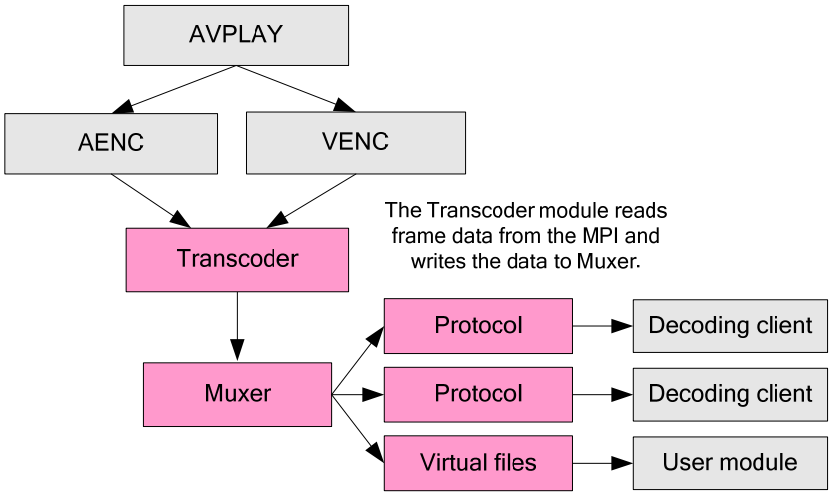
7.2 Features

The HiTranscoder module has the following features:

- Packages local .flv and .ts files
- Allows you to extend dynamic libraries based on other protocols and add customized functions.
- Allows multiple clients to access the protocol server.

Figure 7-1 shows the functions and architecture of the HiTranscoder module.

Figure 7-1 Functions and architecture of the HiTranscoder modules



The HiTranscoder module consists of the following three modules:

- Transcoder

  Processes the raw encoding data.
- Protocol

  Processes the responses of the protocol server, organizes data, and controls the source end.
- Muxer

  Encapsulates media data into files or protocols.

Figure 7-2 Data flowchart of the HiTranscoder module

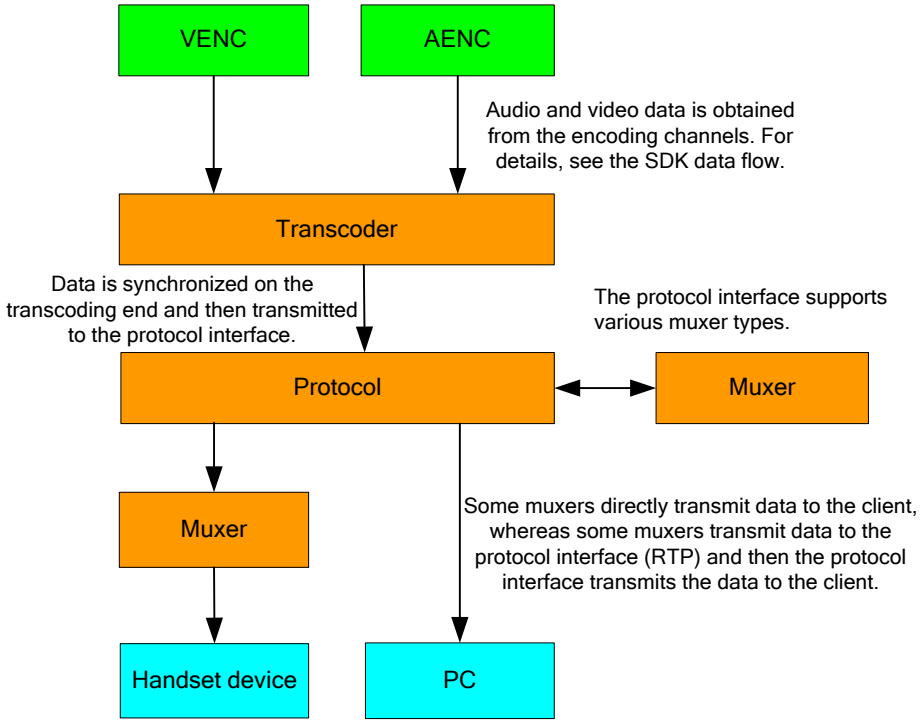


Figure 7-3 Control flowchart of the HiTranscoder module

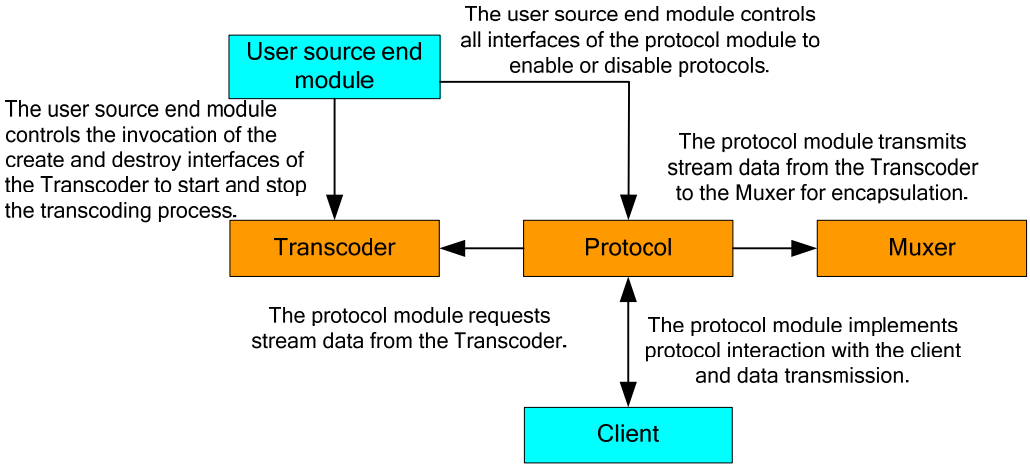
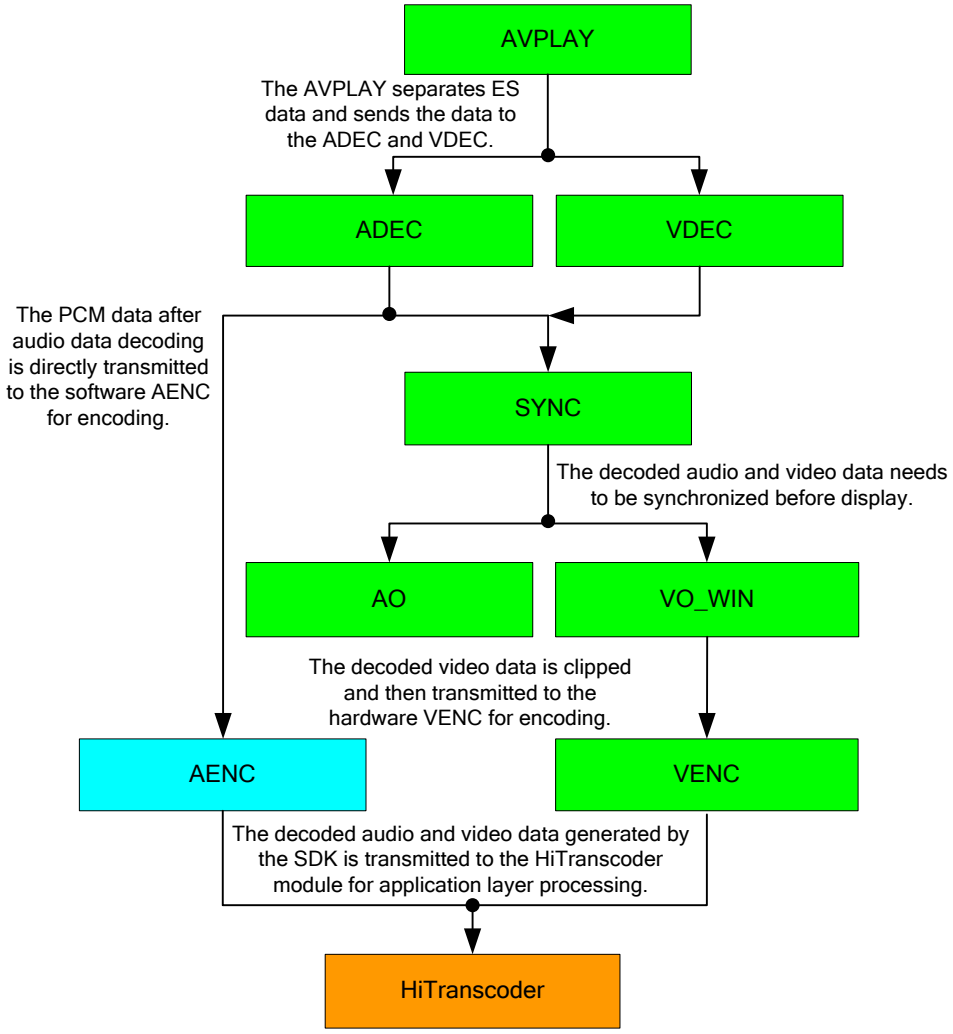


Figure 7-4 SDK HiTranscoder media data flowchart



7.3 Important Concepts

[Transcoder]

The Transcoder module transcodes data. This module creates a data channel for transmitting data from the decoding side to the encoding side by using software. It also synchronizes and buffers the data on the AV encoding side.

[Protocol]

The Protocol module processes protocols and creates a server on the board to allow access by using the handheld devices or PCs. This ensures that the transcoded media data is transmitted to clients based on the standard stream media protocol and can be received and normally played by the players running on handheld devices or PCs. The AV data is synchronized during playback and the playing delay between the server and player is reduced.

[Muxer]

The Muxer module encapsulates raw data of the Transcoder into the data that the Protocol module can use.

7.4 Development Guide

7.4.1 Module Usage

The interfaces of the HiTranscoder module include the interfaces related to the Transcoder, Protocol, and Muxer modules, which must work with each other. If you do not use the self-defined protocol libraries, the Muxer module is not required.

7.4.1.1 Initializing and Deinitializing the HiTranscoder Module

Programming Guide

Initialize the HiTranscoder module before calling other APIs of the module, and deinitialize the HiTranscoder module when it is not used.

Deinitializing the HiTranscoder module releases the resources occupied by the HiTranscoder module, such as the following:

- Allocated encoding channels and virtual windows
- Ports and threads allocated when the protocols are enabled

To initialize the HiTranscoder module, call the following three APIs:

- HI_Transcoder_Init
- HI_Protocol_Init
- HI_Muxer_Init

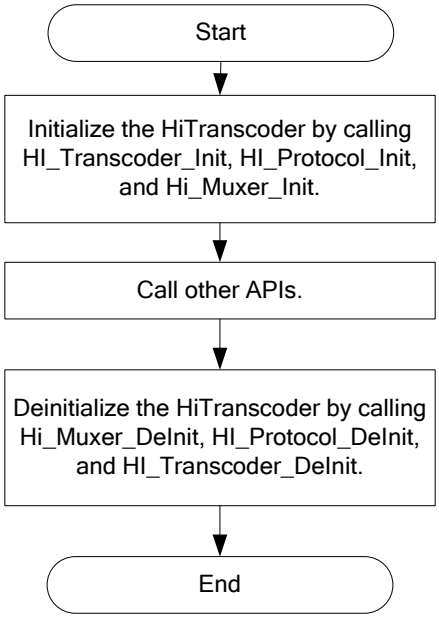
To deinitialize the HiTranscoder module, call the following three APIs:

- HI_Transcoder_DeInit
- HI_Protocol_DeInit
- HI_Muxer_DeInit

Working Process

Figure 7-5 shows the process of initializing and deinitializing the HiTranscoder module.

Figure 7-5 Initializing and deinitializing the HiTranscoder module.



