



HMS

## Development Guide

Issue 05

Date 2015-12-30

**Copyright © HiSilicon Technologies Co., Ltd. 2015. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon Technologies Co., Ltd.

**Trademarks and Permissions**

 **HISILICON**, and other HiSilicon icons are trademarks of HiSilicon Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

**Notice**

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

## HiSilicon Technologies Co., Ltd.

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

Website:   <http://www.hisilicon.com>

Email:       [support@hisilicon.com](mailto:support@hisilicon.com)



# About This Document

## Purpose

This document describes the functions, common interfaces, working principles, and development processes of the modules for the HiSilicon HD STB processors, and precautions to be taken during development.

## Related Versions

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

Product Name	Version
Hi3798M	V1XX
Hi3798C	V1XX
Hi3798C	V2XX
Hi3796C	V1XX
Hi3796M	V1XX

## 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 05 (2015-12-30)**

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

Chapter5, Chapter7, Chapter17, Chapter19, Chapter21 and Chapter40 are modified.

## **Issue 04 (2015-05-30)**

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

Hi3798C V200 is supported.

## **Issue 03 (2015-04-07)**

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

Chapter 6 is modified.

## **Issue 02 (2014-11-20)**

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

Sections 14.4 and 15.4 are modified.

## **Issue 01 (2014-10-30)**

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

Hi3796M V100 is supported.

## **Issue 00B01 (2014-06-20)**

This issue is the first draft release.



# Contents

---

<b>About This Document.....</b>	<b>i</b>
<b>1 Overview.....</b>	<b>1</b>
1.1 Application Architecture .....	1
1.2 SDK Overview (Function Description).....	2
1.3 Function Conventions .....	2
1.3.1 Open/Close.....	2
1.3.2 Create/ Destroy .....	2
1.3.3 Handle.....	3
1.3.4 Attach /Detach.....	3
1.3.5 Get/Put.....	3
1.3.6 Acquire/Release .....	4



# Figures

<b>Figure 1-1</b> Typical application architecture of the MSP.....	1
<b>Figure 1-2</b> Typical procedure for processing data .....	5

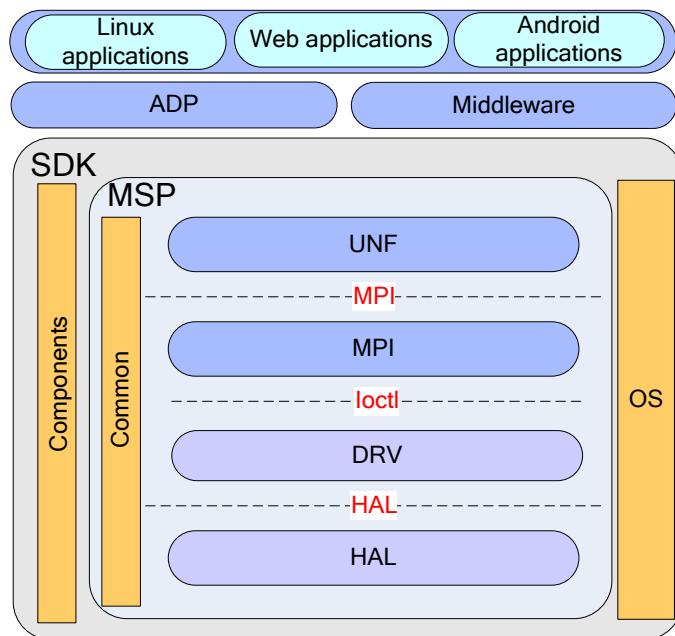


# 1 Overview

## 1.1 Application Architecture

The HiSilicon media solution platform (MSP) implements shielding and encapsulation of media, graphics, and peripheral modules in the HD STB solution processors and provides APIs for applications to support required functions. [Figure 1-1](#) shows the typical application architecture of the MSP.

**Figure 1-1** Typical application architecture of the MSP



The software architecture consists of the following four layers:

- UNF layer  
Unified application development APIs of the MSP
- MPI layer  
User-mode part of the module hardware implementation layer
- DRV layer



- Kernel-mode part of the module hardware implementation layer
- HAL layer
- Module hardware abstraction layer

## 1.2 SDK Overview (Function Description)

All modules of the MSP are classified into the following three categories based on the functions:

- Media processing
  - DEMUX, AVPLAY, SOUND, DISPLAY, VO, HDMI, PVR, VDEC, VENC, ADEC, AENC, VI, and AI
- Graphics processing
  - HiFB, HiGo, TDE, JPEG, JPGE, and GPU
- Peripheral processing
  - Cipher, OTP, PMOC, Frontend, I2C, SCI, KEYLED, GPIO, IR, WDG, and C51

## 1.3 Function Conventions

### 1.3.1 Open/Close

The open and close operations are performed to start and close enumerable devices respectively.

Enumerable devices are the devices whose number is limited. For example, there is only one physical SOUND device (HI\_UNF\_SND\_0), it is an enumerable device.

The open and close operations are performed by setting the parameter ID such as HI\_UNF\_SND\_0. The ID is fixed and each ID maps to a specific device.

In a process, a code indicating success is returned after a device is started for multiple times, and all operation results are the same. When a device is started during multiple processes, if this device supports multiple processes, all operation results are the same; if this device does not support multiple processes, the open operation in the second process fails.

### 1.3.2 Create/ Destroy

The create and destroy operations are performed to create and destroy innumerable devices respectively.

Innumerable devices are the devices whose number is not limited by hardware. That is, the number of innumerable devices is unlimited if the resources are sufficient. For example, numerous AVPLAYs can be created if the memory is sufficient, the AVPLAY therefore is an innumerable device.

As the resources are limited in practice, the SDK defines the allowed maximum number of innumerable devices.



### 1.3.3 Handle

The handle is used to identify an innumerable device.

If a device is successfully created, its handle is returned. The subsequent operations for this device are performed by using this handle.

If an innumerable device is successfully created for multiple times, different handles are returned. Each handle, however, is valid only in the current process. That is, handles cannot be transferred across processes, but different handles can be created in different processes.

A handle is a 32-bit data segment. Its lower eight bits indicate the device ID, and its upper 16 bits indicate the module ID. For example, 0x00110000 indicates device 0 on the module whose ID is 0x11.

### 1.3.4 Attach/Detach

The attach and detach operations are used to attach and detach the direct correlation between devices.

To be specific, the attach operation is performed to attach a data input source (producer) to a target device (consumer). For example, attach an AVPLAY to a window. After the attach operation, the consumer automatically acquires the data generated by the producer without user intervention.

The attach and detach interfaces are always provided by the module of the target device. For example, the SOUND module provides HI\_UNF\_SND\_Attach. This indicates that the SOUND module is a consumer, and an input source needs to be attached to the SOUND module.

After being attached, the target device is in started/enabled status automatically; after being detached, the target device is in stopped/disabled status automatically.

For multi-level attachment, you need to attach the devices in sequence from the last device, and detach the devices in sequence from the first device.

If the attributes of multiple devices attached together can be configured as the same, you can set the attributes by using the last device.

### 1.3.5 Get/Put

The get and put operations are performed to transmit data to a module.

The get operation can be understood as the operation of attempting to write data to a module. In non-block mode, if the get operation is successful, an available buffer in the module is returned. You can then write data to this module. However, if the speed of the get operation is faster than that of data processing, no memory is available in the module. As a result, the get operation fails and the error code HI\_UNF\_ERR\_XXX\_BUF\_FULL or similar error code is returned. You need to wait until there is available memory in the module, and then perform the get operation again. In block mode, the get interface is released only after an available buffer is obtained or a timeout occurs.

The put operation can be understood as the operation of submitting data to a module after writing data. If the put operation is successful, the corresponding module starts to process the written data.



It is recommended that the get and put operations be used in pairs. That is, a put operation is required after a successful get operation. The operation results are the same if the get interface is called repeatedly.



### CAUTION

In non-block mode, if HI\_UNF\_ERR\_XXX\_BUF\_FULL or similar error code is returned after the get interface is called, you need to call the usleep function to clean up the CPU, and then call the get interface again. Otherwise, the CPU usage may remain high. Typically, usleep (10\*1000) is used to clean up the CPU.

## 1.3.6 Acquire/Release

The acquire and release operations are performed to obtain data from a module.

The acquire operation can be understood as the operation of attempting to read data from a module. In non-block mode, if the acquire operation is successful, an internal buffer storing the required data is returned. Then you can fetch and process the data. If the speed of the acquire operation is faster than that of data processing, no valid data is available in the module. As a result, the acquire operation fails and the error code

HI\_UNF\_ERR\_XXX\_BUF\_EMPTY or similar error code is returned. You need to wait until there is new data in the module, and then perform the acquire operation again. In block mode, the acquire interface is released only after data is acquired successfully or a timeout occurs.

The release operation can be understood as the operation of releasing the data buffer for the module after processing the data read from the module.

It is recommended that the acquire and release operations be used in pairs. That is, a release operation is required after a successful acquire operation. Typically the acquire interface cannot be called repeatedly.



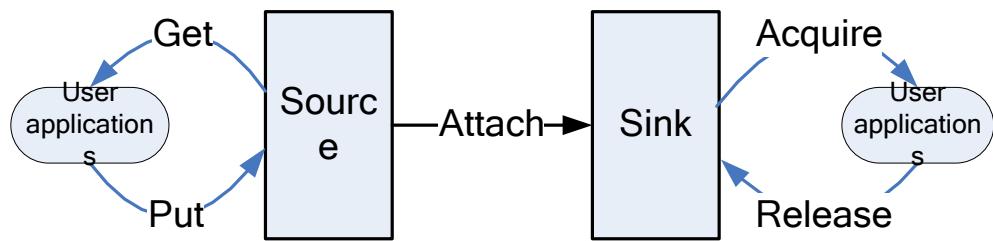
### CAUTION

In non-block mode, if HI\_UNF\_ERR\_XXX\_BUF\_EMPTY or similar error code is returned after the acquire interface is called, you need to call the usleep function to clean up the CPU, and then call the acquire interface again. Otherwise, the CPU usage may remain high. Typically, usleep (10\*1000) is used to clean up the CPU.

[Figure 1-2](#) shows the typical procedure for processing data to illustrate the preceding concepts.



**Figure 1-2** Typical procedure for processing data



As shown in [Figure 1-2](#), a user application transfers data to a module by calling the get and put interfaces, and acquires data from a module by calling the acquire and release interfaces. The attach relationship between modules is set up by calling the attach interface.



# Contents

---

<b>2 Environment Configuration.....</b>	<b>1</b>
2.1 SDK Compilation.....	1
2.2 Compilation Result.....	2
2.3 Running Environment .....	2



# 2 Environment Configuration

## 2.1 SDK Compilation

The compilation method described here is dedicated for the compilation on Linux. If Android is used, see the *Android Solution Development Guide*.

To compile the SDK, perform the following steps:

**Step 1** Install and configure the Linux server.

Linux version: Ubuntu 10 or later is recommended.

glibc: Version 2.11.1 is recommended.

GNU Make: Version 3.8.1 or later is recommended.

Shell: It must be bash.

The ssh, samba, and nfs services are properly configured and started.

**Step 2** Install the cross compilation tool chain:

\server\_install

**Step 3** Select a configuration file.

The default configuration file for the demo board is provided in the configs directory. Run the following command to select the configuration file:

```
cp configs/chip name/xxxx.cfg ./cfg.mak
```

**Step 4** Compile the code.

```
make build
```



### NOTE

**make build** is the command used in the first compilation. For subsequent compilation, run **make rebuild**.

----End



## 2.2 Compilation Result

After the SDK is compiled, all files required for development are copied to the corresponding folders in the **pub** directory. The following describes the files stored in each folder.

- **pub\image**: boot, kernel, and rootbox images
- **pub\include**: header files of each module
- **pub\kmod**: compiled .ko drivers
- **pub\lib\extern**: dynamic library files used in dlopen mode, such as audio and graphics decoding libraries and ffmpeg
- **pub\lib\share**: library files for each module, such as the common and msp modules
- **pub\lib\static**: static library files for each module, such as the common and msp modules

## 2.3 Running Environment

Set the running environment as follows before running applications:

**Step 1** Burn the fastboot, boot, kernel, and the system file images by using the HiTool in the **tools\windows\HiTool** directory.

Select the partition table file that corresponds to the chip to burn the image. For example, for Hi3798M V100, the partition table file is **pub\image\partitions\_hi3798mv100\_emmc.xml**.

For details, see **HiTool Quick Start Video.exe**.

**Step 2** Restart the board, set the network parameters, and mount the SDK on the Linux server to the board in NFS mode.

**Step 3** Run various samples in the **sample** directory of the SDK.

For details, see **readme.txt** in the **sample** directory.