7.4.1.2 Creating and Destroying a Transcoder Handle

After initializing the HiTranscoder module, create a Transcoder handle to obtain the data generated by the AENC and VENC. The created interface obtains encoded data from the AENC or VENC module and outputs the data to the Protocol module.

The prerequisites for creating the Transcoder handle are as follows:

- The AVPLAY that plays the decoded data is created and initialized.

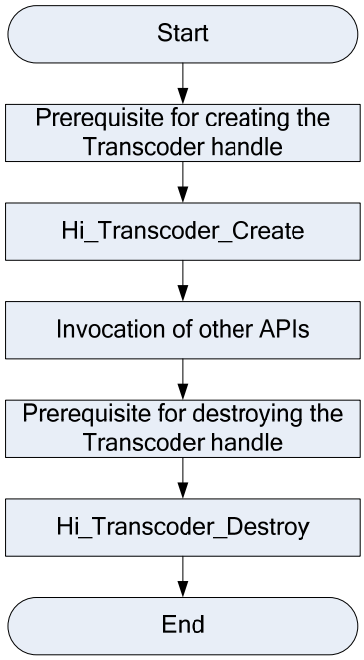
  The AVPLAY is playing or not playing. The decoded AV data of the data source is played after the AVPLAY is created. For details about this procedure, see the source code provided in "Sample Usage" and the **.h** header files of the SDK. The played streams must be AV streams. The single audio or video streams cannot be transcoded for playing.
- The HiTranscoder module is initialized.

  Destroying the Transcoder handle releases the memory that stores the stream data and various software flags. The prerequisite of calling the HiTranscoder destroy interface is that the Protocol handle is unbound from the Transcoder handle.

Working Process

Figure 7-6 shows the process of creating and destroying a Transcoder handle.

Figure 7-6 Creating and destroying a Transcoder handle



7.4.1.3 Creating and Destroying a Protocol Handle

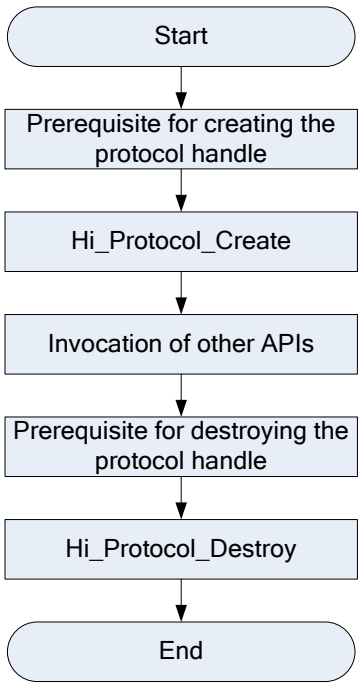
The creating and destroying process of the Protocol handle is not closely related to that of the Transcoder handle and the decoding process of the decoding source end. The Protocol handle is used to create the protocol server and enable various threads. Creating the protocol handle requires the Muxer-related interfaces, therefore the initialization process in section 7.4.1.1 "Initializing and Deinitializing the HiTranscoder Module" must be complete.

- The prerequisite for creating the Protocol handle is that the HiTranscoder module is initialized.
- The prerequisite for destroying the Protocol handle (destroying the protocol server used by the system, and exiting the threads) is that the Protocol handle is unbound from the Transcoder handle.

Working Process

Figure 7-7 shows the process of creating and destroying a Protocol handle.

Figure 7-7 Process of creating and destroying a Protocol handle



7.4.1.4 Binding and Unbinding a Protocol Handle

After a Protocol handle is bound to a Transcoder handle, data can be exchanged between the Transcoder module and the Protocol module. By using the bound Transcoder handle, the Protocol module can enable or disable the Transcoder module and obtain the data read handle to read AV ES data from the Transcoder module.

The RTSP and HTTP protocols are supported by default. You can also customize proprietary protocols. By implementing the data structure struct tagHI_ProtocolInfo_S in **include/hi_proto_intf.h**, you can register proprietary protocols by calling HI_S32 HI_Protocol_RegistProtocol(const HI_CHAR* pLibProtocolName) in the **lib*.so** format.

The prerequisites for binding a Protocol handle are as follows:

- The Protocol handle of the HiTranscoder module is created.
- The Transcoder handle of the HiTranscoder module is created.

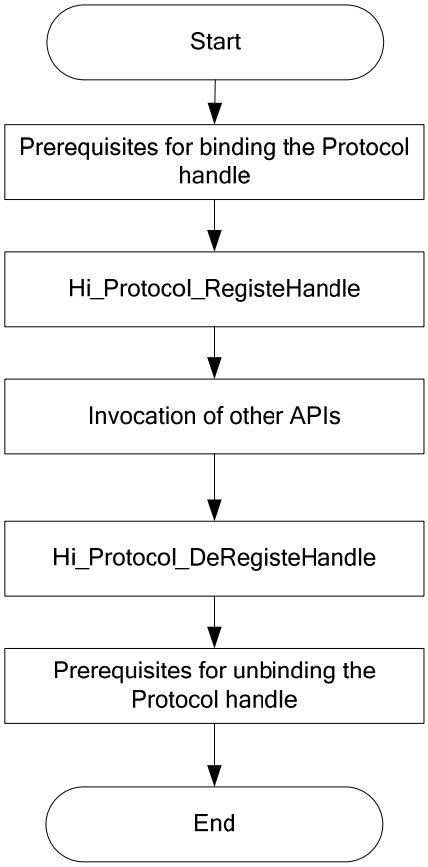
When the Protocol handle is unbound from the Transcoder handle, the handle for reading data from the Transcoder is also unbound from the Protocol module.

The prerequisite for unbinding the Protocol handle is that the Protocol handle has been bound to the Transcoder handle.

Working Process

Figure 7-8 shows the process of binding and unbinding a Protocol handle.

Figure 7-8 Process of binding and unbinding a Protocol handle



7.4.1.5 Data Handling Process of the Muxer

The Muxer module is used to encapsulate data. The encapsulated data can be saved as local files or transmitted to the network by using the Protocol module. The sample_protocol code outputs the transcoded data of the HiTranscoder module to local files.

The default encapsulation formats are RTP, FLV, TS, and ES. You can also register private data encapsulation formats. By implementing the data structure HI_MuxerInfo_S in **Hi_muxer_intf.h**, you can register private muxers by calling HI_S32 HI_Muxer_RegistMuxer(const HI_CHAR* pLibMuxerName) in the **lib*.so** format.

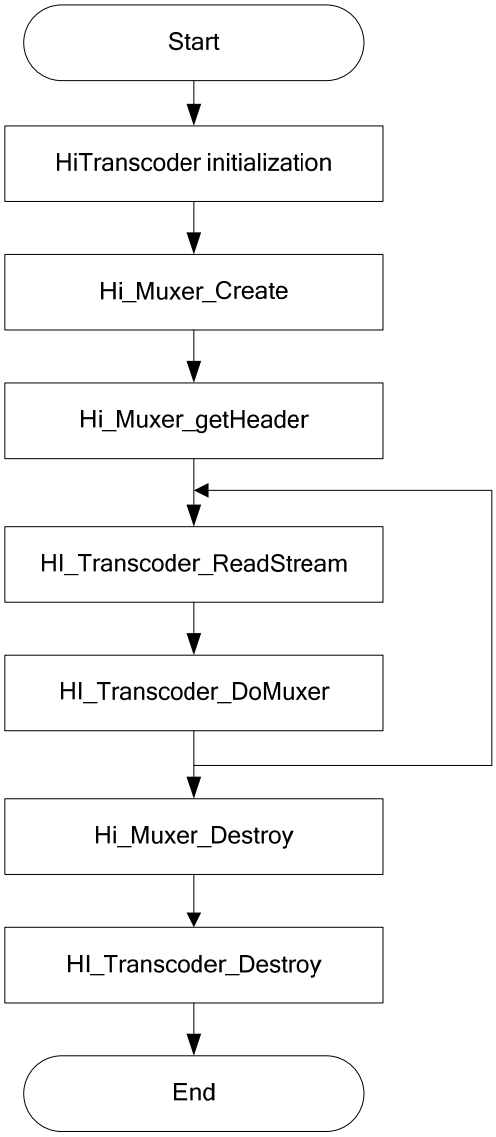
The prerequisites of using the Muxer module are as follows:

- The HiTranscoder module is initialized, as described in section 7.4.1.1 "Initializing and Deinitializing the HiTranscoder Module."
- TranscoderRegistHandle is called to register a handle for reading data from the Transcoder module.

Working Process

Figure 7-9 shows the data handling process of the Muxer module.

Figure 7-9 Muxer data handling process



7.4.1.6 Data Handling Process of the Transcoder

After HI_Transcoder_RegisterHandle is called, raw data is read from the Transcoder, encapsulated as other formats, and transmitted to the player by using the Protocol module.

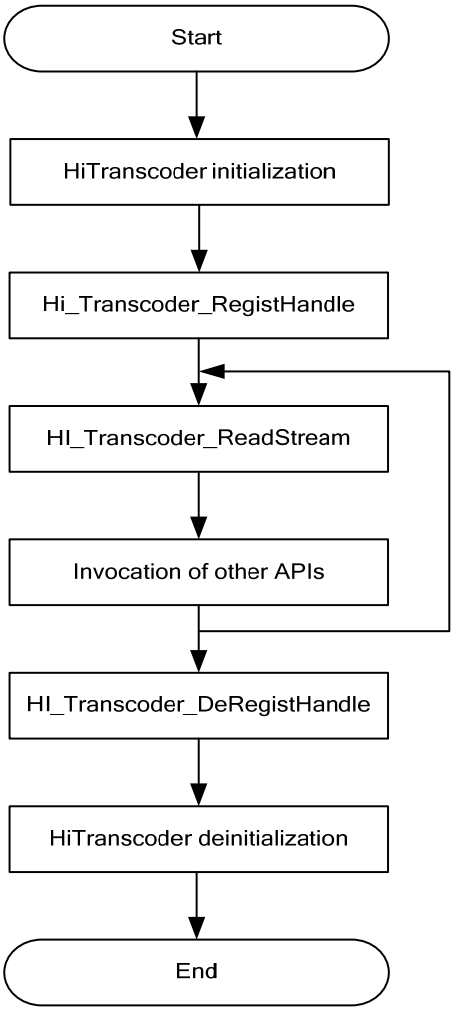
The prerequisites are as follows:

- The HiTranscoder module is initialized, as described in section 7.4.1.1 "Initializing and Deinitializing the HiTranscoder Module."
- The Transcoder handle is created by calling Hi_Transcoder_create.

Working Process

Figure 7-10 shows the Transcoder data handling process.

Figure 7-10 Transcoder data handling process



7.4.2 Configuring the HiTranscoder Running Environment

The HiTranscoder is released with the SDK on Linux and Android.

7.4.2.1 Configuring the Server Running Environment

Android HiTranscoder

To configure the Android **HiTranscoder**, perform the following steps:

Step 1 Compile the Android SDK.

Compile HiSTBAndroid VX00R001C00SPCXXX (Android) SDK by following the *Android SDK Development Guide* in the SDK release package. Burn images to the board as required after successful compilation.

Step 2 Run the transcoding-related samples in the input command line of the COM port.

TS play sample

```
sample_transcoder_tsplay file=fileNameDir f=25 w=320 h=240 pro=rtsp
```

```
b=1024*1024
```

fileNameDir is the file path name. For example, **/data/app/file.ts** must be a TS file.

DVB play sample

```
sample_transcoder_dvbplayer freq=610 srate=6875 qam=64 f=25 w=320 h=240
pro=rtsp b=1024*1024
```

Change the value of **freq** (for example, the value is 610) based on the frequency. For example, change the value of **freq** to **235**.

pipplayer sample

```
sample_transcoder_pipplayer freq=610 srate=6875 qam=64 f=25 w=320 h=240
pro=rtsp b=1024*1024
```

Change the value of **freq** (for example, the value is 610) based on the frequency. For example, change the value of **freq** to **235**.

OSD player sample

```
sample_transcoder_osdplayer f=25 w=320 h=240 pro=rtsp b=1024*1024
```

HlsPlayer Sample

```
sample_transcoder_hlsplayer file=fileNameDir w=1280 h=720 b=4*1024*1024
```

fileNameDir is the file path, for example, **/data/app/file.ts**. The file must be a TS file.

----End

Linux HiTranscoder

Transcoding-related files are stored in the source/component/hitranscoder directory of the Linux SDK.

- **include** is a transcoding API header file.
- **lib** is a dynamic library related to transcoding.
- **handset_software** is the client-related code.

Set the running environment and configure the Linux HiTranscoder as follows before running applications:

Step 1 Compile the Linux SDK.

Run **make menuconfig** in the HiSTBLinux V100R00XC00SPCXXX (Linux) SDK.

Ensure that the HiTranscoder module is selected in **menuconfig** and the configuration is successfully saved. Compile HiSTBLinux V100R002C00SPCXXX (Linux) SDK by following the *install_notes(eng).txt* in the SDK root directory. Burn images to the board as required after successful compilation.

Step 2 Compile transcoding-related samples.

Run **make sample** in the SDK (HiSTBLinux V100R00XC00SPCXXX) root directory.

The following executable files are generated in the **SDK_DIR/sample/hitranscoder** directory:

```
sample_transcoder_tsplay, sample_transcoder_dvbplayer,
sample_transcoder_pipplayer, sample_transcoder_osdPlayer,
sample_transcoder_hlsplayer
```

Step 3 Download the sample to the board over the Trivial File Transfer Protocol (TFTP).

Obtain, run, and configure the tftp-server, and download the five samples generated in step 2 to the board by using the TFTP. Run the following commands over the COM port of the board:

```
mkdir /root/trans
cd /root/trans
tftp -gr sample_transcoder_tsplay host_IP_addr(PC IP address)
tftp -gr sample_transcoder_dvbplayer host_IP_addr(PC IP address)
tftp -gr sample_transcoder_pipplayer host_IP_addr(PC IP address)
tftp -gr sample_transcoder_osdPlayer host_IP_addr(PC IP address)
tftp -gr sample_transcoder_hlsplayer host_IP_addr(PC IP address)
chmod 777 sample_transcoder_*
```

Step 4 Run transcoding-related samples in the directory where samples locate over the COM port.

TsPlay Sample

```
./sample_transcoder_tsplay file=fileNameDir  f=25 w=320 h=240 pro=rtsp
b=1024*1024
./sample_transcoder_tsplay file=fileNameDir  c=2 f=25 w=320 h=240
pro=rtsp b=1024*1024 band=16
```

fileNameDir is the file path name. For example, **/data/app/file.ts** must be n TS file.

DvbPlay Sample

```
./sample_transcoder_dvbplayer freq=610 srate=6875 qam=64 f=25 w=320 h=240
pro=rtsp b=1024*1024
```

Change the value of **freq** (for example, the value is 610) based on the frequency. For example, change the value of **freq** to **235**.

pipplayer Sample

```
./sample_transcoder_pipplayer freq=610 srate=6875 qam=64 f=25 w=320 h=240
pro=rtsp b=1024*1024
```

Change the value of **freq** (for example, the value is 610) based on the frequency. For example, change the value of **freq** to **235**.

OsdPlayer Sample

```
./sample_transcoder_osdplayer f=25 w=320 h=240 pro=rtsp b=1024*1024
band=16
```

HlsPlayer Sample

```
sample_transcoder_hlsplayer file=fileNameDir w=1280 h=720 b=4*1024*1024
```

fileNameDir is the file path, for example, **/data/app/file.ts**. The file must be a TS file.

NOTE

For details about the precautions and parameter description, see "Sample Usage."

----End

Sample Usage

Samples are described as follows:

- sample_transcoder_tsplay: transcoding TS files. That is, the client transcodes and plays video and audio in the TS file at the same time when the server plays the TS file.
- sample_transcoder_dvbplayer: transcoding the digital video broadcasting (DVB) programs. That is, the client simultaneously plays the DVB program that is being played by the server and continues to play the program even if the channel is switched on the server.
- sample_transcoder_pipplayer: transcoding DVB picture-in-picture videos. One visible DVB program is played in full screen, another DVB program is played in a small window on the upper left corner of the screen, and the program can be switched. The client transcodes the audio and video of the program in the small window at the same time.
- sample_transcoder_osdplayer: transcoding the graphics layer and video layer. The server plays TS streams and displays a picture. The client transcodes and plays the video and graphics (but not audio) at the same time.
- sample_transcoder_hlsplayer: transcoding HLS streams. The server plays TSs, and the iOS client transcodes and plays the video and audio in the TS file in real time.

CAUTION

On the Android platform, when the tsplay, dvbplayer, pipplayer, and hlsplayer transcoding samples are running, the default video layer is overlaid with the graphics layer. Therefore the video being played is hidden. You need to display or hide the video images by running commands manually.

Before you run the samples, run the following command to display the video images:

echo hide >/proc/msp/hifb0

After you stop the samples, run the following command to hide the video images and display the graphics layer:

echo show >/proc/msp/hifb0

The following describes sample parameters:

- **freq**: frequency of the played DVB, which can be changed as required. The unit is MHz. TS multiplexed streams are received within the bandwidth of the frequency.
- **f**: video frame rate, which can be changed. The frame rate can be 10, 20, 25, or 30.
- **w**: video width, which can be changed. The maximum width is 1920. The width must be an integral multiple of 4.
- **h**: video height, which can be changed. The maximum height is 1080. The height must be an integer multiple of 4.

- **pro**: protocol of the used server, which can be changed. The protocol can be RTSP, HTTP, or the local protocol. The local protocol here indicates the one for storing TS files to a local directory. Ensure that the directory has the write property.
- **b**: video bit rate, which can be changed. The minimum bit rate is 256 x 1024 and the maximum one is 8 x 1024 x 1024.
- **file**: entered file path, which can be changed as required.

Take the following precautions:

- The transcoding function depends on the audio libraries.

  The transcoding function depends on **libHA.AUDIO.AAC.encode.so**, which may cause copyright issues. Therefore, prepare **libHA.AUDIO.AAC.encode.so** by yourself. The audio libraries vary according to the audio formats.

  - libHA.AUDIO.MP3.decode.so
  - libHA.AUDIO.WMA.decode.so
  - libHA.AUDIO.AAC.decode.so

  Prepare the preceding audio decoding libraries.

  For the Android version, after obtaining these libraries, deliver audio libraries to the **/system/lib** directory by using the Android ADB tool. For the Linux version, download these audio libraries to the **/usr/lib** directory over TFTP.
- For **sample_transcoder_PipPlayer**, the MMZ buffer may be insufficient because two AVPLAYs are started. You need to increase the buffer size by following the SDK description document to ensure that the pipplayer runs properly.
- The dvbplayer and pipplayer samples cannot run together with the HiDTVPlayer provided by HiSilicon. That is, the DVB function cannot be used at the same time in two processes. If you run both services, the second service cannot be started. However, the two services can run at the same time in the same process.
- The OSD+video mirroring function demonstrated by sample_transcoder_osdplayer conflicts with the HiMultiScreen service provided by HiSilicon over resources. The two services cannot run at the same time. If you run both services, the second service cannot be started.
- After the board is burnt, a graphics layer is displayed by default for the Android version, and a blank screen is displayed by default for the Linux version. Therefore, after osdplayer is executed on the Linux version, a blank screen is displayed on the connected client. In this case, you can start another process by using samples such as tsplay and dvdplay in the **Sdk_Dir/sSample** directory of the Linux version for playing videos or displaying graphics, so that some data can be displayed after the client is connected.

Proc

1. HiTranscoder

The HiTranscoder provides the **Hitranscoder** directory in **/proc/hisi** for storing the device files of the module.

The HiTranscoder module has two entries, that is, two device files: **Hitranscoder** and **mbuf**. There is a mapping between the two types of device files, that is, one HiTranscoder device file corresponds to one mbuf device file. Because the HiTranscoder module supports multiple HiTranscoder instances at the same time (allows multiple transcoding data stream channels to run at the same time), there can be multiple pairs of HiTranscoder and mbuf device files.

The entries are named as follows (considering that there are multiple instances and the entries may be called by multiple processes at the same time):

hitranscoder0*X_PID*, **mbuf0*X_PID***

PID indicates the PID of the current process.

After the HiTranscoder module is started by calling the create and start interfaces, you can run the following commands to display proc information:

- For the HiTranscoder device
  - cat command

    Enter the following command over the serial port:

    ```
    cat /proc/hisi/hitranscoder/hitranscoder0X_PID
    ```

    The HiTranscoder-related encoding parameters, current status, and read/write information are displayed over the serial port.
  - echo command

    ```
    echo print=0/1 >/proc/hisi/hitranscoder/hitranscoder0X_PID
    ```

    If **print** is **1**, the mbuf write information is displayed, including the mbuf audio/video PTSs and load type. If **print** is **0**, the preceding information is not displayed. **print** is **0** by default.

    ```
    echo v=0/1 vidEsFileName audEsFileName
    >/proc/hisi/hitranscoder/hitranscoder0X_PID
    ```

    If **v** is **1**, the ES file starts to be written; if **v** is **0**, the write operation stops. **vidEsFileName** is the path for writing video ES streams, for example, **/data/app/vid.h264**. **audEsFileName** is the path for writing audio ES streams, for example, **/data/app/aud.aac**.**vidEsFileName** and **audEsFileName** are mandatory.
- For the mbuf device

  cat command:

  Run the following command over the serial port:

  ```
  cat /proc/hisi/hitranscoder/mbuf0X_PID
  ```

  The mbuf-related information is displayed, including the size and position of the video/audio sync buffer, number of current read handles, and pointers to read handles.

2. HiMuxer

The HiMuxer provides the **himuxer** directory in **/proc/hisi** for storing the device files of the module. This module has one proc device file named **hinuxer**.

- cat command

  ```
  cat /proc/hisi/himuxer/himuxer
  ```

  Information is displayed over the serial port, including the number of current muxer instances, and type and handle value (0xXXXXXXX) of each instance.

  For example:

  ```
  muxerNum: 1
  muxerType:ts_default muxerHanle 0xb7fb80f0
  ```

- echo command

  ```
  echo v=0/1 muxHandle FileName >/proc/hisi/himuxer/himuxer
  ```

  If **v** is **1**, files start to be written after the mux operation; if **v** is **0**, the write operation stops. **muxHandle** is the handle value of a muxer instance displayed after the **cat**

command is executed. **FileName** is the path for writing data after the mux operation, for example, **/data/app/local.ts**.

3. HiProtocol

Currently the HiProtocol supports the RTSP and HTTP protocols. The HiProtocol provides the **HiProtocol** directory in **/proc/hisi** for storing the device files of the module. Both the HTTP and RTSP protocols support simultaneous access of multiple clients, therefore the devices are created for each access session.

- For the RTSP device

  RTSP devices are named **rtspSess0X**.

  cat command:

  ```
  cat /proc/hisi/hiprotocol/rtspSess0X
  ```

  Information related to the current client session is displayed over the serial port, including the IP address for the client, RTP/RTCP port ID, and audio/video data transmission information.