**----End**



# Contents

---

<b>3 DEMUX .....</b>	<b>1</b>
3.1 Overview .....	1
3.2 Important Concepts .....	1
3.3 Features .....	2
3.4 Development Guide .....	4
3.4.1 Data Receiving.....	4
3.4.2 Audio and Video Playback .....	8
3.4.3 TS Recording .....	9
3.4.4 IPTV or PVR Playback .....	10
3.4.5 Using the TSO Port .....	12



# Figures

<b>Figure 3-1</b> DEMUX in the system.....	1
<b>Figure 3-2</b> Process for receiving data .....	5
<b>Figure 3-3</b> Working process of DEMUX filters .....	6
<b>Figure 3-4</b> Process for playing audio and video clips.....	8
<b>Figure 3-5</b> Process for recording TSs .....	9
<b>Figure 3-6</b> IPTV or PVR playback process .....	11



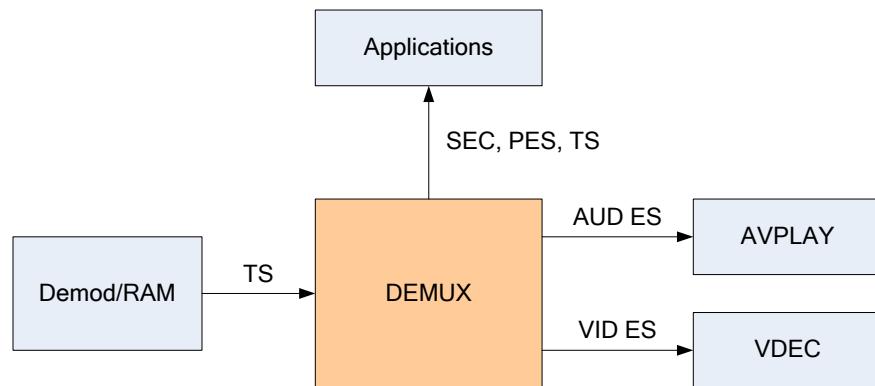
# 3 DEMUX

## 3.1 Overview

The DEMUX module is used to parse and demultiplex MPEG2 TSs complying with the standards of the ISO 13818-1 (GB 17975-1) system layer. It provides data receiving and TS recording functions.

[Figure 3-1](#) shows the DEMUX module in the system.

**Figure 3-1** DEMUX in the system



The DEMUX module receives TSs from the Demod or memory and extracts the SEC, PES, TS, audio, or video data based on the configuration of user applications. The applications obtain and process SEC, PES, and TS data, the AVPLAY obtains and processes audio data, and the VDEC obtains and processes video data.

## 3.2 Important Concepts

[Port]

Ports are used to process TSs and can be classified into IF ports, TSI ports, RAM ports, and TSO ports.

[IF port]



The IF port is the input port for tuner signals. It uses the embedded QAM and is numbered from 0. There is typically no more than one IF port.

[TSI port]

The TSI port is also the input port for tuner signals. It is different from the IF port because it uses the external QAM. The DEMUX module has multiple TSI ports, and the TSI ports are numbered TSI port 0, TSI port 1..., and TSI port N.

[RAM port]

The RAM port is used to process TSs input from the memory. The DEMUX module has multiple RAM ports, which are numbered from RAM port 0.

[TSO port]

The TSO port is used to output TSs.

[Child DEMUX]

The DEMUX module contains multiple child DEMUXs, which are numbered from DEMUX0.

[Channel]

The channel is used to filter PIDs and parse data. Channels are classified into SEC channels, PES channels, TS channels, audio channels, and video channels.

[Filter]

The filter is used to filter data parsed by channels.

[PoolBuf]

PoolBuf is the shared data buffer for the SEC, PES, and POST channels.

### 3.3 Features

The DEMUX module provides the following functions:

- Initialization and deinitialization. The following APIs are provided:
  - HI\_UNF\_DMX\_Init: Initializes the DEMUX module.
  - HI\_UNF\_DMX\_DeInit: Deinitializes the DEMUX module.
- Capability obtaining. The following API is provided:  
HI\_UNF\_DMX\_GetCapability: Obtains the capabilities of the DEMUX module.
- Port management. The following APIs are provided:
  - HI\_UNF\_DMX\_GetTSPortAttr: Obtains (IF/TSI/RAM) port attributes.
  - HI\_UNF\_DMX\_SetTSPortAttr: Sets (IF/TSI/RAM) port attributes.
  - HI\_UNF\_DMX\_GetTSOPortAttr: Obtains TSO port attributes.
  - HI\_UNF\_DMX\_SetTSOPortAttr: Sets TSO port attributes.
  - HI\_UNF\_DMX\_AttachTSPort: Binds a port to a child DEMUX.
  - HI\_UNF\_DMX\_DetachTSPort: Unbinds a port from a child DEMUX.
  - HI\_UNF\_DMX\_GetTSPortId: Obtains the port that is bound to the child DEMUX.
  - HI\_UNF\_DMX\_CreateTSBuffer: Creates the TS buffer for the RAM port.
  - HI\_UNF\_DMX\_DestroyTSBuffer: Destroys the TS buffer for the RAM port.



- HI\_UNF\_DMX\_GetTSBuffer: Obtains the available memory blocks in the TS buffer.
- HI\_UNF\_DMX\_GetTSBufferEx: Obtains the available memory blocks in the TS buffer.
- HI\_UNF\_DMX\_ResetTSBuffer: Reset TS Buffer.
- HI\_UNF\_DMX\_PutTSBuffer: Writes data to the TS buffer.
- HI\_UNF\_DMX\_PutTSBufferEx: Writes data to the TS buffer.
- HI\_UNF\_DMX\_GetTSBufferStatus: Obtains the status of the TS buffer.
- HI\_UNF\_DMX\_GetTSBufferPortId: Obtains the RAM port based on the TS buffer handle.
- HI\_UNF\_DMX\_GetTSBufferHandle: Obtains the TS buffer handle based on the RAM port.
- HI\_UNF\_DMX\_GetTSPortPacketNum: Obtains the number of TS packets over a port.
- Channel management. The following APIs are provided:
  - HI\_UNF\_DMX\_GetChannelDefaultAttr: Obtains the default attributes of a channel.
  - HI\_UNF\_DMX\_CreateChannel: Creates a channel.
  - HI\_UNF\_DMX\_DestroyChannel: Destroys a channel.
  - HI\_UNF\_DMX\_GetChannelAttr: Obtains channel attributes.
  - HI\_UNF\_DMX\_SetChannelAttr: Sets channel attributes.
  - HI\_UNF\_DMX\_SetChannelPID: Sets the channel PID.
  - HI\_UNF\_DMX\_GetChannelPID: Obtains the channel PID.
  - HI\_UNF\_DMX\_OpenChannel: Enables a channel.
  - HI\_UNF\_DMX\_CloseChannel: Disables a channel.
  - HI\_UNF\_DMX\_GetChannelStatus: Obtains the status of a channel.
  - HI\_UNF\_DMX\_GetChannelTsCount: Obtains the number of TS packets of a channel.
  - HI\_UNF\_DMX\_GetChannelHandle: Obtains the channel ID based on the PID.
  - HI\_UNF\_DMX\_GetFreeChannelCount: Obtains the number of available channels.
  - HI\_UNF\_DMX\_GetScrambledFlag: Obtains the scrambling flag of a channel.
- Filter management. The following APIs are provided:
  - HI\_UNF\_DMX\_CreateFilter: Creates a filter.
  - HI\_UNF\_DMX\_DestroyFilter: Destroys a filter.
  - HI\_UNF\_DMX\_DeleteAllFilter: Destroys all filters of a channel.
  - HI\_UNF\_DMX\_SetFilterAttr: Sets filter attributes.
  - HI\_UNF\_DMX\_GetFilterAttr: Obtains filter attributes.
  - HI\_UNF\_DMX\_AttachFilter: Binds a filter to a channel.
  - HI\_UNF\_DMX\_DetachFilter: Unbinds a filter from a channel.
  - HI\_UNF\_DMX\_GetFilterChannelHandle: Obtains the channel ID based on the filter.
  - HI\_UNF\_DMX\_GetFreeFilterCount: Obtains the number of available filters.
- Data obtaining. The following APIs are provided:
  - HI\_UNF\_DMX\_GetDataHandle: Obtains the IDs of channels which have received data.
  - HI\_UNF\_DMX\_SelectDataHandle: Obtains the IDs of selected channels which have received data.



- HI\_UNF\_DMX\_AcquireBuf: Obtains the buffer for storing channel data.
- HI\_UNF\_DMX\_ReleaseBuf: Releases the buffer that stores channel data.
- TS recording. The following APIs are provided:
  - HI\_UNF\_DMX\_CreateRecChn: Creates a recording channel.
  - HI\_UNF\_DMX\_DestroyRecChn: Destroys a recording channel.
  - HI\_UNF\_DMX\_AddRecPid: Adds the PID of data to be recorded when the data with selected PIDs is being recorded.
  - HI\_UNF\_DMX\_DelRecPid: Deletes the added PID channel when the data with selected PIDs is being recorded.
  - HI\_UNF\_DMX\_DelAllRecPid: Deletes all added PID channels when the data with selected PIDs is being recorded.
  - HI\_UNF\_DMX\_AddExcludeRecPid: Adds the PIDs of data to be excluded from recording when all data is being recorded.
  - HI\_UNF\_DMX\_DelExcludeRecPid: Cancels the PID of data that is excluded from recording when all data is being recorded.
  - HI\_UNF\_DMX\_DelAllExcludeRecPid: Cancels all the PIDs of data that is excluded from recording when all data is being recorded.
  - HI\_UNF\_DMX\_StartRecChn: Enables a recording channel.
  - HI\_UNF\_DMX\_StopRecChn: Disable a recording channel.
  - HI\_UNF\_DMX\_AcquireRecData: Obtains recorded TS data.
  - HI\_UNF\_DMX\_ReleaseRecData: Releases the obtained recorded data.
  - HI\_UNF\_DMX\_AcquireRecIndex: Obtains the index information generated during recording.
  - HI\_UNF\_DMX\_AcquireRecDataAndIndex: Obtains the recorded data and index information at the same time.
  - HI\_UNF\_DMX\_ReleaseRecDataAndIndex: Releases the recorded data and index information obtained at the same time.
  - HI\_UNF\_DMX\_GetRecBufferStatus: Obtains the status of the current recording buffer.

## 3.4 Development Guide

The DEMUX module is used in the following scenarios:

- Data receiving
- Audio and video playback
- TS recording
- Internet Protocol television (IPTV) or personal video recorder (PVR) playback

### 3.4.1 Data Receiving

#### Scenario

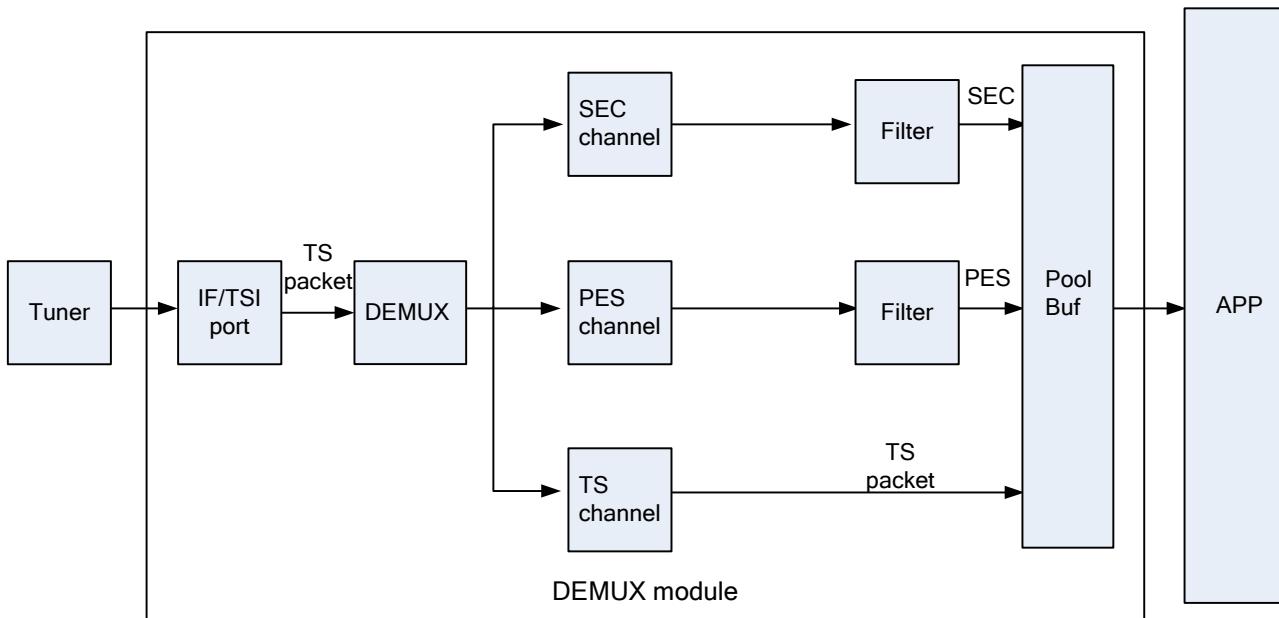
The DEMUX module can be used to receive SEC, PES, and TS data such as program specific information (PSI) or specific information (SI).



## Working Process

Figure 3-2 shows the process for receiving data.

**Figure 3-2** Process for receiving data



Note the following:

- The dotted frame in the figure indicates the part that can be ignored in practice.
- Filters are used to filter SEC and PES data:
  - Byte 1 and bytes 4–18 of SEC data are filtered by the SEC channel.
  - Byte 4 and bytes 7–21 of PES data are filtered by the PES channel.

Set the filtering parameters as follows:

```
#define DMX_FILTER_MAX_DEPTH 16

typedef struct
{
    HI_U32 u32FilterDepth;
    HI_U8 au8Match[DMX_FILTER_MAX_DEPTH];
    HI_U8 au8Mask[DMX_FILTER_MAX_DEPTH];
    HI_U8 au8Negate[DMX_FILTER_MAX_DEPTH];
} HI_UNF_DMX_FILTER_ATTR_S;
```

The filtering parameters include **FilterDepth**, **Match**, **Mask**, and **Negate**, which are described as follows:

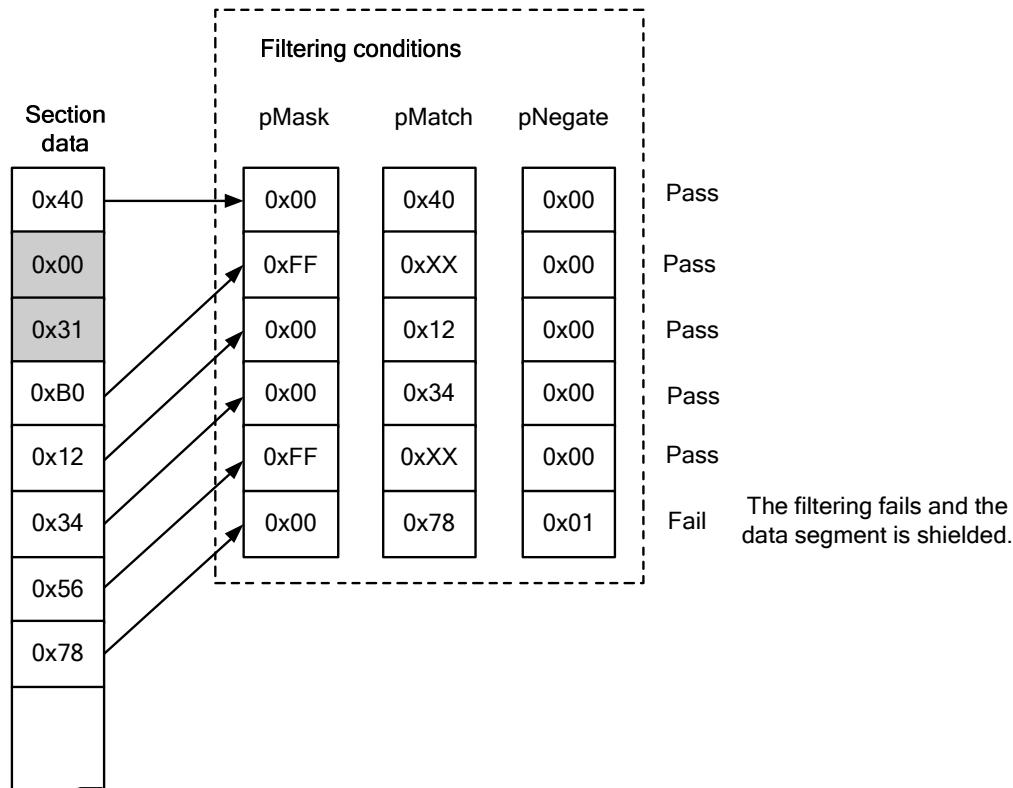
- **FilterDepth** specifies the allowed valid length of values of **Match**, **Mask**, and **Negate**.
- **Match** specifies the match value.



- **Mask** is the mask value. **0** indicates that the received data must match the bit in **Match**, and **1** indicates that the bit can be ignored.
- **Negate** specifies whether to inverse bytes. **0** indicates that matched data is received, and **1** indicates that unmatched data is received, and the match and mask results are inversed. Other values are invalid.

Figure 3-3 shows the working process of DEMUX filters.

**Figure 3-3** Working process of DEMUX filters



The following describes how to receive SEC data. PES or TS data is received in the same way, except that the filter usage is different.

Before using the APIs provided by the DEMUX module to receive SEC data, you must find out the port from which the data comes and whether the data and data PIDs are scrambled. (The scrambler must be set if data is to be scrambled. For details, see the related sections.)

If tuner 0 is locked, the SEC data is received as follows:

- Step 1** Initialize the DEMUX module by calling HI\_UNF\_DMX\_Init.
- Step 2** Bind tuner port 0 to DEMUX0 by calling HI\_UNF\_DMX\_AttachTSPort.
- Step 3** Create an SEC channel on DEMUX0 by calling HI\_UNF\_DMX\_CreateChannel.
- Step 4** Set the channel PID by calling HI\_UNF\_DMX\_SetChannelPID.
- Step 5** Create a filter by calling HI\_UNF\_DMX\_CreateFilter and set filtering parameters.
- Step 6** Bind the filter to the channel by calling HI\_UNF\_DMX\_AttachFilter.



- Step 7** Enable the channel by calling HI\_UNF\_DMX\_OpenChannel.
- Step 8** Query whether the channel has data by calling HI\_UNF\_DMX\_GetDataHandle.
- Step 9** Obtain channel data (if any) by calling HI\_UNF\_DMX\_AcquireBuf.
- Step 10** Process obtained channel data.
- Step 11** Release the buffer for storing channel data by calling HI\_UNF\_DMX\_ReleaseBuf.
- Step 12** Unbind the filter from the channel by calling HI\_UNF\_DMX\_DetachFilter.
- Step 13** Destroy the filter by calling HI\_UNF\_DMX\_DestroyFilter.
- Step 14** Disable the channel by calling HI\_UNF\_DMX\_CloseChannel.
- Step 15** Destroy the channel by calling HI\_UNF\_DMX\_DestroyChannel.
- Step 16** Unbind the port from DEMUX0 by calling HI\_UNF\_DMX\_DetachTSPort.
- Step 17** Deinitialize the DEMUX module by calling HI\_UNF\_DMX\_DeInit.

----End

## Notes

Note the following:

- Filters are required to receive SEC data.
- When receiving PES data, you can use filters or receive all data without using filters.
- Filters cannot be used when TS data is being received.
- The maximum filtering depth of a filter is 16 bytes.
- A filter can be bound to only one channel.
- A channel can be bound to 32 filters, and the filters are ORed. To be specific, data can be received after passing any filter of the channel.
- Filters can be bound to or unbound from a channel when the channel is enabled. When a filter is unbound, data that has passed the filter is not deleted.
- SEC, PES, and TS data are stored in PoolBuf.
- The default size of PoolBuf is 2 MB. It can be configured by running **make menuconfig** as follows:

```
Msp --->
DEMUX Config --->
(0x200000) DEMUX Poolbuf Size
```

## Samples

The following are the samples that illustrate how to set the filtering parameters.

Sample 1: Only the SEC data with the first byte of 0x00 is received.

```
stFilterAttr.au8Match[0] = 0x00;
stFilterAttr.au8Mask[0] = 0x00;
stFilterAttr.au8Negate[0] = 0;
stFilterAttr.u32FilterDepth = 1;
```



Sample 2: Only the SEC data with the first byte of 0x50 or 0x51 is received.

Only bit[0] is different for 0x50 and 0x51. Other bits are the same. Therefore, just ignore bit[0].

```
stFilterAttr.au8Match[0] = 0x50;  
stFilterAttr.au8Mask[0] = 0x01;  
stFilterAttr.au8Negate[0] = 0;  
stFilterAttr.u32FilterDepth = 1;
```

#### NOTE

In this sample, the result is the same if Match[0] is set to **0x51**.

Sample 3: The SEC data with the first byte of 0x50 or 0x51 is excluded (not received).

```
stFilterAttr.au8Match[0] = 0x50;  
stFilterAttr.au8Mask[0] = 0x01;  
stFilterAttr.au8Negate[0] = 1;  
stFilterAttr.u32FilterDepth = 1;
```

#### NOTE

In this sample, the result is the same if Match[0] is set to **0x51**.

For details about other samples, see [sample\demux\dvb\\_demux\\_sample.c](#).

## 3.4.2 Audio and Video Playback

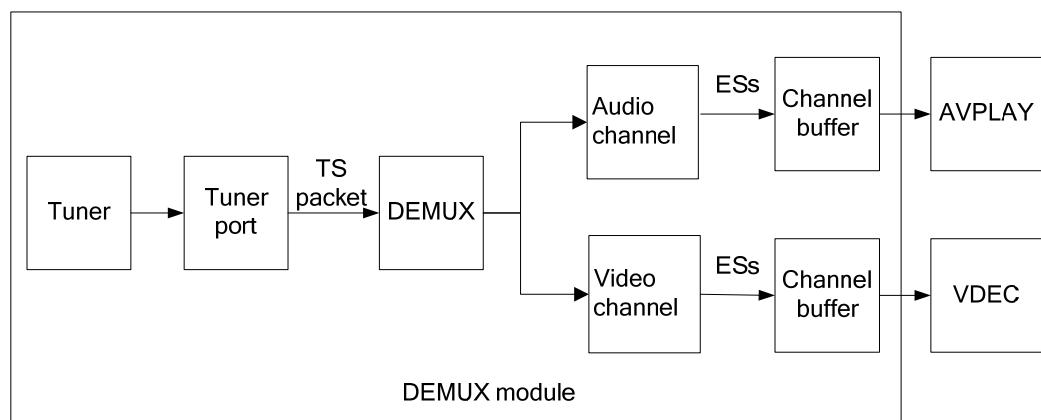
### Scenario

The DEMUX module can be used to play audio and video ESs.

### Working Process

[Figure 3-4](#) shows the process for playing audio and video clips.

**Figure 3-4** Process for playing audio and video clips



The audio and video channels of the DEMUX module are managed by the AVPLAY module. For details, see chapter 4 "AVPLAY."



## Notes

None

## Sample

None

### 3.4.3 TS Recording

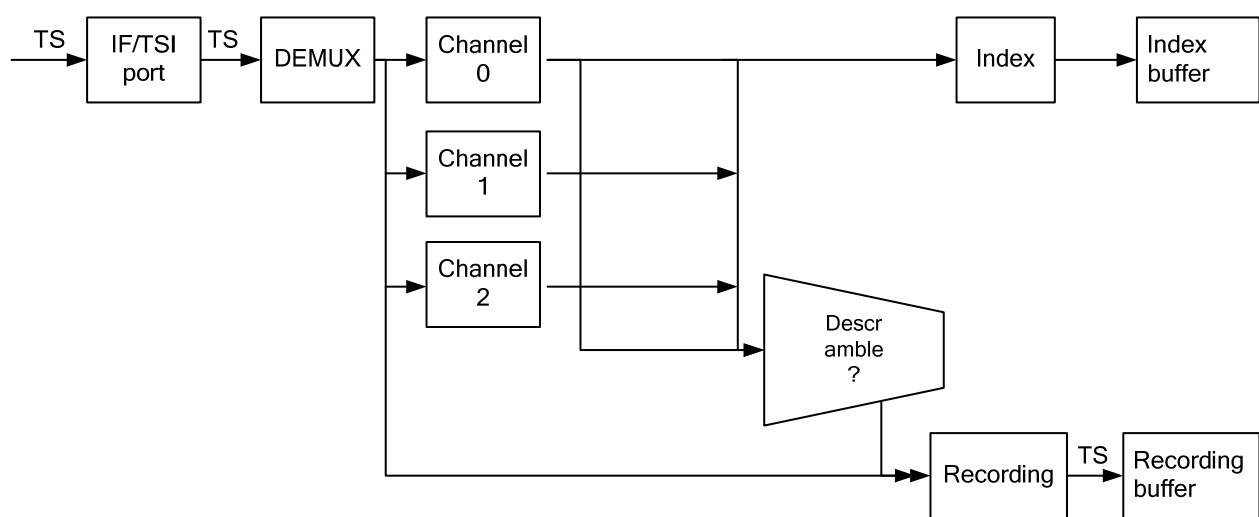
#### Scenario

The DEMUX module supports TS recording. The TS recording function can be used to implement the PVR functions. During TS recording, you can record all data or some data by selecting the PIDs. If you record all data, you can exclude the data with specific PIDs. If you record some data by selecting the PIDs, you can choose the data of a PID to generate index information.

#### Working Process

[Figure 3-5](#) shows the process for recording TSs.

**Figure 3-5** Process for recording TSs



Note the following:

- If all the data of the DEMUX port is saved and written to the recording buffer, all the data is recorded.
- The DEMUX module specifies PIDs through the recording channels, and only the data of specified PIDs is written to the buffer.