- For the HTTP device

  HTTP devices are named as follows: **httpSess0X**.

  cat command:

  ```
  cat /proc/hisi/hiprotocol/httpSess0X
  ```

  Information related to the current client session is displayed over the serial port, including the IP address for the client, ID of the connected port, and data transmission information after encapsulation.

7.4.2.2 Configuring the Client Running Environment

The transcoding client developed by HiSilicon features low delay and supports the RTSP protocol. It is released as open-source code. The transcoding client supports the RTSP protocol.

- In the Android version, the client source code is stored in **device/hisilicon/bigfish/hidolphin/component/hitranscoder/handset_software** of the SDK root directory.
- In the Linux version, the client source code is stored in **source/component/hitranscoder/handset_software** of the SDK root directory.

  The **handsetsoftware** directory contains the following two sub directories:

  - **libHiTransPlayer_jni**: C-layer source code at the client, which is used to generate dynamic libraries that the player depends on.
  - **HiTransPlayer**: Java-layer source code at the client, which is used to generate the Android APK.

Compiling libHiTransPlayer_jni

The **libHiTransPlayer_jni** directory needs to be compiled by using the Android NDK tool. Set the running environment as follows before running applications:

Step 1 Run the following command to create a directory in the CLI:

```
mkdir -p ~/workspace/NDK
```

Step 2 Download the Android NDK tool from the following path of the Google Android website.

```
http://dl.google.com/android/ndk/android-ndk-r7b-linux-x86.tar.bz2
```

The Android NDK is a tool for compiling the C-layer code of the client. The android-ndk file is downloaded in the Linux system environment.

Step 3 Decompress the NDK compiler.

Run the following command in the Linux CLI:

```
tar xjf android-ndk-r7b-linux-x86.tar.bz2 -C ~/workspace/NDK
```

Decompress **ndk-r7b** to the **~/workspace/NDK** directory.

Step 4 Configure the NDK compiler by running the following command in the Linux CLI:

```
vi ~/.bashrc
```

Add the following statement to the end of the file:

```
export NDK_HOME=~/workspace/NDK/android-ndk-r7b/
export PATH=$PATH:$NDK_HOME
vi ":wq"
```

Save the file and exit. Then run the following command to make the setting take effect:

```
source ~/.bashrc
```

Step 5 Copy **libHiTransPlayer_jni** and **envsetup_player.sh** to the **~/workspace** directory.

Step 6 Run the following command in the **workspace** directory in Linux command line mode:

```
./envsetup_player.sh
```

Step 7 Run the ./**ndk-build** command in the **libHiTransPlayer_jni** directory.

Then **libtest1_jni.so**, **libhi_ffmpeg.so**, and **libandroid4.4_jni.so** are generated in the **~/workspace/libHiTransPlayer_jni/libs/armeabi-v7a** directory.

----End

Compiling the Client Software in the HiTransPlayer Directory

To compile HiTransPlayer, perform the following steps:

Step 1 Configure Eclipse and Android SDK in Windows.

The client software is developed based on Eclipse 4.2.0 and ADT 21.0.0. Download Eclipse 4.2.0, ADT 21.0.0, and Google Android SDK by yourself.

1. Install Eclipse 4.2.0, ADT 21.0.0, and Google Android SDK.
2. Open Eclipse.
3. Set the path to the decompressed Android SDK by choosing **windows** > **Preference** > **Android** > **SDK Location** of the eclipse.
4. Choose **help > Install New Software >Add**. Enter **Android Plugin** in **name** and click **Archive**. Then add the ADT package and click **OK**.

Step 2 Copy **HiTransPlayer** to Windows.

Step 3 Copy the **libs** directory to the **HiTransPlayer** directory, and ensure that **libtest1_jni.so**, **libhi_ffmpeg.so**, and **libandroid4.4_jni.so** compiled from **libHiTransPlayer_jni** are in the **HiTransPlayer/libs/armeabi** directory.

Step 4 Import a project and generate an APK file.

1. Open eclipse and choose **File > import**.
2. Choose **General > Existing Project into workspace**.
3. Click **Browser** and find the **HiTransPlayer** directory that is copied to Windows.
4. Click **Finish** and generate **EATAPlayer.apk.**

CAUTION

If "Unable to resolve target 'android-8'" is displayed, choose **Window** > **Preference** > **Android**, query the API version corresponding to the selected platform, and replace **8** in the **target** variable with the found version in the **project.properties** file.

Step 5 Install this client in the Android mobile phone or pad.

The client supports Android2.3 or later.

----End

CAUTION

As the EATAPlayer client is compatible with Android 2.3 and later versions, the Java JDK V1.6 is required. Before using Eclipse, choose **Window** > **Preference** >**Java** > **Compiler** in the Eclipse menu, and set the JDK version to 1.6. Ensure that the JDK 1.6 has been installed in the computer that runs Eclipse.

Client Usage

To set the client running environment, perform the following steps:

Step 1 Ensure that binary sample programs related to HiTranscoder are uploaded to the board and are running.

Step 2 Connect the TV set to the development board by using a video/audio cable and turn on the TV set and select the correct AV input modes.

Step 3 Confirm that an IP address is allocated for the board.

Step 4 Connect the hand-hold device to the network by using the Wi-Fi and ensure that the hand-hold device and the board communicate with each other, and the playing software supporting the RTSP/HTTP is installed on the hand-hold device.

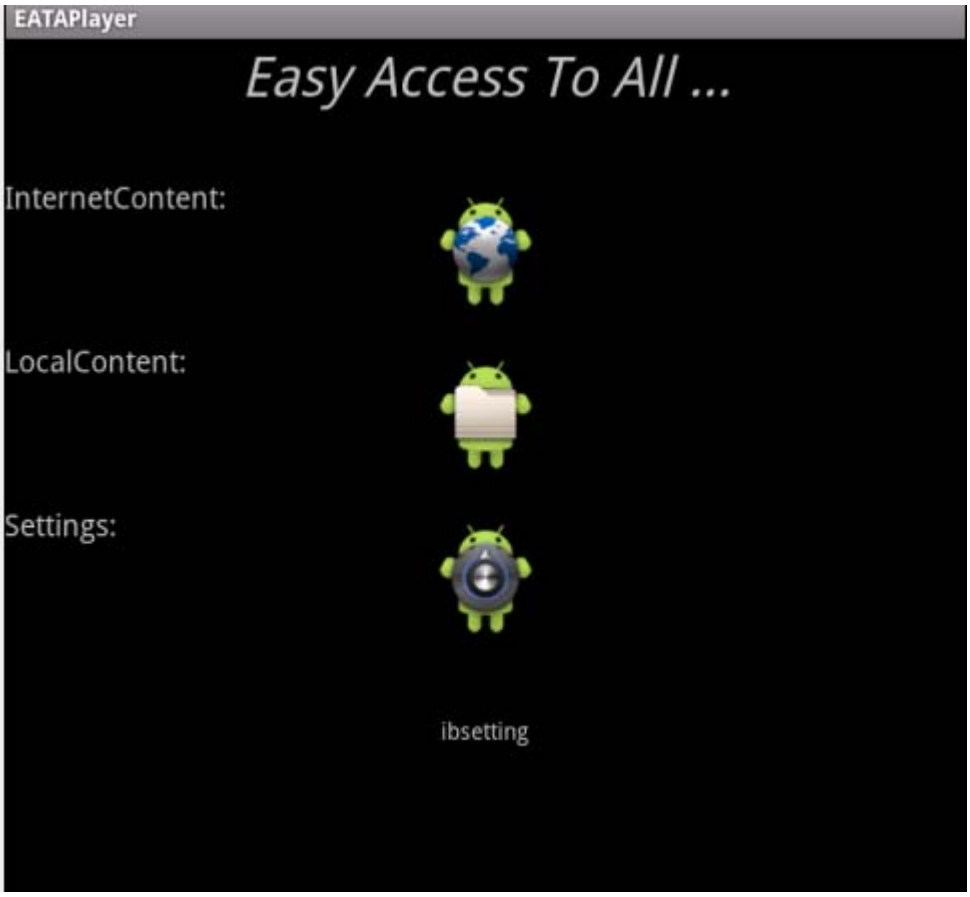
The PC connects to the LAN with a cable to ensure that the PC and the board can communicate with each other. Ensure that the playing software supporting RTSP/HTTP is installed on the PC.

Step 5 Play transcoded data by using the playing software. The following uses the low-delay client as an example:

- The IP address is the IP address of the board.
- Ensure that the **EATAPlayer.apk** has been installed on the hand-hold device.
- The port is the protocol port, which is set by users in the released sample code of the server.

The port ID must be unique and is 4098 in this example.

Figure 7-11 Low delay client of the HiTranscoder



Enter the URL **rtsp://Ip:Port/hisi** to access the server of the board after entering LocalContent. The low-delay client supports connection over only the RTSP protocol.

Step 6 The HiTranscoder also supports other protocols and encapsulation formats.

- RTSP:

  Plays rtsp://IP:port/, which cannot be suspended.
- HTTP:
  - TS encapsulation format: supports the playback of **http://IP:port/XX.ts**.
  - Flv encapsulation format: supports **http://IP:port/XX.flv**.

    XX indicates two or three characters.

    You can play audio and videos by using other players.

----End

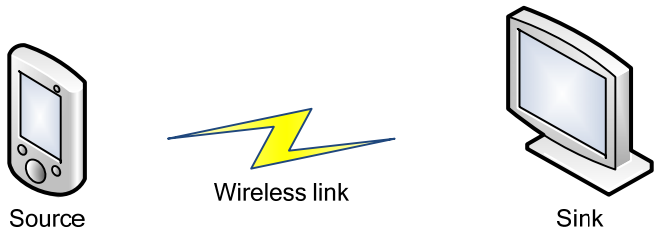
8 HiMiracast

8.1 Overview

8.1.1 Miracast Networking

As shown in Figure 8-1, the Miracast audio/video data source end encodes and transmits audio/video data, and the sink end (Miracast service RX end) receives and decodes audio/video streams transmitted from the source end and transmits decoded streams to the monitor for display. Currently Miracast is implemented by the sink end device.

Figure 8-1 Miracast networking



8.1.2 Important Concepts

[Wi-Fi Direct]

In October 2010, the Wi-Fi Alliance released the Wi-Fi Direct White Paper, which describes basic information, features, and functions of the Wi-Fi Direct technology. The Wi-Fi Direct standard enables devices in the wireless network to connect easily with each other without a wireless router. Compared with the Bluetooth technology, this standard enables point-to-point interconnection between wireless devices, provides a much higher transfer speed, and allows much longer transfer distance.

[Wi-Fi display]

Based on the Wi-Fi Direct standard, the Wi-Fi display technology implements reliable and P2P HD video and audio stream transfer between wireless devices and HD monitors. By using this technology, you can mirror the audio/video content from a mobile device to a large screen, enabling reliable audio/video transfer between any devices anywhere at any time.

[Miracast]

Miracast is a Wi-Fi certified Miracast™ project launched on September 19, 2012 by the Wi-Fi Alliance. It is a wireless display standard formulated by the Wi-Fi Alliance. Based on the Wi-Fi Direct protocol, Miracast allows data to be transmitted from one device to another without a router.

[HDCP]

High-bandwidth digital content protection (HDCP) is used to ensure the security of audio/video data during the transfer from the HDCP TX device to the HDCP RX device. The HDCP technology is divided into two types based on the application field:

- HDCP technology related to interfaces. It is used to protect audio/video data during HDMI transfer, and the current version is HDCP 1.4.
- HDCP technology unrelated to interfaces. It is used to protect audio/video data during wireless transfer, and the current version is HDCP 2.2.