If tuner 0 is locked, the process for recording the data of specified PIDs is as follows:

**Step 1** Initialize the DEMUX module by calling HI\_UNF\_DMX\_Init.

**Step 2** Bind tuner port 0 to DEMUX0 by calling HI\_UNF\_DMX\_AttachTSPort.



**Step 3** Create a recording channel on DEMUX0 by calling HI\_UNF\_DMX\_CreateRecChn, and set the recording parameters as follows:

- Set the recording buffer to **4 MB**.
- Set the recording type to **HI\_UNF\_DMX\_REC\_TYPE\_SELECT\_PID**.
- Set **bDescrambled** to **HI\_TRUE** to record descrambled data (if supported by the chip).
- Set the index type to **HI\_UNF\_DMX\_REC\_INDEX\_TYPE\_VIDEO**.
- Set the index PID to the PID of the video channel.
- Set the video encoding protocol to the encoding protocol of the video PID.

**Step 4** Add PIDs of all data to be recorded by calling HI\_UNF\_DMX\_AddRecPid repeatedly.

**Step 5** Start recording by calling HI\_UNF\_DMX\_StartRecChn.

**Step 6** Obtain recorded data by calling HI\_UNF\_DMX\_AcquireRecData.

**Step 7** Store recorded data to a file.

**Step 8** Call HI\_UNF\_DMX\_ReleaseRecData to release the buffer storing the recorded data obtained.

**Step 9** Obtain index data by calling HI\_UNF\_DMX\_AcquireRecIndex.

**Step 10** Store index data to a file.

**Step 11** Stop recording by calling HI\_UNF\_DMX\_StopRecChn.

**Step 12** Delete PIDs of all recorded data by calling HI\_UNF\_DMX\_DelRecPid repeatedly.

**Step 13** Destroy the recording channel by calling HI\_UNF\_DMX\_DestroyRecChn.

----End

## Notes

When all data is recorded, the index cannot be generated, and TSs cannot be descrambled.

## Sample

See sample\demux\dvb\_demux\_record.c.

## 3.4.4 IPTV or PVR Playback

### Scenario

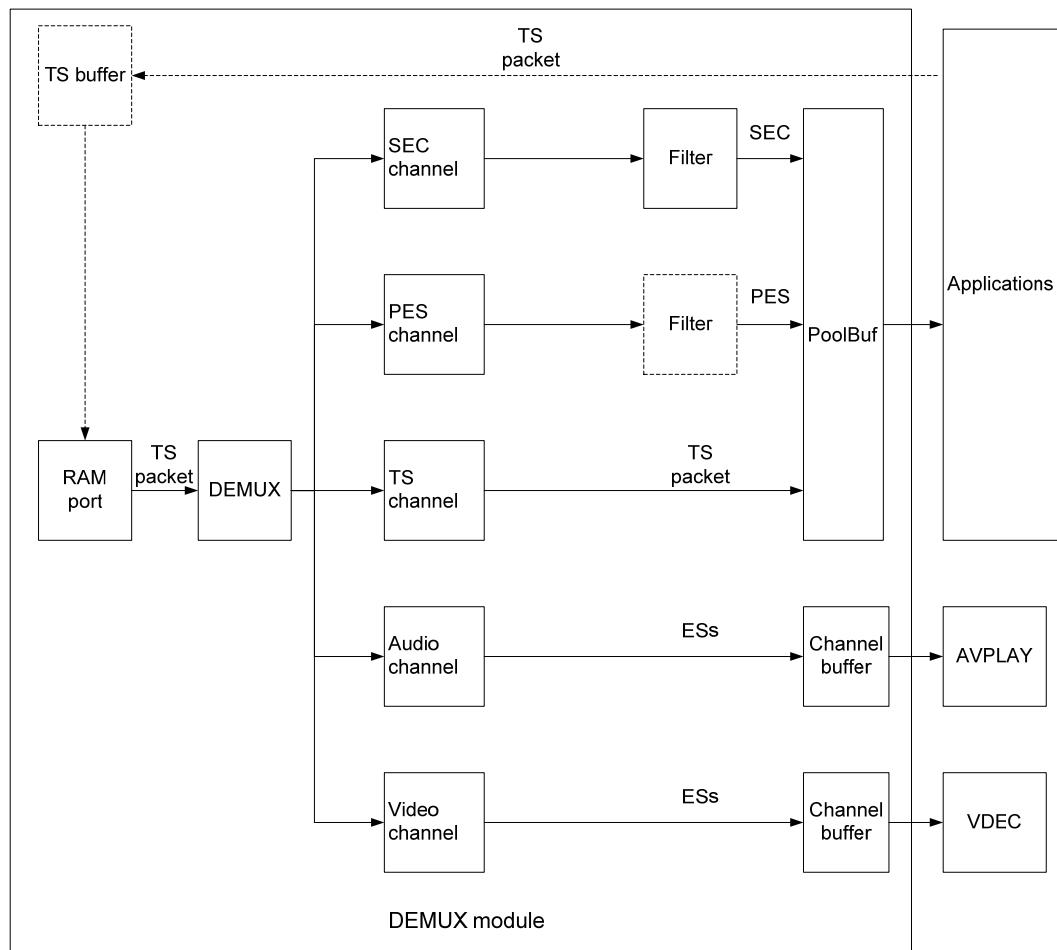
The DEMUX module can be used to implement the IPTV or PVR playback. This section describes the usage of the RAM port.

### Working Process

Figure 3-6 shows the IPTV or PVR playback process.



**Figure 3-6** IPTV or PVR playback process



The process for injecting data in TS files into the TS buffer during PVR playback is as follows:

- Step 1** Create a TS buffer on RAM port 0 by calling HI\_UNF\_DMX\_CreateTSBuffer.
- Step 2** Obtain the available memory blocks in the TS buffer by calling HI\_UNF\_DMX\_GetTSBuffer.
- Step 3** Read data in the TS files (or read data obtained from the network) and copy the data to the available memory blocks.
- Step 4** Inject data in the TS buffer by calling HI\_UNF\_DMX\_PutTSBuffer.
- Step 5** Repeat steps 2 to 4 until the playback is complete or all data is received.
- Step 6** Destroy the TS buffer by calling HI\_UNF\_DMX\_DestroyTSBuffer.

----End

## Notes

Note the following:

- You are advised to create a thread to store TS data to the TS buffer.



- It is recommended that at least 20 TS packets are stored to the TS buffer at a time to improve the RAM port efficiency.

## Sample

See `sample\tsplay\tsplay.c`.

## 3.4.5 Using the TSO Port

### Scenario

The DEMUX module can output TSs from the IF/TSI/RAM port to the TSO port. The TSO port is used in scenarios such as CI descrambling and reducing the external QAMs required (the TSO source is the IF port).

### Working Process

The process for using the TSO port is as follows:

**Step 1** Obtain the default TSO port attributes by calling `HI_UNF_DMX_GetTSOPortAttr`.

**Step 2** Set the TSO port attributes by calling `HI_UNF_DMX_SetTSOPortAttr`.

----End

### Notes

The TSO attribute data structure is defined as follows:

```
typedef struct hiUNF_DMX_TSO_PORT_ATTR_S
{
    HI_BOOL bEnable;
    HI_BOOL bClkReverse;
    HI_UNF_DMX_PORT_E enTSSource;
    HI_UNF_DMX_TSO_CLK_MODE_E enClkMode;
    HI_UNF_DMX_TSO_VALID_MODE_E enValidMode;
    HI_BOOL bBitSync;

    HI_BOOL bSerial;
    HI_UNF_DMX_TSO_SERIAL_BIT_E enBitSelector;

    HI_BOOL bLSB;

    HI_UNF_DMX_TSO_CLK_E enClk;
    HI_U32 u32ClkDiv;
} HI_UNF_DMX_TSO_PORT_ATTR_S;
```

- The fields in red can be ignored because the default values are typically used.



- The TSO port output clock equals the TSO module working clock (**enClk**) divided by the divider (**u32ClkDiv**).
- The TSO clock mode (**enClkMode**) and the valid signal mode cannot be set to the following values at the same time:  
`enClkMode = HI_UNF_DMX_TSO_CLK_MODE_NORMAL;`  
`enValidMode = HI_UNF_DMX_TSO_VALID_ACTIVE_OUTPUT;`
- The TSO divider **u32ClkDiv** must be an integral multiple of 2 ranging from 2 to 32.

## Sample

None



# Contents

---

<b>4 AVPLAY.....</b>	<b>1</b>
4.1 Overview .....	1
4.2 Important Concepts .....	2
4.3 Features .....	2
4.4 Development Guide .....	4
4.4.1 DVB Playback .....	5
4.4.2 TS Playback .....	6
4.4.3 ES Playback .....	7
4.4.4 I-Frame Decoding .....	8
4.4.5 Multi-Track Playback.....	9
4.4.6 External Audio and Video Decoding Libraries .....	10
4.4.7 Playback Control.....	12
4.4.8 Last Frame Output .....	13
4.4.9 Attribute Configuration .....	15
4.4.10 Event Processing.....	19
4.4.11 Low Delay.....	20



# Figures

<b>Figure 4-1</b> Position of the AVPLAY module in a typical media play terminal solution .....	1
<b>Figure 4-2</b> Logical block diagram of the AVPLAY module in ES mode.....	4
<b>Figure 4-3</b> Relationship between the external audio decoding library and the SDK.....	11
<b>Figure 4-4</b> Sample of an external .mp3 audio library .....	12



# Tables

<b>Table 4-1</b> Playback status control .....	13
<b>Table 4-2</b> AVPLAY attributes .....	15
<b>Table 4-3</b> Stream attributes .....	16
<b>Table 4-4</b> VDEC attributes .....	17
<b>Table 4-5</b> Audio/Video synchronization attributes.....	18



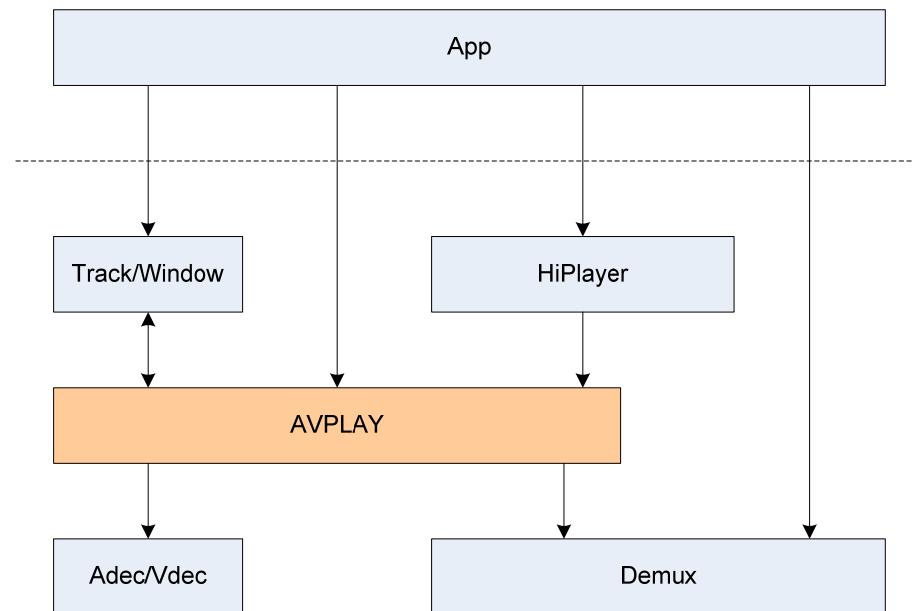
# 4 AVPLAY

## 4.1 Overview

The AVPLAY module integrates audio and video playback modules in the SDK and provides APIs for basic playback services.

**Figure 4-1** shows the position of the AVPLAY module in a typical media playback terminal solution.

**Figure 4-1** Position of the AVPLAY module in a typical media play terminal solution



The AVPLAY module relies on the audio decoder (ADEC), video decoder (VDEC), and DEMUX modules. It provides APIs related to basic playback services for applications or middleware players.



## 4.2 Important Concepts

### [Stream type]

The stream type is the type of streams supported by the AVPLAY, including the TS and ES.

### [Synchronization control]

Synchronization control includes audio synchronization, program clock reference (PCR) synchronization, and free playback.

### [Synchronization start region]

If the difference between the audio and video PTSs exceeds this region, audio and video asynchronization occurs and synchronization adjustment starts. The adjustment mode can be configured.

### [Synchronization exception region]

If the difference between the audio and video PTSs exceeds this region, streams are abnormal and synchronization adjustment starts. The adjustment mode can be configured.