In the HDCP ecosystem, there are three types of devices:

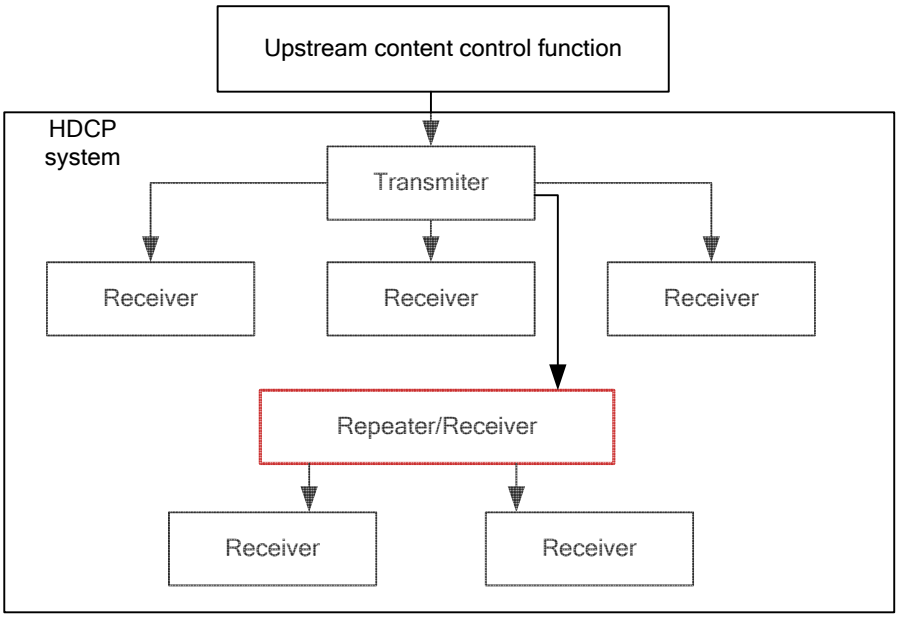
- Transmitter: Transmits audio/video data to the repeater or receiver.
- Repeater: Receives data from the transmitter and forwards the data to the downlink receiver or display device (such as the television). It is a special receiver.
- Receiver: Receives and displays data.

Table 8-1 HDCP devices

HDCP Device	Description	Scenario
Transmitter	Encrypts data to be transmitted.	Typically a mobile phone that supports HDCP is used as the transmitter. It encrypts audio/video data after being connected successfully to the network and transmits the data by using protocols such as the TCP, UDP, or RST.
Repeater	Receives and forwards data.	The STB acts as the repeater. It decrypts and then encrypts the received audio/video data and transmits the data to another STB or receiver.
Receiver	Decrypts data.	The display device functions as the receiver. It receives and decrypts the encrypted audio/video data from the transmitter (mobile phone) for display.

The STB is a forwarding device, but it can also be considered as a special receiver. HDCP 2.2 provides three basic functions: HDCP transmitter, repeater, and receiver. The current HiSilicon Android platform supports the HDCP 2.2 receiver.

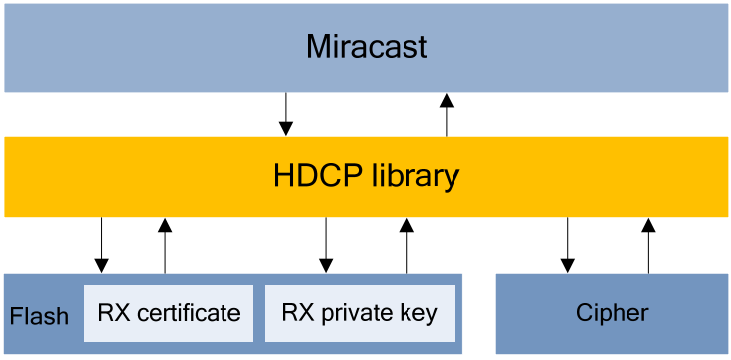
Figure 8-2 HDCP system topology



8.1.3 Function Implementation

Miracast transmits the received encrypted data to the HDCP library, and the HDCP library decrypts the data and returns the data to Miracast. See Figure 8-3.

Figure 8-3 Logic block diagram



The working process of the entire HDCP library complies with the HDCP 2.2 standards.

NOTE

You can download the HDCP 2.2 standards from the website of DCP LLC (official organization of HDCP) http://www.digital-cp.com/.

8.2 Miracast HDCP Mode Configuration

Miracast supports the following HDCP modes:

- SW HDCP mode
- HW HDCP mode
- Non-HDCP mode

Currently the non-HDCP mode is used in full compilation. Audio/Video streams that support HDCP cannot be parsed in this mode.

The HW HDCP mode is developed by HiSilicon and Discretix. It depends on the libraries of Descretix. It is not our mainstream solution and is not used in normal cases.

The SW HDCP mode is our mainstream solution. If you want to use the SW HDCP mode, you need to burn the HDCP key. For details about the development and production in SW HDCP mode, see section 8.5 "Development and Production of SW HDCP."

You can select a mode for compilation as required. The configuration procedure is as follows:

Step 1 Open **hdcp.xml** in **device/hisilicon/bigfish/hidolphin/component/miracast/android/packages/apps/Miracast/res/raw**.

Step 2 Select a mode as required.

- To select the non-HDCP mode, modify the file as follows:

```
<miracast>
    <name>none hdcp</name>
    <mode>1</mode>
</miracast>
```

- To select the HW HDCP mode, modify the file as follows:

```
<miracast>
    <name>HW hdcp</name>
    <mode>2</mode>
</miracast>
```

- To select the SW HDCP mode, modify the file as follows:

```
<miracast>
    <name>SW hdcp</name>
    <mode>3</mode>
</miracast>
```

Step 3 Perform full compilation according to section 2.3 "Development Compilation", and burn images to the board as required after compilation.

----End

8.3 Development Guide

8.3.1 Using the Proc

The proc information helps developers view the current Miracast state information and know about the current running status, and provides the **echo** command for setting the current device parameters.

8.3.1.1 Checking the Current Status by Running the cat Command

To check the current status by running the **cat** command, perform the following steps:

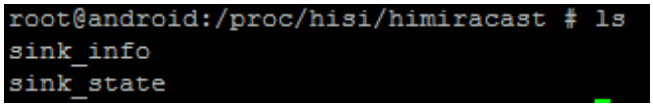
Step 1 Initiate a connection request on the source end and start screen mirroring.

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

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

Step 3 View the number of entries, as shown in Figure 8-4.

Figure 8-4 Viewing the number of entries



Step 4 View the content of the **sink_info** entry, as shown in Figure 8-5.

Figure 8-5 sink_info entry

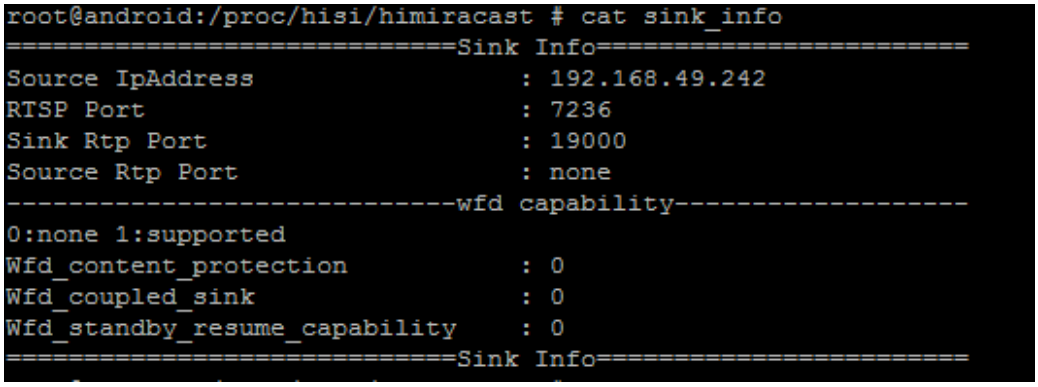


Table 8-2 describes the parameters.

Table 8-2 sink_info parameters

Parameter	Description
Source IpAddress	IP address of the source end

Parameter	Description
RTSP Port	ID of the port for the RTSP connection on the source end
Sink Rtp Port	RTP port ID of the sink end
Source Rtp Port	RTP port ID of the source end **None** indicates that the port ID is not obtained.
wfd_content_protection	Whether HDCP is supported 0: not supported 1: supported
wfd_coupled_sink	Whether the secondary sink is supported 0: not supported 1: supported
wfd_standby_resume_capability	Whether the standby mode is supported 0: not supported 1: supported

Step 5 View the content of the **sink_state** entry, as shown in Figure 8-6.

Figure 8-6 sink_state entry

```
root@android:/proc/hisi/himiracast # cat sink_state
============================Sink State=========================
0:UNDEFINED 1:CONNECTING 2:CONNECTED 3:PAUSED 4:PLAYING
Sink state                      : 4
-------------------------------------------------------------
stat lost pkt time              : 1
lost packets num                : 0
-------------------------------------------------------------
0:normal 1:simple
vdec output mode                : 1
============================Sink State=========================
```

Table 8-3 describes the parameters.

Table 8-3 sink_state parameters

Parameter	Description
Sink state	Sink state 0: not initialized 1: connecting 2: connected 3: paused 4: mirroring the screen
Stat lost pkt time	Interval (in second) for counting the number of lost packets
Lost packets num	Number of packets lost during the interval specified by **Stat lost pkt time**
Vdec output mode	VDEC mode 0: quick output mode 1: intermittence mode

----End

8.3.1.2 Setting Parameters by Running the echo Command

- Setting the keep alive time of the sink end during screen mirroring

  Preset state: mirroring

  Run **echo alivetime=X > /proc/hisi/himiracast/sink_info** over the serial port, as shown in Figure 8-7.

Figure 8-7 Setting the keep alive time of the sink device

```
root@android:/ # echo alivetime=6 >/proc/hisi/himiracast/sink_info
Arg[0] = alivetime=6
CMD_SINK_KEEPALIVE_TIMEOUT
```

The default keep alive time of the sink end is 5 seconds. The value range is 5 to 30 seconds. If the value is beyond the range, a message is displayed, asking you to enter another value. See Figure 8-8.

Figure 8-8 Information displayed when the keep alive time is beyond the range

```
root@android:/ # echo alivetime=1 >/proc/hisi/himiracast/sink_info
Arg[0] = alivetime=1
too small, please input number between 5 - 30
root@android:/ #
root@android:/ # echo alivetime=31 >/proc/hisi/himiracast/sink_info
Arg[0] = alivetime=31
too big, please input number between 5 - 30
root@android:/ #
```

- Setting the interval for counting the lost RTP packets on the sink end during mirroring

  Preset state: mirroring

  Run **echo lostpktime=X > /proc/hisi/himiracast/sink_info** over the serial port, as shown in Figure 8-9.

Figure 8-9 Setting the interval for counting the lost RTP packets on the sink end

```
root@android:/ # echo lostpkttime=3 >/proc/hisi/himiracast/sink_info
Arg[0] = lostpkttime=3
CMD_SINK_STAT_LOSTPKT_TIMEOUT
```

The default interval for counting the number of lost packets on the sink end is 1 second. The value range is 1 to 10 seconds. If the value is beyond the range, a message is displayed, asking you to enter another value. See Figure 8-10.