## 4.3 Features

The AVPLAY module provides the following functions:

- Player management, including creating and destroying players and setting player attributes. The following APIs are provided:
  - HI\_UNF\_AVPLAY\_GetDefaultConfig: Obtains default player configuration.
  - HI\_UNF\_AVPLAY\_Create: Creates a player.
  - HI\_UNF\_AVPLAY\_Destroy: Destroys a player.
  - HI\_UNF\_AVPLAY\_ChnOpen: Creates a channel.
  - HI\_UNF\_AVPLAY\_ChnClose: Destroys a channel.
  - HI\_UNF\_AVPLAY\_SetAttr: Sets attributes.
  - HI\_UNF\_AVPLAY\_GetAttr: Obtains attributes.
  - HI\_UNF\_AVPLAY\_SetDecodeMode: Sets the decoding mode.
- Playback control, including operations such as starting, stopping, pausing, resuming, resetting, and trick playing. The following APIs are provided:
  - HI\_UNF\_AVPLAY\_Start: Starts the playback.
  - HI\_UNF\_AVPLAY\_Stop: Stops the playback.
  - HI\_UNF\_AVPLAY\_Pause: Pauses the playback.
  - HI\_UNF\_AVPLAY\_Resume: Resumes the playback.
  - HI\_UNF\_AVPLAY\_Tplay: Plays streams at a speed several times of the normal speed.
  - HI\_UNF\_AVPLAY\_Reset: Resets the playback.
  - HI\_UNF\_AVPLAY\_Step: Plays streams in step forward mode.
- Event management, including registering and reporting the playback events. The following APIs are provided:
  - HI\_UNF\_AVPLAY\_RegisterEvent: Registers a playback event.

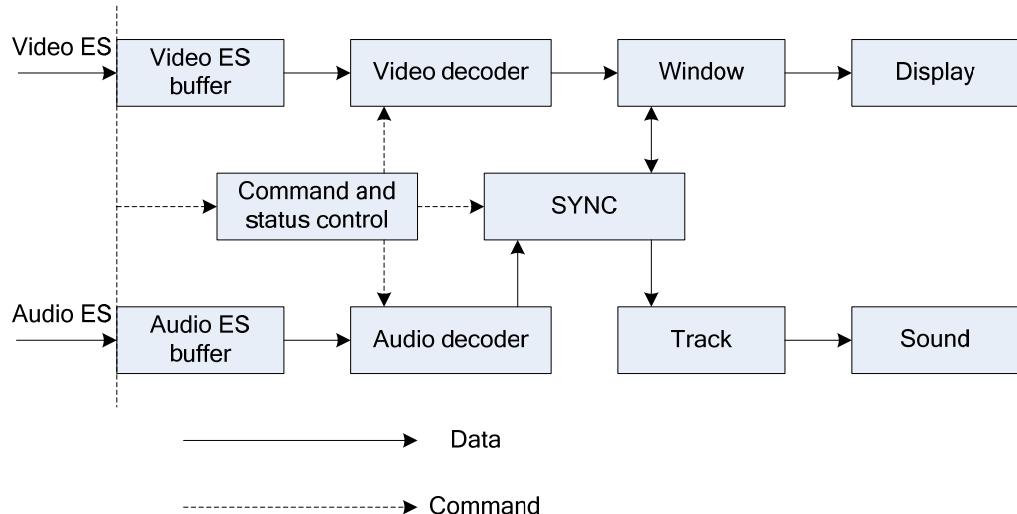


- HI\_UNF\_AVPLAY\_UnRegisterEvent: Deregisters a playback event.
- Playback status query, including querying the status and stream information about the player. The following APIs are provided:
  - HI\_UNF\_AVPLAY\_GetStatusInfo: Obtains the AVPLAY status information.
  - HI\_UNF\_AVPLAY\_GetStreamInfo: Obtains the stream information.
  - HI\_UNF\_AVPLAY\_IsBuffEmpty: Queries whether the AVPLAY buffer is empty.
- Buffer management. The following APIs are provided:
  - HI\_UNF\_AVPLAY\_GetBuf: Obtains the ES buffer.
  - HI\_UNF\_AVPLAY\_PutBuf: Releases the ES buffer.
  - HI\_UNF\_AVPLAY\_PutBufEx: Releases the ES buffer when streams are transmitted by frame.
- Output of the last frame of the stream. The last frame of the stream is output and the event is reported. The following API is provided:
  - HI\_UNF\_AVPLAY\_FlushStream: Notifies the AVPLAY of the completion of stream transmission.
- I-frame decoding. The I frames that are separately transmitted can be decoded and frame information after decoding can be captured. The following APIs are provided:
  - HI\_UNF\_AVPLAY\_DecodeIFrame: Decodes I frames.
  - HI\_UNF\_AVPLAY\_ReleaseIFrame: Releases the frame buffer during I-frame decoding.
- External VDEC and ADEC functions. The following APIs are provided:
  - HI\_UNF\_AVPLAY\_RegisterAcodecLib: Registers the ADEC.
  - HI\_UNF\_AVPLAY\_RegisterVcodecLib: Registers the VDEC.
- User data processing. User data in streams can be processed. The following APIs are provided:
  - HI\_UNF\_AVPLAY\_AcqUserData: Obtains user data.
  - HI\_UNF\_AVPLAY\_RlsUserData: Releases the buffer storing user data.

Figure 4-2 shows the internal logical block diagram of the AVPLAY module by using the ES data playback as an example.



**Figure 4-2** Logical block diagram of the AVPLAY module in ES mode



The AVPLAY module consists of the following parts:

- ADEC and VDEC
- Synchronous playback control module
- Command and status control module

The command and status control module responds to the external control commands and queries and reports the status of the AVPLAY module so that the decoded frames can be output through the synchronous play control module.

## 4.4 Development Guide

The AVPLAY module is used in the following scenarios:

- DVB playback
- TS playback
- ES playback
- I-frame decoding
- Multi-track playback
- External audio and video decoding libraries
- Playback control
- Last frame output
- Attribute configuration
- Event processing
- Low delay



## 4.4.1 DVB Playback

### Scenario

The entire DVB playback process needs to be implemented, including searching the program list, locking the frequency, demultiplexing, decoding, and playing. APIs of the tuner, DEMUX, AVPLAY, VO, AO, and display modules at the UNF layer must be called to support the services.

### Working Process

To implement the DVB playback process, perform the following steps:

- Step 1** Initialize and configure required devices, such as the DISPLAY device, VO device, SOUND device, and DEMUX.
- Step 2** Initialize the tuner and lock the frequency.
- Step 3** Search for programs by using the DEMUX module.
- Step 4** Initialize the AVPLAY device by calling `HI_UNF_AVPLAY_Init`.
- Step 5** Obtain the default configuration of the AVPLAY by calling `HI_UNF_AVPLAY_GetDefaultConfig`. Note that you need to set the stream type to `HI_UNF_AVPLAY_STREAM_TYPE_TS`.
- Step 6** Modify stream attributes as required.
- Step 7** Create an AVPLAY by calling `HI_UNF_AVPLAY_Create`.
- Step 8** Enable the audio/video channels by calling `HI_UNF_AVPLAY_ChnOpen`.
- Step 9** Set the attributes of the ADEC, attributes of the VDEC, audio/video PIDs, and synchronization attributes by calling `HI_UNF_AVPLAY_SetAttr`.
- Step 10** Obtain the default track attributes by calling `HI_UNF_SND_GetDefaultTrackAttr`.
- Step 11** Create a track by calling `HI_UNF_SND_CreateTrack`.
- Step 12** Bind the audio player to the track by calling `HI_UNF_SND_Attach`.
- Step 13** Bind the video player to a VO window by calling `HI_UNF_VO_AttachWindow`.
- Step 14** Enable the window by calling `HI_UNF_VO_SetWindowEnable`.
- Step 15** Send a start command to start to play audio or videos by calling `HI_UNF_AVPLAY_Start`.
- Step 16** After playback is complete, stop the AVPLAY by calling `HI_UNF_AVPLAY_Stop`, and then destroy the AVPLAY to release resources by calling `HI_UNF_AVPLAY_Destroy`.
- Step 17** Disable related devices, such as the DISPLAY device, VO device, SOUND device, DEMUX, and AVPLAY.

----End

### Notes

Do not overlay the video layer with the graphics layer when setting the layer attributes of the DISPLAY device.



## Sample

See sample\dvbplay\dvbplay.c.

## 4.4.2 TS Playback

### Scenario

The entire TS playback process needs to be implemented, including searching the program list, locking the frequency, demultiplexing, decoding, and playing. APIs of the tuner, DEMUX, AVPLAY, VO, AO, and display modules at the UNF layer must be called to support this service.

### Working Process

To implement the entire TS playback process, perform the following steps:

- Step 1** Initialize and configure required devices, such as the DISPLAY device, VO device, SOUND device, and DEMUX.
- Step 2** Create a thread for transmitting TSs.
- Step 3** Search for programs by using the DEMUX module.
- Step 4** Initialize the AVPLAY device by calling `HI_UNF_AVPLAY_Init`.
- Step 5** Obtain the default configuration of the AVPLAY by calling `HI_UNF_AVPLAY_GetDefaultConfig`. Note that you need to set the stream type to `HI_UNF_AVPLAY_STREAM_TYPE_TS`.
- Step 6** Modify stream attributes as required.
- Step 7** Create an AVPLAY by calling `HI_UNF_AVPLAY_Create`.
- Step 8** Enable the audio/video channels by calling `HI_UNF_AVPLAY_ChnOpen`.
- Step 9** Set the attributes of the ADEC, attributes of the VDEC, audio/video PIDs, and synchronization attributes by calling `HI_UNF_AVPLAY_SetAttr`.
- Step 10** Set the attributes of the ADEC or VDEC and set the synchronization mode to free playing by calling `HI_UNF_AVPLAY_SetAttr`.
- Step 11** Create a track by calling `HI_UNF_SND_CreateTrack`.
- Step 12** Bind the audio player to the track by calling `HI_UNF_SND_Attach`.
- Step 13** Bind the video player to a VO window by calling `HI_UNF_VO_AttachWindow`.
- Step 14** Enable the window by calling `HI_UNF_VO_SetWindowEnable`.
- Step 15** Send a start command to start to play audio or videos by calling `HI_UNF_AVPLAY_Start`.
- Step 16** After playback is complete, stop the AVPLAY by calling `HI_UNF_AVPLAY_Stop`, and then destroy the AVPLAY to release resources by calling `HI_UNF_AVPLAY_Destroy`.
- Step 17** Destroy the thread for transmitting TSs.
- Step 18** Disable related devices, such as the DISPLAY device, VO device, SOUND device, DEMUX, and AVPLAY.

----End



## Notes

Do not overlay the video layer with the graphics layer when setting the layer attributes of the DISPLAY device.

## Sample

See sample\tsplay\tsplay.c.

## 4.4.3 ES Playback

### Scenario

The entire ES playback process needs to be implemented, including reading, decoding, and playing ESs. APIs of the AVPLAY, VO, AO, and display modules at the UNF layer must be called to support this service.

### Working Process

To implement the entire ES playback process, perform the following steps:

- Step 1** Initialize and configure required devices, such as the DISPLAY device, VO device, SOUND device, and DEMUX.
- Step 2** Initialize the AVPLAY device by calling HI\_UNF\_AVPLAY\_Init.
- Step 3** Obtain the default configuration of the AVPLAY by calling HI\_UNF\_AVPLAY\_GetDefaultConfig. Note that you need to set the stream type to **HI\_UNF\_AVPLAY\_STREAM\_TYPE\_ES**.
- Step 4** Modify stream attributes as required.
- Step 5** Create an AVPLAY by calling HI\_UNF\_AVPLAY\_Create.
- Step 6** Enable the audio channel or video channel by calling HI\_UNF\_AVPLAY\_ChnOpen.
- Step 7** Set the attributes of the ADEC or VDEC and set the synchronization mode to free playing by calling HI\_UNF\_AVPLAY\_SetAttr.
- Step 8** Create a track by calling HI\_UNF\_SND\_CreateTrack.
- Step 9** Bind the audio player to the track by calling HI\_UNF\_SND\_Attach.
- Step 10** Bind the video player to a VO window by calling HI\_UNF\_VO\_AttachWindow.
- Step 11** Enable the window by calling HI\_UNF\_VO\_SetWindowEnable.
- Step 12** Send a start command to start to play audio or videos by calling HI\_UNF\_AVPLAY\_Start.
- Step 13** Obtain the buffer for storing audio or videos by calling HI\_UNF\_AVPLAY\_GetBuf.
- Step 14** Write the audio or video data to the buffer by copying data from the memory or reading data from files.
- Step 15** Update the read and write pointers in the audio and video buffers by calling HI\_UNF\_AVPLAY\_PutBuf.
- Step 16** After playback is complete, stop the AVPLAY by calling HI\_UNF\_AVPLAY\_Stop, and then destroy the AVPLAY to release resources by calling HI\_UNF\_AVPLAY\_Destroy.



- Step 17** Disable related devices, such as the DISPLAY device, VO device, SOUND device, DEMUX, and AVPLAY.

----End

## Notes

Do not overlay the video layer with the graphics layer when setting the layer attributes of the DISPLAY device.

## Sample

See sample\esplay\esplay.c.