Figure 8-10 Information displayed when the interval for counting the lost RTP packets is beyond the range

```
root@android:/ # echo lostpkttime=0 >/proc/hisi/himiracast/sink_info
Arg[0] = lostpkttime=0
too small, please input number between 1 - 10
root@android:/ #
root@android:/ # echo lostpkttime=11 >/proc/hisi/himiracast/sink_info
Arg[0] = lostpkttime=11
too big, please input number between 1 - 10
```

- Setting the VDEC output mode of the sink end during mirroring

  Preset state: mirroring

  Run **echo mode=X > /proc/hisi/himiracast/sink_info** over the serial port. **mode=0** indicates the fixed normal mode, that is, the intermittence mode. **mode=1** indicates the fixed simple mode. **mode=2** indicates that the VDEC information is automatically adjusted based on the packet loss rate. See Figure 8-11.

Figure 8-11 Setting the VDEC output mode on the sink end

```
root@android:/ # echo mode=2 >/proc/hisi/himiracast/sink_info
Arg[0] = mode=2
CMD_SINK_VDEC_MODE_MIX_NORMAL_SIMPLE
```

8.3.2 Changing the Device Name

After the Miracast application is started, the device name displayed on the UI is **Android_***xxxx* (*xxxx* are four random digits). The device name obtained from the source end is also **Android_***xxxx*. To change the device name, perform the following steps:

Step 1 Open **WfdService.java** in **/device/hisilicon/bigfish/hidolphin/component/miracast/android/packages/apps/Miracast/src/com/hisilicon/miracast/service**.

Step 2 Call mWfdBusiness.setDeviceName(newDeviceName). Search for the key character string **WIFI_P2P_STATE_ENABLED**. The code is as follows:

```
if (isEnabled)
{
LogUtil.i("WIFI_P2P_STATE_ENABLED");
hideWaitingDialog();
mWfdBusiness.setDeviceName(newDeviceName);
}
else
{
LogUtil.i("WIFI_P2P_STATE_DISABLED");
}
```

Step 3 Recompile **Miracast.apk**.

Run **mm –B** in **/device/hisilicon/bigfish/hidolphin/component/miracast/android/packages/apps/Miracast** to generate**Miracast.apk** in **out/target/product/Hi3716CV200/system/app/**.

Step 4 Push **Miracast.apk** to the board.

Assume that the IP address of the STB is 10.161.179.2, and **Miracast.apk** is stored in **D:** of the PC, run the following commands:

```
adb connect 10.161.179.2
adb remount
adb push D:\Miracast.apk system/app
```

----End

8.4 Debugging Guide

8.4.1 Overview

The Miracast connection process involves the functional services of multiple modules, including the Wi-Fi driver, Wpa_supplicant, framework, Wi-Fi Display protocol stack, and application. An error that occurs in any step of the process may result in connection exceptions, and identifying connection issues is quite difficult and time-consuming. This section provides guidance for identifying Miracast connection issues efficiently and accurately.

The causes of the connection issues are summarized as follows:

- Adaptation of the Wi-Fi driver

  Some customers use their own drivers instead of the Wi-Fi driver provided by HiSilicon, and they do not perform many tests on the Miracast service. Therefore, the Wi-Fi driver may not be properly adapted, which results in connection issues.
- Adaptation of Wpa_supplicant

  The connection of the Miracast service is highly dependent on Wi-Fi Direct. Wpa_supplicant is the process control center of Wi-Fi module and is complicated. If it is not adapted properly, connection exceptions may occur.
- Compatibility of the Wi-Fi Display protocol

The mobile phones or pad devices that support the Miracast service have increased from one or two to several dozen or even a hundred since the Miracast service emerged. The implementation of the Wi-Fi Display code varies according to the vendor, which results in compatibility issues.

- Mobile phone issues

  The connection may also fail due to the faults of the mobile phones.

- Network environment

  When the wireless network environment is complicated with large interference, group owner (GO) negotiation timeout may occur when the Wi-Fi Direct connection is being established, resulting in a connection failure.

- WPS authentication failure

  During the Miracast connection process, if another source device (a mobile phone or pad) in the network environment attempts to connect to the Miracast service or similar WPS service (for example, mobile phone B attempts to connect to STB B while mobile phone A is connecting to STB A), the WPS authentication will fail, which leads to a connection failure.

- Misoperations

  Some users are unfamiliar with the Miracast service. They may confuse Wi-Fi Direct with Wi-Fi Display and initiate connection on the Wi-Fi Direct UI.

8.4.2 Preparations

Preparing the Environment

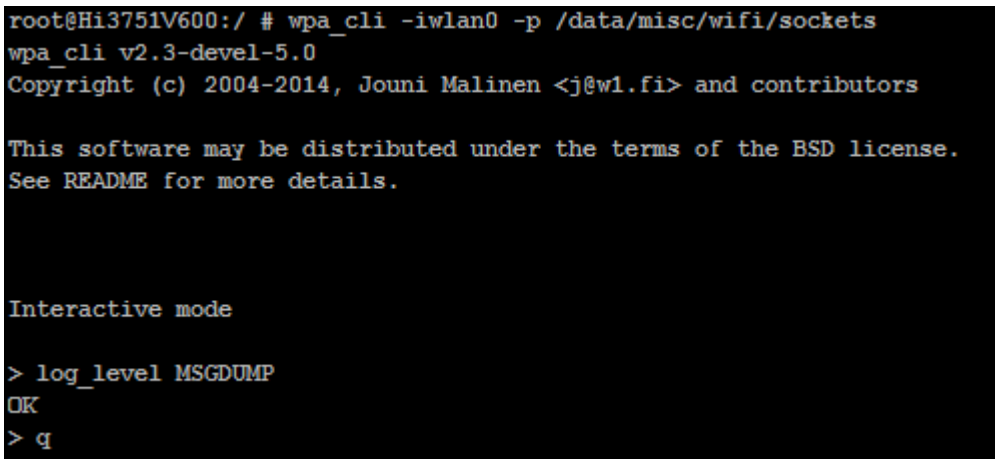
The Miracast service is based on the Wi-Fi Direct protocol. Therefore, the customer's device must provide the Wi-Fi device that supports the Wi-Fi Direct protocol and is in the version compatibility list released by HiSilicon.

Capturing Logs

To capture the logs correctly, note the following:

- Start the logcat before connection.
- Clear the logs by running **logcat –c** before capturing logs.
- Record the time while capturing the logs by running **logcat –v time**.
- If there are many repeated and meaningless logs, shield them by running **logcat –s XXX:F *:v**. *XXX* indicates the flag of many repeated logs.
- Enable the log switch of Wpa_supplicant before capturing the Wi-Fi Direct process logs. See Figure 8-12.

Figure 8-12 Enabling the Wpa_supplicant logs



Capturing Empty Packets

As the Miracast involves the interaction between the source and sink, it is difficult to determine which end is faulty. Therefore, sniffer packets need to be captured and analyzed. To capture sniffer packets, prepare the following items:

- Hardware: packet capturing network adapter (the packet capturing network adapter of the RT5572 is recommended)
- Software: network adapter driver and packet capturing software omnipeek (omnipeek 7.5 is recommended)

8.4.3 Fault Identification

The Miracast connection process consists of two parts:

- Wi-Fi Direct connection
- Wi-Fi Display connection

The Wi-Fi Display connection process starts only after the Wi-Fi Direct connection is successfully established.

After logs are captured, the cause of the connection issue can be identified based on the logs. The following describes some typical scenarios.

Wi-Fi Direct Connection Process and Key Logs

The establishment of the Wi-Fi Direct connection involves the interaction of more than 20 messages. The connection fails if an error occurs in any message. The establishment of the Wi-Fi Direct connection can be divided into three parts for analysis:

- GO negotiation process

  Start message:

  ```
  p2p0: P2P: Received GO Negotiation Request from
  02:11:22:18:9e:74(freq=2412)
  ```

  02:11:22:18:9e:74 is the MAC address for the source end.

  Success message:

```
p2p0: P2P: GO Negotiation with 02:11:22:18:9e:74 completed (local end
will be GO)
p2p0: P2P-GO-NEG-SUCCESS
```

The preceding logs indicate that GO negotiation is successful. **local end will be GO** indicates that the sink end is the GO. If the content in the parentheses is **peer end will be GO**, the source end is the GO.

- Group formation process

  After GO negotiation is successful, the GO creates a group, and group formation starts by PBC authentication.

  Start message:

  ```
  p2p0: CTRL-EVENT-EAP-STARTED
  ```

  The preceding log indicates that the WPS process starts.

  Success message:

  ```
  p2p0: P2P: Group Formation completed successfully with
  02:11:22:18:9e:74
  p2p0: P2P-GROUP-FORMATION-SUCCESS
  ```

- Connection process

  After group formation is successful, the group client (GC) connects to the GO using the negotiated key.

  Key log 1:

  ```
  I/com.hisilicon.miracast.service.WfdService$WfdReceiver( 2953):
  onReceive L1216 ---- Wi-Fi Direct connected ----
  ```

  Key log 2:

  ```
  D/WifiDisplaySink( 2953): init L116
  ```

  - If any of the preceding key logs is displayed during a connection process, the Wi-Fi Direct connection is successfully established.
  - If neither of the preceding key logs is displayed, the Wi-Fi Direct connection fails to be established.
  - If the Wi-Fi Direct connection fails to be established, you can check which part fails based on the preceding logs to narrow down the range.

Wi-Fi Display Connection Success Indicator

Key logs:

```
01-01 00:04:18.405 D/WifiDisplaySink( 2953): sendM7 L1369 request = 'PLAY
rtsp://192.168.49.1:7236/wfd1.0/streamid=0 RTSP/1.0
01-01 00:04:18.405 D/WifiDisplaySink( 2953): Date: Wed, 01 Jan 2014
00:04:18 +0000
01-01 00:04:18.405 D/WifiDisplaySink( 2953): CSeq: 3
01-01 00:04:18.405 D/WifiDisplaySink( 2953): Session: 00000060
01-01 00:04:18.405 D/WifiDisplaySink( 2953):
01-01 00:04:18.405 D/WifiDisplaySink( 2953): '
01-01 00:04:18.413 V/WifiDisplaySink( 2953): onReceiveClientData L1014
session 1 received 'RTSP/1.0 200 OK
01-01 00:04:18.413 V/WifiDisplaySink( 2953): cseq: 3
01-01 00:04:18.413 V/WifiDisplaySink( 2953): date: Mon, Mar 30 2015
```

```
03:45:16 GMT
01-01 00:04:18.413 V/WifiDisplaySink( 2953):
01-01 00:04:18.413 V/WifiDisplaySink( 2953):
```

As shown in the preceding key logs, the M7 message is the last message during the negotiation process of the Wi-Fi Display connection. If the sendM7 message is displayed in the logs, the Wi-Fi Display connection is successfully established.

HDCP Negotiation Failure

Some mobile phones have HDCP compatibility issues. After the Wi-Fi Display connection is established, a teardown message is sent from the mobile phone, which causes the connection failure.