## 4.4.4 I-Frame Decoding

### Scenario

I frames need to be decoded, displayed, and then returned to users.

### Working Process

To decode I frames, perform the following steps:

- Step 1** Initialize and configure required devices, such as the DISPLAY device, VO device, SOUND device, and DEMUX.
- Step 2** Initialize the AVPLAY device by calling HI\_UNF\_AVPLAY\_Init.
- Step 3** Obtain the default configuration of the AVPLAY by calling HI\_UNF\_AVPLAY\_GetDefaultConfig. Note that you need to set the stream type to **HI\_UNF\_AVPLAY\_STREAM\_TYPE\_ES**.
- Step 4** Modify stream attributes as required.
- Step 5** Create an AVPLAY by calling HI\_UNF\_AVPLAY\_Create.
- Step 6** Enable the audio channel or video channel by calling HI\_UNF\_AVPLAY\_ChnOpen.
- Step 7** Set the attributes of the ADEC or VDEC and set the synchronization mode to free playing by calling HI\_UNF\_AVPLAY\_SetAttr.
- Step 8** Create a track by calling HI\_UNF\_SND\_CreateTrack.
- Step 9** Bind the audio player to the track by calling HI\_UNF\_SND\_Attach.
- Step 10** Bind the video player to a VO window by calling HI\_UNF\_VO\_AttachWindow.
- Step 11** Enable the window by calling HI\_UNF\_VO\_SetWindowEnable.
- Step 12** Read the I-frame file.
- Step 13** Transmit and decode I-frame data by calling HI\_UNF\_AVPLAY\_DecodeIFrame.
- Step 14** Process the returned I-frame information, for example, send the frames to the graphics layer for display.
- Step 15** Release the I-frame buffer by calling HI\_UNF\_AVPLAY\_ReleaseIFrame. (If the data is directly sent to the VO device, you do not need to call this API. See the following notes.)



**Step 16** After I frames are decoded, stop the AVPLAY by calling `HI_UNF_AVPLAY_Stop`, and then destroy the AVPLAY to release resources by calling `HI_UNF_AVPLAY_Destroy`.

**Step 17** Disable related devices, such as the DISPLAY device, VO device, SOUND device, and AVPLAY.

----End

## Notes

The AVPLAY can decode static I-frame images and directly display the decoded data in the video window to which the decoder is bound, or extract decoded frame data for special processing.

If you want to directly display the decoded data in the video window to which the decoder is bound, stop the AVPLAY and call `HI_UNF_AVPLAY_DecodeIFrame`.

If you want to extract decoded frame data for special processing, call `HI_UNF_AVPLAY_DecodeIFrame` after stopping the AVPLAY, process the obtained frame data, and then call `HI_UNF_AVPLAY_ReleaseIFrame` to release resources.

## Sample

See `sample\iframe_dec\sample_iframe_dec.c`.

## 4.4.5 Multi-Track Playback

### Scenario

Multi-track streams are played and tracks are smoothly switched.

### Working Process

The following is the process for playing local multi-track TS files.

**Step 1** Initialize and configure required devices, such as the DISPLAY device, VO device, SOUND device, and DEMUX.

**Step 2** Create a thread for transmitting TSs.

**Step 3** Search for programs by using the DEMUX module.

**Step 4** Initialize the AVPLAY device by calling `HI_UNF_AVPLAY_Init`.

**Step 5** Obtain the default configuration of the AVPLAY by calling `HI_UNF_AVPLAY_GetDefaultConfig`. Note that you need to set the stream type to `HI_UNF_AVPLAY_STREAM_TYPE_TS`.

**Step 6** Modify stream attributes as required.

**Step 7** Create an AVPLAY by calling `HI_UNF_AVPLAY_Create`.

**Step 8** Enable the audio/video channels by calling `HI_UNF_AVPLAY_ChnOpen`.

**Step 9** Set the attributes of the ADEC, attributes of the VDEC, audio/video PIDs, and synchronization attributes by calling `HI_UNF_AVPLAY_SetAttr`.

**Step 10** Set the attributes of the ADEC or VDEC and set the synchronization mode to free playing by calling `HI_UNF_AVPLAY_SetAttr`.



- Step 11** Set multi-track attributes by calling HI\_UNF\_AVPLAY\_SetAttr.
- Step 12** Create a track by calling HI\_UNF\_SND\_CreateTrack.
- Step 13** Bind the audio player to the track by calling HI\_UNF\_SND\_Attach.
- Step 14** Bind the video player to a VO window by calling HI\_UNF\_VO\_AttachWindow.
- Step 15** Enable the window by calling HI\_UNF\_VO\_SetWindowEnable.
- Step 16** Send a start command to start to play audios or videos by calling HI\_UNF\_AVPLAY\_Start.
- Step 17** Set the audio PID and switch the track by calling HI\_UNF\_AVPLAY\_SetAttr.
- Step 18** After playback is complete, stop the AVPLAY by calling HI\_UNF\_AVPLAY\_Stop, and then destroy the AVPLAY to release resources by calling HI\_UNF\_AVPLAY\_Destroy.
- Step 19** Destroy the thread for transmitting TSs.
- Step 20** Disable related devices, such as the DISPLAY device, VO device, SOUND device, DEMUX, and AVPLAY.

----End

## Notes

You do not need to stop the AVPLAY when switching the track. You can switch to the track with corresponding PID by setting HI\_UNF\_AVPLAY\_ATTR\_ID\_AUD\_PID.

For the same AVPLAY, if the multi-track playback is switched to single-track playback, the multi-track attributes need to be reconfigured. That is, you need to set the track number to 1 and set the correct audio PID.

## Sample

See sample\tsplay\tsplay\_multiaud.c.

## 4.4.6 External Audio and Video Decoding Libraries

### Scenario

The AVPLAY does not contain inherent ADEC. Therefore, you need to provide an ADEC by registration. In this way, various audio decoding protocols are supported. However, the audio decoding libraries must be consistent with the format provided by HiSilicon.

External video libraries must be adapted based on the definitions in **hi\_video\_codec.h**. The external decoder has priority over the hardware decoder. If a protocol is not supported by the hardware decoder, the system automatically switches to the software decoder for decoding.

### Working Process

Take the external audio decoding libraries as an example. The HiSilicon SDK allows you to dynamically mount audio libraries to meet differentiated design requirements. The audio CODEC libraries that meet the HiSilicon standards defined in **hi\_audio\_codec.h** can be seamlessly registered with the SDK. There are three scenarios:

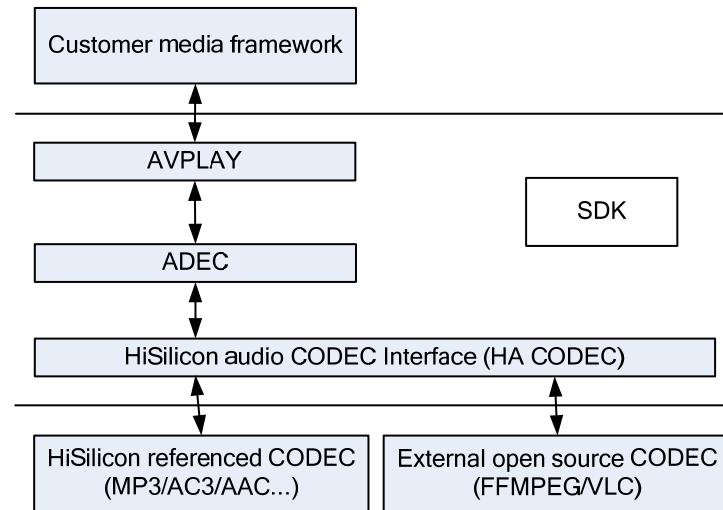
- Replace the HiSilicon reference decoder with customized audio decoder without modifying the HiSilicon SDK.



- Add a new audio decoder without modifying the HiSilicon SDK.
- Add a private audio decoder without modifying the HiSilicon SDK.

Figure 4-3 shows the relationship between the external audio decoding library and the SDK.

**Figure 4-3** Relationship between the external audio decoding library and the SDK



This section describes the interfaces of the external audio library by taking the external .mp3 decoder as an example, as shown in Figure 4-4. For details, see the code of `hi_audio_codec.h` in the *HMS API Development Reference.chm*. The external encoder and external decoder are used in the same way.

- `szName`  
Abbreviation of a CODEC name.
- `enCodecID`  
Unique ID of a CODEC. After a CODEC is registered with the SDK, the ID is its unique identifier. The SDK does not allow duplicate CODEC IDs.
- `uVersion`  
CODEC version. The SDK does not check this parameter currently.
- `pszDescription`  
Description of a CODEC.
- `DecInit`  
This interface is used to create a decoder.
- `DecDeInit`  
This interface is used to destroy a decoder.
- `DecSetConfig`  
This interface is used to set the commands of a decoder. You can extend private commands of the decoder without modifying the SDK.
- `DecGetMaxPcmOutSize`  
This interface is used to obtain the size of the buffer for the decoded audio pulse code modulation (PCM) output. In the SDK, the PCM output buffer is allocated based on the



maximum memory required by each audio PCM frame. If PCM output after decoding is not supported, the value **0** is returned.

- DecGetMaxBitsOutSize

This interface is used to obtain the size of the buffer for outputting decoded audio IEC61937 frames (AC3/DTS transparent transmission). In the SDK, the transparent transmission output buffer is allocated based on the maximum memory required by each transparently transmitted audio frame. If the transparent transmission output is not supported, the value **0** is returned.

**Figure 4-4** Sample of an external .mp3 audio library

```
#ifdef HA_AUDIO_STATIC_LINK_MODE
HI_HA_DECODE_S g_ha_audio_mp3_decode_entry = {
#else
HI_HA_DECODE_S ha_audio_decode_entry = {
#endif
    .szName      = (const HI_PCHAR )"mp3",
    .enCodecID   = HA_AUDIO_ID_MP3,
    .uVersion.u32Version =
                    0x10000001,
    .pszDescription = (const HI_PCHAR)"hisilicon mp3 decoder",
    .DecInit     = HA_MP3DecInit,
    .DecDeInit   = HA_MP3DecDeInit,
    .DecSetConfig = HA_SetConfig,
    .DecGetMaxPcmOutSize = HA_MP3DecGetMaxPcmOutSize,
    .DecGetMaxBitsOutSize = HA_MP3DecGetMaxBitsOutSize,
    .DecDecodeFrame = HA_MP3DecDecodeFrame,
}
```

Each audio CODEC has a unique ID.

## Notes

- The external audio decoding library is independent of the SDK. All audio decoding libraries can be used only after being registered.
- The external ADEC must be set correctly.
- Currently the external video decoding libraries of the AVPLAY module are provided only for internal components.

## Sample

See **sample\ha\_libmad\src\ha\_libmad\_wrapper.c**.

This sample describes the development of the external MP3 decoding libraries by using the famous MP3 decoding open source software libmad as an example.

## 4.4.7 Playback Control

### Scenario

Controls the playback by performing operations such as starting, stopping, pausing, trick playing, resuming, or resetting.



## Working Process

The working process for playback control is similar to those for DVB playback and TS playback. The only difference is that playback control interfaces are called during playback to operate the AVPLAY.

## Notes

When using the playback control function of the AVPLAY, pay attention to the switchover between playback states. The AVPLAY has five playback states. You can switch the states by performing corresponding operations.

**Table 4-1** Playback status control

Status	Stop	Play	Tplay	Pause	EOS
Stop	-	Start	X	X	X
Play	Stop	-	Tplay	Pause	Complete the stream playback.
Tplay	Stop	Resume	-	Pause	Complete the stream playback.
Pause	Stop	Resume	Tplay	-	X
EOS	Stop	-	-	-	-

In **Table 4-1**, the first column indicates start status, and the first row indicates end status. The intersection of each column and each row indicates the operation to be performed for switching a start status to an end status. In addition, the mark "-" indicates invalid operation, and the mark "X" indicates that a start status cannot be switched to an end status.

## Sample

See `sample\tsplay\tsplay.c`.

### 4.4.8 Last Frame Output

#### Scenario

This function applies to the following scenarios:

- The last frame must be played during stream playback.
- The user wants to know whether the stream playback is complete so that other operations can be performed.

## Working Process

Perform the following steps:



- Step 1** Initialize and configure required devices, such as the DISPLAY device, VO device, SOUND device, and DEMUX.
- Step 2** Initialize the AVPLAY device by calling `HI_UNF_AVPLAY_Init`.
- Step 3** Obtain the default configuration of the AVPLAY by calling `HI_UNF_AVPLAY_GetDefaultConfig`. Note that you need to set the stream type to `HI_UNF_AVPLAY_STREAM_TYPE_ES`.
- Step 4** Modify stream attributes as required.
- Step 5** Create an AVPLAY by calling `HI_UNF_AVPLAY_Create`.
- Step 6** Enable the audio channel or video channel by calling `HI_UNF_AVPLAY_ChnOpen`.
- Step 7** Set the attributes of the ADEC or VDEC and set the synchronization mode to free playing by calling `HI_UNF_AVPLAY_SetAttr`.
- Step 8** Register the `HI_UNF_AVPLAY_EVENT_EOS` event by calling `HI_MPI_AVPLAY_RegisterEvent`.
- Step 9** Create a track by calling `HI_UNF_SND_CreateTrack`.
- Step 10** Bind the audio player to the track by calling `HI_UNF_SND_Attach`.
- Step 11** Bind the video player to a VO window by calling `HI_UNF_VO_AttachWindow`.
- Step 12** Enable the window by calling `HI_UNF_VO_SetWindowEnable`.
- Step 13** Send a start command to start to play audios or videos by calling `HI_UNF_AVPLAY_Start`.
- Step 14** Obtain the buffer for storing audio or videos by calling `HI_UNF_AVPLAY_GetBuf`.
- Step 15** Write the audio or video data to the buffer by copying data from the memory or reading data from files.
- Step 16** Update the read and write pointers in the audio or video buffer by calling `HI_UNF_AVPLAY_PutBuf`.
- Step 17** Call `HI_UNF_AVPLAY_FlushStream` after streams are transmitted.
- Step 18** Check whether the `HI_UNF_AVPLAY_EVENT_EOS` event is reported in the event callback function. If it is reported, the last frame is played.
- Step 19** After playback is complete, stop the AVPLAY by calling `HI_UNF_AVPLAY_Stop`, and then destroy the AVPLAY to release resources by calling `HI_UNF_AVPLAY_Destroy`.
- Step 20** Disable related devices, such as the DISPLAY device, VO device, SOUND device, DEMUX, and AVPLAY.

----End

## Notes

After `HI_UNF_AVPLAY_FlushStream` is called, if there are still streams to be transmitted, `HI_UNF_AVPLAY_EVENT_EOS` is not reported.

After `HI_UNF_AVPLAY_FlushStream` is called and before the EOS event is reported, do not call other playback control APIs such as `HI_UNF_AVPLAY_Reset`. Otherwise, the EOS event cannot be reported.



## Sample

None

## 4.4.9 Attribute Configuration

### Scenario

The AVPLAY has various attributes. To properly use the AVPLAY, you must know about the usage and limitations of each attribute.

### Working Process

The working process for attribute configuration is similar to those for DVB playback and TS playback. The only difference is that corresponding interfaces are called after the AVPLAY is created or during playback to set attributes.

### Notes

There are some limitations when you set the AVPLAY attributes, and the limitations vary according to attributes. [Table 4-2](#) describes the limitations.

**Table 4-2** AVPLAY attributes

Attribute ID	Description	Modification Limitation
HI_UNF_AVPLAY_ATTR_ID_STREAM_MODE	Playback mode	The mode can be changed after an AVPLAY is created and the audio/video channels are disabled.
HI_UNF_AVPLAY_ATTR_ID_ADEC	Audio attribute	The audio attributes can be changed only when an audio channel is enabled but audio files are not played.
HI_UNF_AVPLAY_ATTR_ID_VDEC	Video attribute	The video attributes except the video encoding type can be set when a video channel is working. The video encoding type can be changed only when a video channel is enabled but videos are not played.
HI_UNF_AVPLAY_ATTR_ID_AUD_PID	Audio PID	The audio PID can be changed only when an audio channel is enabled.
HI_UNF_AVPLAY_ATTR_ID_VID_PID	Video PID	The video PID can be changed only when a video channel is enabled but videos are not played.
HI_UNF_AVPLAY_ATTR_ID_PCR_PID	PCR PID	The PCR PID can be set only after an AVPLAY is stopped.
HI_UNF_AVPLAY_ATTR_ID_SYNC	Synchronization attribute	The synchronization attributes can be changed after an AVPLAY is created.



Attribute ID	Description	Modification Limitation
HI_UNF_AVPLAY_ATTR_ID_OVERFLOW	Processing for channel buffer overflow	This attribute can be modified after a player is created and the audio/video channels are disabled.
HI_UNF_AVPLAY_ATTR_ID_MULTIAUD	Multi-track attributes	The multi-track attributes can be changed after a player is created and the audio/video channels are disabled.
HI_UNF_AVPLAY_ATTR_ID_FRMRATE_PARAM	Frame rate parameters	The frame rate parameters can be modified after a video channel is enabled.
HI_UNF_AVPLAY_ATTR_ID_FRMPACK_TYPE	3D frame pack type attributes	The 3D frame pack type attributes can be configured after the video channel is enabled.
HI_UNF_AVPLAY_ATTR_ID_LOW_DELAY	Low delay attribute of the player	The low-delay attribute of the player can be configured after the player is created.

- Stream attributes

You can obtain the default attributes of a player by calling `HI_UNF_AVPLAY_GetDefaultConfig` and specify the player type.

Typically default attributes apply to most audio/video playback scenarios. However, you are advised to change the values of `u32VidBufSize` and `u32AudBufSize` based on actual requirements (such as the bit rates of audio/video streams).

**Table 4-3** Stream attributes

Attribute Descriptor	Default Configuration	Description
u32DemuxId	0	ID of the DEMUX used by the player. This attribute is valid only in TS mode.
enStreamType	None	Default stream attributes are obtained by stream type. That is, you need to enter a stream type to obtain default stream attributes.
u32VidBufSize	5 MB	Size of the video PES (ES) buffer
u32AudBufSize	384 KB (in TS mode) or 256 KB (in ES mode)	Size of the audio PES (ES) buffer

- VDEC attributes



The VDEC can decode the videos in the format of H.264, AVS, MPEG-2, or MPEG-4. You need to set the video decoding attributes by calling `HI_UNF_AVPLAY_SetAttr`. The AVPLAY performs decoding based on the configured decoding type.

[Table 4-4](#) describes the attributes of the VDEC.

**Table 4-4** VDEC attributes

Attribute Descriptor	Default Configuration	Description
enType	None	Decoding type of the VDEC. This attribute is mandatory.
unExtAttr	None	Extra attribute of the VDEC, VC1 or VP6 decoding type. This attribute is mandatory.
enMode	<code>HI_UNF_VCODEC_MODE_NORMAL</code>	Decoding mode of the VDEC. <ul style="list-style-type: none"><li>• <code>HI_UNF_VCODEC_MODE_NORMAL</code>: Normal decoding mode.</li><li>• <code>HI_UNF_VCODEC_MODE_IP</code> : I frames and P frames are decoded.</li><li>• <code>HI_UNF_VCODEC_MODE_I</code>: Only I frames are decoded.</li><li>• <code>HI_UNF_VCODEC_MODE_DROP_INVALID_B</code>: All frames (except B frames that closely follow the I frames) are decoded.</li></ul>
u32ErrCover	100	Error concealment threshold for the output frames of the VDEC. Its value ranges from 0 to 100 (in percentage). <ul style="list-style-type: none"><li>• 0: No frames are output in the case of errors.</li><li>• 100: Frames are output even the error percentage is 100%.</li></ul>
u32Priority	0	VDEC priority. The value ranges from 1 to <code>HI_UNF_VCODEC_MAX_PRIORITY</code> . The value <b>0</b> is a reserved value. If you set the value to <b>0</b> , no error message is displayed, but the value <b>1</b> is used automatically. A smaller value indicates a lower priority.
bOrderOutput	<code>HI_FALSE</code>	Whether the videos are output by the decoding sequence. You are advised to set this parameter to <code>HI_TRUE</code> in VP mode, and



Attribute Descriptor	Default Configuration	Description
		<b>HI_FALSE</b> in other modes.
s32CtrlOptions	0	Special control options of the VDEC.
pCodecContext	HI_NULL	Private commands when an external decoder is used.

- Synchronization attributes

The AVPLAY supports the control of audio/video synchronization. You can select the synchronization mode and set related attributes by calling `HI_UNF_AVPLAY_SetAttr`.

The following features are supported:

- Audio-based synchronization, PCR-based synchronization, and free playing
- Configurable ranges for the synchronization start region and synchronization exception region
- Configurable speed adjustment for the synchronization start region and synchronization exception region
- Whether to enable pre-synchronization. The pre-synchronization time can be configured, and fast synchronization is used.
- Fast output when pre-synchronization is enabled
- Automatic switchover from PCR-based synchronization to audio-based synchronization if the PCR is invalid
- Fixed deviation regulation of audio/video PTSs

The free playing without synchronization is designed for debugging, which prevents playback errors due to synchronization.

**Table 4-5** Audio/Video synchronization attributes

Attribute Descriptor	Default Configuration	Description
enSyncRef	None	Synchronization mode
s32VidPtsAdjust	None	Adjustment value of the video PTS. All video PTSs are adjusted after being added with the adjustment value. The value is typically set to <b>0</b> .
s32AudPtsAdjust	None	Adjustment value of an audio PTS. All audio PTSs are adjusted after being added with the adjustment value. The value is generally set to <b>0</b> .
u32PreSyncTimeoutMs	None	Pre-synchronization



Attribute Descriptor		Default Configuration	Description
			timeout (in ms). The value <b>0</b> indicates no pre-synchronization.
bQuickOutput		None	Fast output enable
stSyncStartRegion	s32VidPlusTime	None	Plus time range during video synchronization
	s32VidNegativeTime	None	Negative time range during video synchronization
	bSmoothPlay	None	Slow synchronization enable in the case of exception. You are advised to enable this function.
stSyncNovelRegion	s32VidPlusTime	None	Plus time range in the case of video exception
	s32VidNegativeTime	None	Negative time range in the case of video exception
	bSmoothPlay	None	Slow synchronization enable in the case of critical exceptions. You are advised to enable this function.

The maximum difference value that can be regulated for audio/video synchronization depends on the audio/video buffer size and audio/video bit rate. It is in proportion to the buffer size while in inverse proportion to the bit rate. In addition, it is defined in the SDK that if the audio/video PTS difference is greater than 10s, the synchronization adjustment is aborted. You must specify an appropriate buffer size when creating an AVPLAY to obtain the expected maximum PTS difference for audio/video synchronization.

## Sample

None

## 4.4.10 Event Processing

### Scenario

During playback, various events related to the playback are reported. You can know about the playback status of the AVPLAY from these events or even control upper-layer software to optimize the playback effect by using these events properly.



## Working Process

The event processing process is similar to the DVB playback/ES playback process. The main difference is that after the AVPLAY is created, HI\_UNF\_AVPLAY\_RegisterEvent is called to register the callback function, and the playback events can be processed in the callback function.

## Notes

The following describes the event mechanism by using the buffer status event and sync event as an example:

- Buffer status event

The buffer status event can be used to notify the upper-layer software of the audio/video buffer status in the SDK. The data flow speed can be controlled by the upper-layer software based on the buffer status. When the buffer is full, the flow speed is slowed; when the buffer is empty, the flow speed is increased. In this way, the playback effect is optimized.

- Sync event

The sync event can be used to notify the upper-layer software of the audio/video sync status in the SDK. When the sync module detects the PTS jump or sync status change information, it reports to the user. Then the user can optimize the playback effect by using some method (such as pausing sync for two seconds and starting sync again). In this way, the playback effect is optimized when the PTS error occurs.

## Sample

None

## 4.4.11 Low Delay

### Scenario

In the video phone or cloud game scenario, you need to set the AVPLAY to the low delay mode to meet playback delay requirements.

## Working Process

The working process for low delay playback is similar to that for the ES playback. The main difference is that HI\_UNF\_AVPLAY\_SetAttr needs to be called to set low delay attributes and enable the low delay mode.

## Notes

During low delay playback, the AVPLAY implements the following:

- Disable synchronization.
- Disable frame rate conversion.
- Set the VDEC to low delay mode. In this mode, the VDEC does not decode B frames. It outputs data based on the decoding sequence and optimizes management of the frame buffer queue.



- Set the VO to fast output mode so that the VO always plays the latest frame in the frame buffer.