The following is an example of the teardown message received by the board:

```
01-01 04:27:26.426 V/WifiDisplaySink( 2953): onReceiveClientData L1014
session 1 received 'SET_PARAMETER rtsp://localhost/wfd1.0 RTSP/1.0
01-01 04:27:26.426 V/WifiDisplaySink( 2953): content-length: 30
01-01 04:27:26.426 V/WifiDisplaySink( 2953): content-type:
text/parameters
01-01 04:27:26.426 V/WifiDisplaySink( 2953): cseq: 4
01-01 04:27:26.426 V/WifiDisplaySink( 2953):
01-01 04:27:26.426 V/WifiDisplaySink( 2953): wfd_trigger_method: TEARDOW
01-01 04:27:26.426 V/WifiDisplaySink( 2953):
```

Mobile Phone Exceptions

During the Wi-Fi Display connection process, the mobile phone is the RTSP server end. After it initiates a connection invitation, an RTSP server is started and enters the listening state, waiting for the TCP connection from the STB end. However, for some mobile phones, the RTSP server may fail to start during connection, which results in the failure of the TCP connection requested by the STB end. Therefore, the Wi-Fi Display connection fails. In this scenario, the STB logs are as follows:

```
01-02 09:31:29.607 D/WifiDisplaySink( 2549): init L115
01-02 09:31:29.607 D/WifiDisplaySink( 2549): sinkProcCreate L283
01-02 09:31:29.608 V/WifiDisplaySink( 2549): initHDCPRptOrRcv L374
01-02 09:31:46.628 D/WifiDisplaySink( 2549): initHDCPRptOrRcv Success
initHDCPRptOrRcv L384
01-02 09:31:46.628 D/WifiDisplaySink( 2549): platformCheck L255
01-02 09:31:46.628 D/WifiDisplaySink( 2549): start L232
01-02 09:31:46.631 E/WifiDisplaySink( 2549): onMessageReceived L468 RTSP
error
01-02 09:31:46.631 E/WifiDisplaySink( 2549): onMessageReceived L483 An
error occurred in session 1 (-111, 'Connection failed/Connection
refused').
```

If the preceding socket issue occurs during connection, you can disconnect the mobile phone from all connected wireless routers, restart the mobile phone, and initiate the connection again.

User Misoperations

If a user initiates the Miracast connection in the Wi-Fi Direct option, the connection failure logs are as follows:

```
01-01 04:01:17.502 D/WifiDisplaySink(16737): init L11601-01 04:01:17.502
V/NetworkSession(16737): ANetworkSession L663
01-01 04:01:17.503 V/TunnelRenderer(16737): TunnelRenderer L61
01-01 04:01:17.503 V/RTPSink (16737): RTPSink L296
01-01 04:01:17.503 D/RTPSink (16737): start L302
01-01 04:01:17.503 V/NetworkSession(16737): ANetworkSession L663
01-01 04:01:17.503 V/NetworkSession(16737): ANetworkSession L663
01-01 04:01:17.503 D/WifiDisplaySink(16737): sinkProcCreate L284
01-01 04:01:17.503 D/(16737): Sink_PROC_CreateMode L212
01-01 04:01:17.504 D/WifiDisplaySink(16737): platformCheck L256
01-01 04:01:17.504 D/HiSinkJNI(16737):
Java_com_hisilicon_miracast_ndk_SinkNative_startSink 71
01-01 04:01:17.504 D/WifiDisplaySink(16737): start L233
01-01 04:01:17.509 I/NetworkSession(16737): createClientOrServer L990
connecting socket 57 to 192.168.49.1:7236
01-01 04:01:17.581 E/NetworkSession(16737): threadLoop L1361 writeMore on
socket 57 failed w/ error -111 (Connection refused)
01-01 04:01:17.581 E/WifiDisplaySink(16737): onMessageReceived L469 RTSP
error
01-01 04:01:17.581 E/WifiDisplaySink(16737): onMessageReceived L484 An
error occurred in session 1 (-111, 'Connection failed/Connection
refused').
01-01 04:01:17.581 I/WifiDisplaySink(16737): onMessageReceived L488 Lost
control connection.
04:01:17.581 V/NetworkSession(16737): ~Session L211 Session 1 gone
```

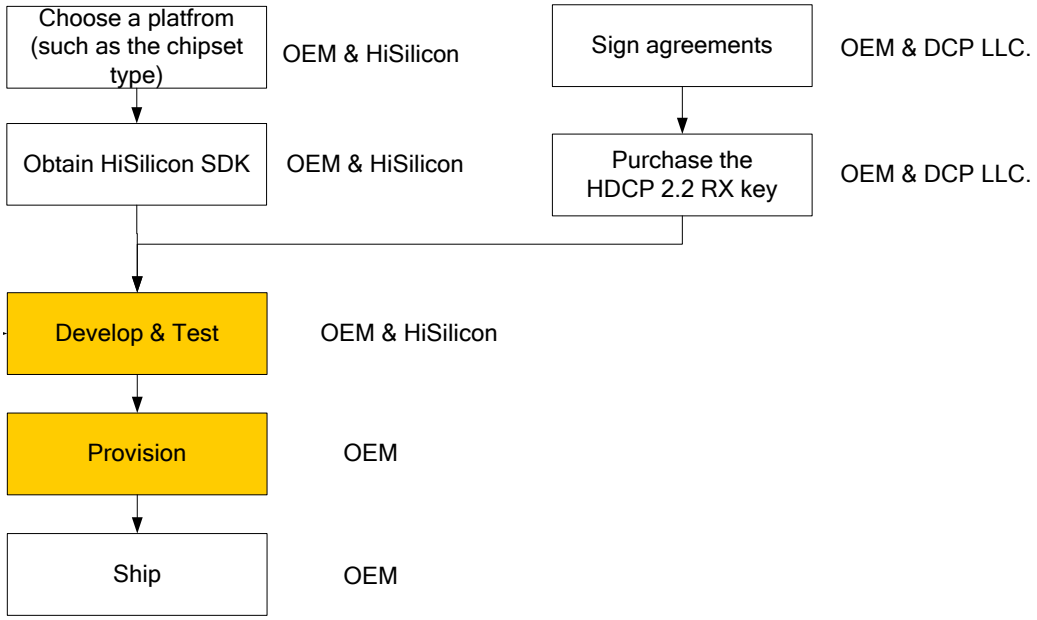
This issue occurs due to misoperations. For details about how to use the Miracast service correctly, see the Miracast section in the *Android Solution User Guide*.

8.5 Development and Production of SW HDCP

8.5.1 SW HDCP Development Process

Figure 8-13 shows the overall development process.

Figure 8-13 SW HDCP development process



The process is as follows:

Step 1 The OEM vendor requests the HDCP 2.2 RX key from the DCP LLC.

Step 2 The OEM vendor determines the chip and SDK version to be used and obtains the SDK from HiSilicon.

Step 3 After obtaining the HDCP 2.2 RX key and the HiSlicon SDK that supports HDCP 2.2, the OEM vendor customizes the production tool or application for burning the HDCP key based on the production environment, formulates the output protection solution, and performs tests.

Step 4 The OEM vendor produces and ships products.

----End

In step 3, the OEM vendor needs to customize the output protection solution. In the output protection solution, the HDMI HDCP feature needs to be enabled. In this case, HDCP 1.4 must be enabled.

NOTE

The HDMI supports HDCP 1.4. For details, see the HDCP Key User Guide or consult the FAE.

Therefore, before the development and test of SW HDCP 2.2, you need to purchase the HDCP 2.2 and HDCP 1.4 keys from the DCP LLC.

The current HiSilicon SDK contains the HDCP 2.2 library (**libhihdcp.so**). However, you still need to burn the HDCP 2.2 key. For details, see the following section.

8.5.2 Burning the HDCP Key During Factory Production

In the SW HDCP 2.2 solution, the HDCP 2.2 key is encrypted by using STB_ROOT_KEY and then burnt to the flash memory. Therefore, before burning the HDCP key, the OEM vendor needs to burn STB_ROOT_KEY; otherwise, the HDCP library automatically

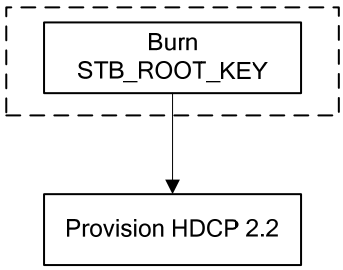
generates and burns a random STB_ROOT_KEY. The entire HDCP key is burnt to the end of the deviceinfo partition and occupies 4 KB space. If this memory space is damaged, reburn the HDCP key.

NOTE

STB_ROOT_KEY can be generated and managed by the OEM vendor. After being burnt to the OTP, STB_ROOT_KEY is locked. The CPU cannot read it from the OTP, which ensures data security.

Figure 8-14 shows the factory production process.

Figure 8-14 Factory production process



The operation in the dashed frame is optional for the OEM vendor. If the OEM vendor needs to manage STB_ROOT_KEY, it must burn STB_ROOT_KEY; otherwise, the HDCP library automatically generates and burns random STB_ROOT_KEY. For details about how to burn STB_ROOT_KEY, see the following samples:

- device/hisilicon/bigfish/sdk/sample/otp/sample_otp_setstbrootkey.c
- device/hisilicon/bigfish/sdk/sample/otp/sample_otp_lockstbrootkey.c

To burn the HDCP key, perform the following steps (the burning library **libhihdcp_tool.a** and APIs are provided in the SDK):

Step 1 Split the HDCP 2.2 key to obtain a single plaintext key by using the packet split tool in **device/hisilicon/bigfish/sdk/tools/windows/HdcpTools/HDCP_Key_Huawei_STB.zip**.

Step 2 Burn STB_ROOT_KEY in the factory production APK. Skip this step if the OEM vendor does not need to manage STB_ROOT_KEY.

Step 3 Burn the APIs and libraries to the factory production APK by calling HI_S32 HI_HDCP2_SetHDCPKey(HI_U8 *pKey, HI_U32 u32Length).

The header file is **device/hisilicon/bigfish/security/hdcp/include/HI_ADP_HDCP_Receiver.h**.

The demonstration sample is as follows:

```
device/hisilicon/bigfish/security/hdcp/sample/sample_hdcp_burn.c
```

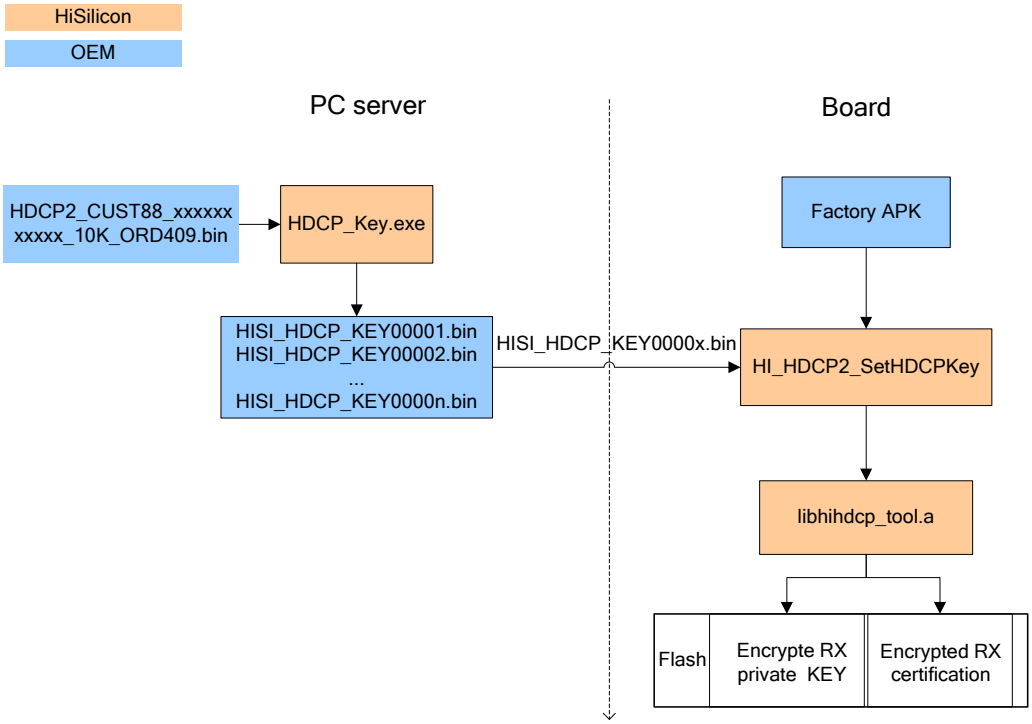
During manual debugging, download the compiled sample_hdcp_burn and split the HDCP key **HISI_HDCP_KEYxxxxx.bin** to the **/data/** directory, and run the following commands:

```
#./chmod 777 sample_hdcp_burn
#./sample_hdcp_burn HISI_HDCP_KEYxxxxx.bin
```

Step 4 Download the key from the server to the board during production, and write the key to the board by using the production application.

Figure 8-15 shows the burning process if you do not need to burn STB_ROOT_KEY.

Figure 8-15 Burning the HDCP key



For details about how to use Miracast after burning the HDCP key, see the Miracast section in the *Android Solution User Guide*.

----End

9 HiKaraoke

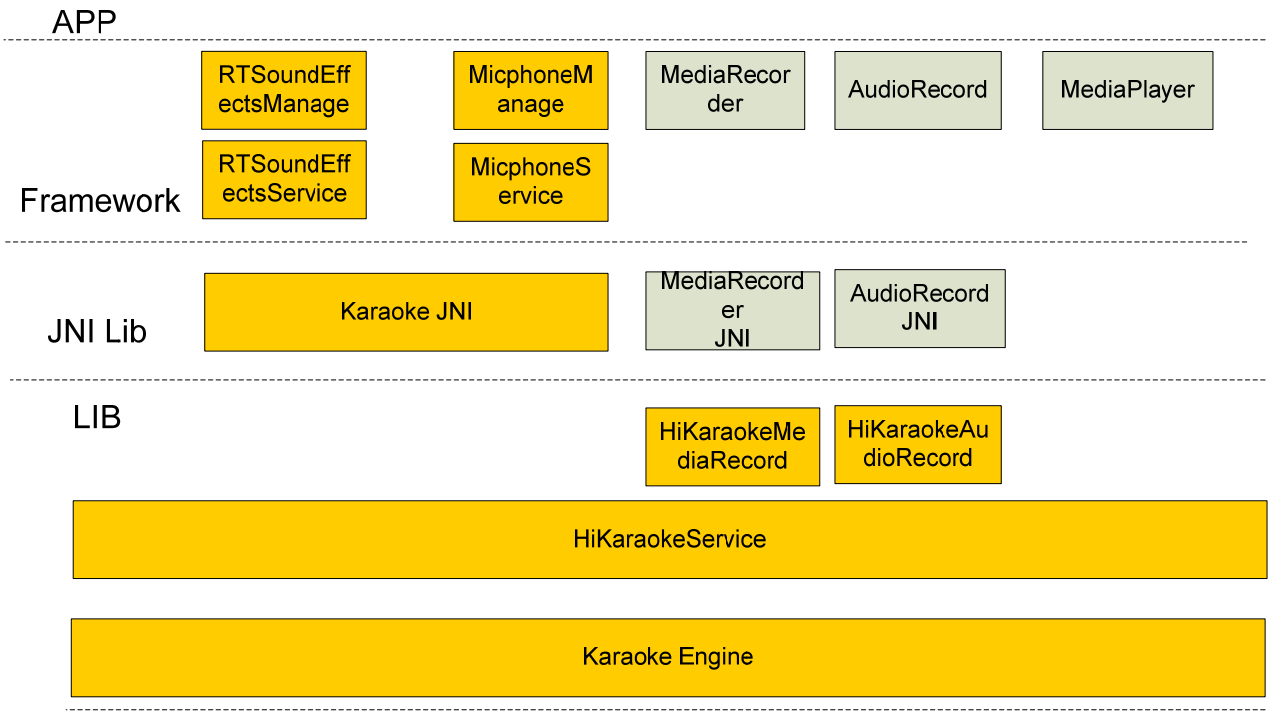
9.1 Overview

The HiKaraoke is a Karaoke solution implemented by HiSilicon. This solution implements the network or local playback accompaniment and outputs the audio data after audio mixing and audio effect processing of the collected microphone data to achieve the perfect Karaoke effect.

Figure 9-1 shows the framework of the HiKaraoke.

- The modules in yellow are new modules.
- The modules in off-white are modified based on the Android native framework.
- APP is the application layer. You can develop your own Karaoke applications.
- The framework layer contains two services and three interface classes. It provides APIs for the application layer.
- The JNI lib layer is a transitional layer between the Java layer and C++ layer for Karaoke APIs.
- The LIB layer is the implementation layer. It contains the HiKaraoke service and Karaoke engine. The engine is stored in the **sdk/source/component/karaoke** directory as a component library, and other parts exist as the source code.

Figure 9-1 HiKaraoke framework



9.2 Features

9.2.1 Main Functions

The HiKaraoke mainly provides the following functions:

- Real-time audio mixing

  The human voice (microphone input) and accompaniment (MV or MP3) are mixed in real time for output.
- Low delay

  The delay from the microphone input to sound box output is less than 30 ms.
- Audio effect processing

  Three audio effects (recording studio, KTV, and vocal concert) are provided, which can be dynamically switched.
- Rating

  Upper-layer applications can obtain real-time microphone data for rating.
- Switching between the original vocal and accompaniment

  You can switch between the original vocal and accompaniment dynamically.
- Recording of mixed audio

  Upper-layer applications can obtain mixed audio data (AAC encoding) after audio effect processing for playback or sharing.

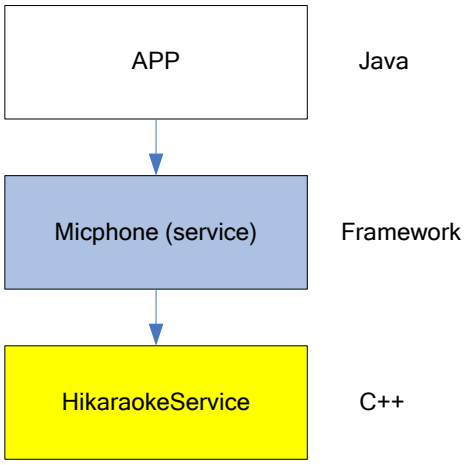
9.2.2 Function Implementation

9.2.2.1 Micphone Service

The Micphone service is an Android AIDL service for controlling the microphone, including enabling or disabling the microphone and adjusting the volume.

Figure 9-2 shows the control process of the Micphone service.

Figure 9-2 Micphone service control process

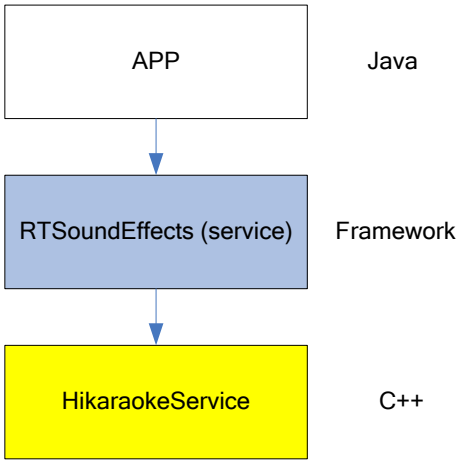


9.2.2.2 RTSoundEffects Service

The RTSoundEffects service is an Android AIDL service for controlling the audio effect, including configuring the recording studio, KTV, and vocal concert audio effects.

Figure 9-3 shows the control process of the RTSoundEffects service.

Figure 9-3 RTSoundEffects service control process



9.2.2.3 HiKaraokeService Service

The HiKaraokeService is an Android C++ service for connecting the upper-layer services and the Karaoke engine.

9.2.2.4 Karaoke Engine

The Karaoke engine is a library that implements functions such as microphone control, real-time audio mixing, low delay, audio effect processing, rating, and recording of mixed audio.

9.3 Development Guide

9.3.1 Using the Micphone Service

The Micphone service is loaded when the system is started. Applications can obtain the interfaces of the service by using getSystemService to control the microphone.

- Initialize and enable the microphone when the playback of an MP3 or MV file starts:

```
Micphone micphone = (Micphone) getSystemService("Micphone");
micphone.initial();
micphone.start();
```

- Set the microphone volume during singing:

```
micphone. setVolume (volume);
```

- Disable the microphone to release resources after singing is over:

```
micphone. stop ();
micphone. release ();
```

9.3.2 Using the RTSoundEffects Service

The RTSoundEffects service is loaded when the system is started. Applications can obtain the interfaces of the service by using getSystemService to switch the audio effect.

Switch the audio effect dynamically during singing:

```
RTSoundEffects rtSoundEffects = (RTSoundEffects)
getSystemService("RTSoundEffects");
rtSoundEffects.setReverbMode(RTSoundEffects. REVERB_MODE_STUDIO);
```

9.3.3 Switching Between the Original Vocal and Accompaniment

The MediaPlayer implements the switchChannel interface of the MediaPlayerEx class. You can switch between the original vocal and accompaniment by using this interface.

Switch between the original vocal and accompaniment during singing:

```
MediaPlayer mPlayer = new MediaPlayer();
if(mPlayer instanceof MediaPlayerEx){
((MediaPlayerEx) mPlayer). switchChannel (CHANNEL. LEFT);
}
```

9.3.4 Obtaining Microphone Data

The **CMCC_KARAOK_MIC** option is added to AudioSource. AudioRecord is created by using the **CMCC_KARAOK_MIC** option and microphone data is obtained by using the read interface.

- Create AudioRecord:

```
int minFramesize = AudioRecord.getMinBufferSize(44100,
AudioFormat.CHANNEL_CONFIGURATION_MONO,
AudioFormat.ENCODING_PCM_16BIT);
AudioRecord mAudioRecord = new
AudioRecord(AudioSource.CMCC_KARAOK_MIC, 44100,
AudioFormat.CHANNEL_CONFIGURATION_MONO,
AudioFormat.ENCODING_PCM_16BIT, minFramesize);
```

- Read the microphone data by using the read interface of AudioRecord.

9.3.5 Recording Mixed Audio

The MediaRecorder implements the pause and resume interfaces of the MediaRecorderEx class for controlling the recording of mixed audio. The **CMCC_KARAOK_SPEAKER** option is added to AudioSource. The MediaRecorder is created by using the **CMCC_KARAOK_SPEAKER** option for recording.

- Create the MediaRecorder and start recording:

```
MediaRecorder mMediaRecorder = new MediaRecorder();
mMediaRecorder.setAudioSource(HikaraokeUtil.KARAOKE_MEDIARECORD_SOURC
E);
mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.AAC_ADTS);
mMediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AAC);
mMediaRecorder.setAudioSamplingRate(48000);
mMediaRecorder.setAudioEncodingBitRate(64000);
mMediaRecorder.setAudioChannels(2);
mMediaRecorder.prepare();
mMediaRecorder.start();
```

- Pause recording:

```
if(mMediaRecorder instanceof MediaRecorderEx){
((MediaRecorderEx) mMediaRecorder). pause();
}
```

- Resume recording:

```
if(mMediaRecorder instanceof MediaRecorderEx){
((MediaRecorderEx) mMediaRecorder). resume();
}
```

9.3.6 Microphone Hot Swap

The system supports the microphone hot swap broadcast.

- When a USB microphone device is inserted, the system gives the com.cmcc.intent.action.MICPHONE_PLUG_IN broadcast.
- When a USB microphone device is removed, the system gives the com.cmcc.intent.action.MICPHONE_PLUG_OUT broadcast.

Applications can register these two broadcasts to monitor the status of the USB microphone.

9.3.7 Adapting the Remote Control

Functions such as switching between the original vocal and accompaniment and selecting the audio effect can be directly integrated into the menus of applications. If you want to use the shortcut key, you need to adapt the remote control of the platform to the application.

9.3.8 Testing Interfaces

The HiKaraokeDemo test application in the **device/hisilicon/bigfish/development/app** directory can be used to test the interfaces. You can also develop the program based on this demo.