## Sample

None



# Contents

---

<b>5 SOUND .....</b>	<b>1</b>
5.1 Overview .....	1
5.2 Important Concepts .....	1
5.3 Features .....	3
5.4 Development Guide .....	5
5.4.1 Playing Audio Data .....	5
5.4.2 Playing External PCM Data .....	8
5.4.3 Capturing Track Data .....	10
5.4.4 Capturing SOUND Output Data .....	12



# Figures

<b>Figure 5-1</b> SOUND module in the system.....	1
<b>Figure 5-2</b> Block diagram of the SOUND device .....	2
<b>Figure 5-3</b> Typical process for playing audio data .....	7
<b>Figure 5-4</b> Typical process for playing external PCM data .....	9
<b>Figure 5-5</b> Typical process for capturing track data .....	11
<b>Figure 5-6</b> Typical process for capturing SOUND output data .....	13



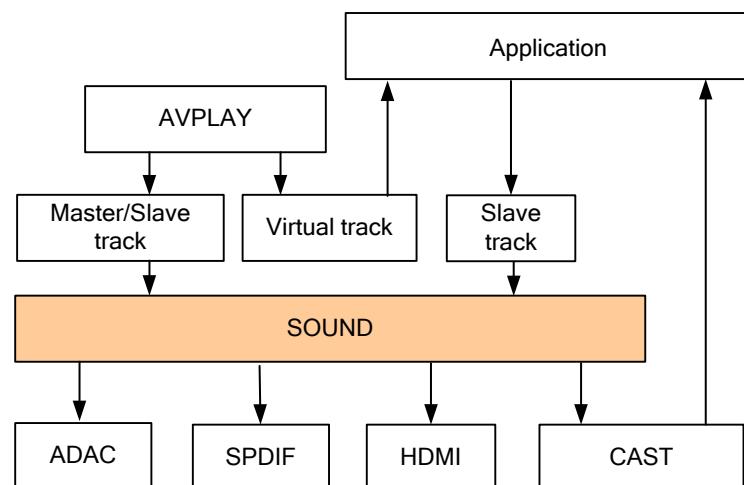
# 5 SOUND

## 5.1 Overview

The SOUND module is an audio adapter provided by HiSilicon. It receives audio data from the AVPLAY or external PCM data and sends the data to the ADAC (CVBS output), SPDIF, and HDMI interfaces. It supports simultaneous multi-channel outputs, bringing rich audio applications.

[Figure 5-1](#) shows the position of the SOUND module in the system.

**Figure 5-1** SOUND module in the system



## 5.2 Important Concepts

[SOUND]

There are three SOUND devices:

- HI\_UNF\_SND\_0
- HI\_UNF\_SND\_1

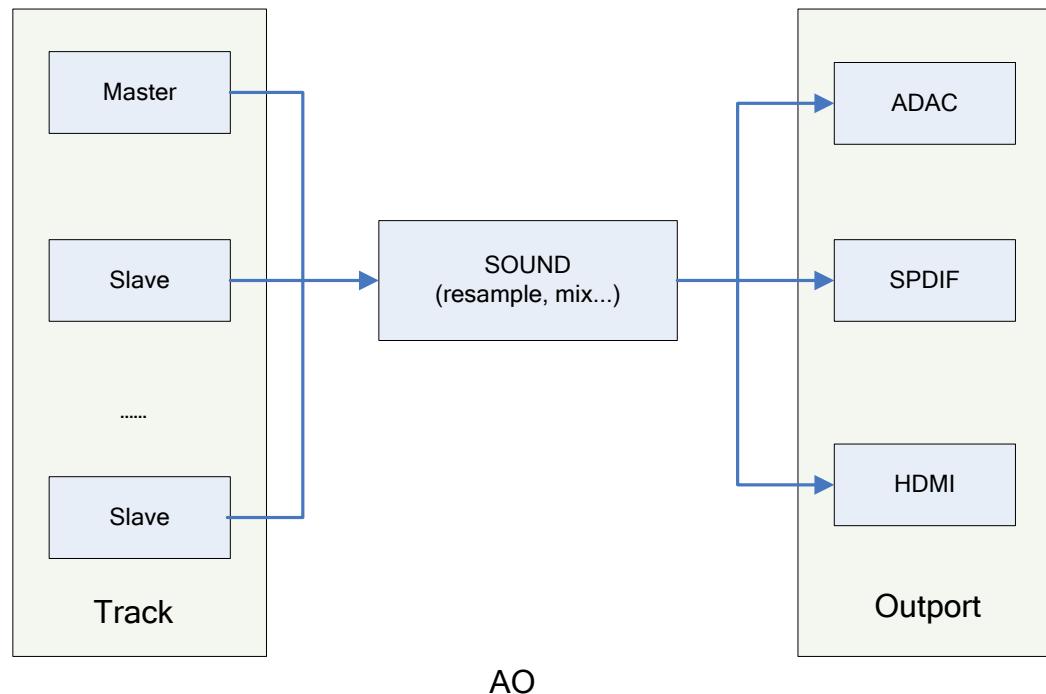


- HI\_UNF\_SND\_2

The SOUND device processes (resampling, audio mixing) data from various tracks and output the processed data to the bound output.

Figure 5-2 shows the block diagram of the SOUND device.

**Figure 5-2** Block diagram of the SOUND device



#### [Track]

The track is the audio data playback channel. There are three types of tracks:

- Master track: track for PCM data and pass-through audio data. The main difference between the master track and other tracks is that the master track supports transparent transmission.
- Slave track: track for PCM 2.0 audio data. The slave track is used as the playback track for the background music, menu tone, keypad tone, and game music, or the audio mixing track for multi-track playback.
- Virtual track: virtual track for PCM 2.0 audio data. Data in the virtual track is stored in a buffer without the audio mixing operation. You can obtain the data by calling the UNF interface for other purposes such as transcoding.

All the SOUND devices support a maximum of six output tracks (one master track/three slave tracks or six slave tracks) and six virtual tracks, and each SOUND device supports only one master track.

#### [Outport]



The outport is the physical output port for audio data, including the ADAC, SPDIF, HDMI, and I<sup>2</sup>S outports. You can set the mute mode, volume, and track mode for the outport.

#### [Cast]

An audio cast outport is virtualized after the audio cast function is enabled. This outport can cast all decoded audio data output by the system. Data casted by the audio cast outport is the 2-channel 16-bit PCM data, and the data sampling rate equals to the input sampling rate when the SOUND device is started, typically 48 kHz.

#### [PCM data and pass-through data]

Audio data is divided into the PCM data and pass-through data. The PCM data indicates the decoded data or raw data that is not encoded. The pass-through data indicates the encoded data that is encapsulated complying with the IEC61937 protocol and output to the SOUND module without decoding. The pass-through data is decoded by peripherals. It can be divided into low-bit-rate (LBR) pass-through data (DD and DTS) and high-bit-rate (HBR) pass-through data (DDP, DTSHD, and TRUEHD). The SOUND module supports the transmission of the PCM data and pass-through data at the same time.

#### [SPDIF mode]

The SPDIF mode indicates the output mode of the SPDIF outport. Two SPDIF modes are supported: PCM 2.0 output and pass-through output.

#### [HDMI mode]

The HDMI mode indicates the output mode for the HDMI outport. Four HDMI modes are supported: 2.0 PCM output, pass-through output, blu-ray next generation output (DDP-to-DD output, DTSHD-to-DTS output), and automatic matched output based on the HDMI EDID capability (if the HDMI interface does not support the pass-through program data, the 2.0 PCM output is used; if the amplifier is connected, the pass-through output is used.)

#### [Track mode]

The track mode can be configured for a single outport or all outports.

#### [Volume]

The track volume (track weight) and outport volume can be configured. You can set the volume of each track separately or the volume of the audio-left or audio-right channel of a track separately. You can also set the volume of a single outport or all outports. Two volume configuration modes are supported: linear volume configuration and non-linear volume configuration. The value range for linear volume configuration is 0 to 100, and that for non-linear volume configuration is -70 dB to 0 dB.

## 5.3 Features

The AO module provides the following functions:

- Initialization and deinitialization. The following APIs are provided:
  - HI\_UNF\_SND\_Init: Initializes the AO module.
  - HI\_UNF\_SND\_DeInit: Deinitializes the AO module.
- SOUND device management, including starting and stopping the SOUND device. The following APIs are provided:
  - HI\_UNF\_SND\_GetDefaultOpenAttr: Obtains the attributes that are enabled for the SOUND device.



- HI\_UNF\_SND\_Open: Starts a SOUND device.
- HI\_UNF\_SND\_Close: Stops a SOUND device.
- Outport management. The following APIs are provided:
  - HI\_UNF\_SND\_SetMute: Sets the outport mute mode.
  - HI\_UNF\_SND\_GetMute: Obtains the outport mute status.
  - HI\_UNF\_SND\_SetHdmiMode: Sets the HDMI output mode
  - HI\_UNF\_SND\_SetSpdifMode: Sets the SPDIF output mode.
  - HI\_UNF\_SND\_SetVolume: Sets the outport volume.
  - HI\_UNF\_SND\_GetVolume: Obtains the outport volume value.
  - HI\_UNF\_SND\_SetDelayCompensationMs: Sets the outport delay.
  - HI\_UNF\_SND\_GetDelayCompensationMs: Obtains the outport delay.
  - HI\_UNF\_SND\_SetAdacEnable: Enables or disables ADAC output.
  - HI\_UNF\_SND\_SetTrackMode: Sets the outport track mode.
  - HI\_UNF\_SND\_GetTrackMode: Obtains the outport track mode.
  - HI\_UNF\_SND\_SetLowLatency: Sets the outport low-delay limitation.
  - HI\_UNF\_SND\_GetLowLatency: Obtains the outport low-delay limitation.
- Track management. The following APIs are provided:
  - HI\_UNF\_SND\_GetDefaultTrackAttr: Obtains the default attributes of the track.
  - HI\_UNF\_SND\_CreateTrack: Creates a track.
  - HI\_UNF\_SND\_DestroyTrack: Destroys a track.
  - HI\_UNF\_SND\_Attach: Binds a track to an AVPLAY.
  - HI\_UNF\_SND\_Detach: Unbinds a track from an AVPLAY.
  - HI\_UNF\_SND\_SetTrackAttr: Sets track attributes.
  - HI\_UNF\_SND\_GetTrackAttr: Obtains track attributes.
  - HI\_UNF\_SND\_SetTrackWeight: Sets the track volume.
  - HI\_UNF\_SND\_GetTrackWeight: Obtains the track volume value.
  - HI\_UNF\_SND\_SetTrackAbsWeight: Sets the volume of audio-left and audio-right channels.
  - HI\_UNF\_SND\_GetTrackAbsWeight: Obtains the volume of audio-left and audio-right channels.
  - HI\_UNF\_SND\_AcquireTrackFrame: Obtains the audio frames in the buffer of a virtual track.
  - HI\_UNF\_SND\_ReleaseTrackFrame: Releases the audio frames in the buffer of a virtual track.
  - HI\_UNF\_SND\_SendTrackData: Sends the external PCM data to a track.
- Cast management. The following APIs are provided:
  - HI\_UNF\_SND\_GetDefaultCastAttr: Obtains the default attributes of the cast.
  - HI\_UNF\_SND\_CreateCast: Creates a cast.
  - HI\_UNF\_SND\_DestroyCast: Destroys a cast.
  - HI\_UNF\_SND\_SetCastEnable: Enables/Disables the cast function.
  - HI\_UNF\_SND\_GetCastEnable: Obtains the cast enable status.
  - HI\_UNF\_SND\_AcquireCastFrame: Obtains the audio frames in the buffer of a cast.
  - HI\_UNF\_SND\_ReleaseCastFrame: Releases the audio frames in the buffer of a cast.



## 5.4 Development Guide

The SOUND devices are used in the following scenarios:

- Playing audio data
- Playing external PCM data
- Capturing track data
- Capturing SOUND output data

The SOUND device receives data from the AVPLAY module and outputs the PCM data from the AVPLAY to the ADAC, SPDIF, and HDMI ports, or transparently transmits the data that is not decoded to the SPDIF and HDMI ports.

### 5.4.1 Playing Audio Data

#### Scenario

In this scenario, the AVPLAY transmits the decoded audio data to the corresponding track. The SOUND device processes data in each track (resampling, audio mixing) and outputs the processed data to the physical port that is bound to the SOUND device. During the output process, you can set the volume, mute mode, and pass-through mode (to implement complete pass-through functions, you also need to set the decoder to the pass-through mode or pass-through+decoding mode) for the output.

#### Working Process

Initialize the AO module before calling other SOUND APIs, and deinitialize the AO module when it is not used.

The initialization procedure is as follows:

- Step 1** Initialize the AO module by calling `HI_UNF_SND_Init`.
- Step 2** Obtain the SOUND attributes that are enabled by default by calling `HI_UNF_DMX_GetDefaultOpenAttr`.
- Step 3** Modify the attributes and start the SOUND device by calling `HI_UNF_SND_Open`.
- Step 4** Set one or more attributes of the SOUND device as required by calling the APIs with the prefix of `HI_UNF_SND_Set`.
- Step 5** Obtain the default track attributes by calling `HI_UNF_SND_GetDefaultTrackAttr`.
- Step 6** Modify track attributes, and create a track by calling `HI_UNF_SND_CreateTrack`. You can create multiple tracks to implement multi-track playback.
- Step 7** Bind the track to an input source (AVPLAY) by calling `HI_UNF_SND_Attach`.
- Step 8** Start the input source (AVPLAY) to output audio data.

You can dynamically set the attributes of the SOUND device by calling the APIs with the prefix of `HI_UNF_SND_Set` during output.

**----End**

The deinitialization procedure is as follows:



**Step 1** Unbind the input source from the SOUND device by calling HI\_UNF\_SND\_Detach.

**Step 2** Destroy the track by calling HI\_UNF\_SND\_DestroyTrack.

**Step 3** Stop the SOUND device by calling HI\_UNF\_SND\_Close.

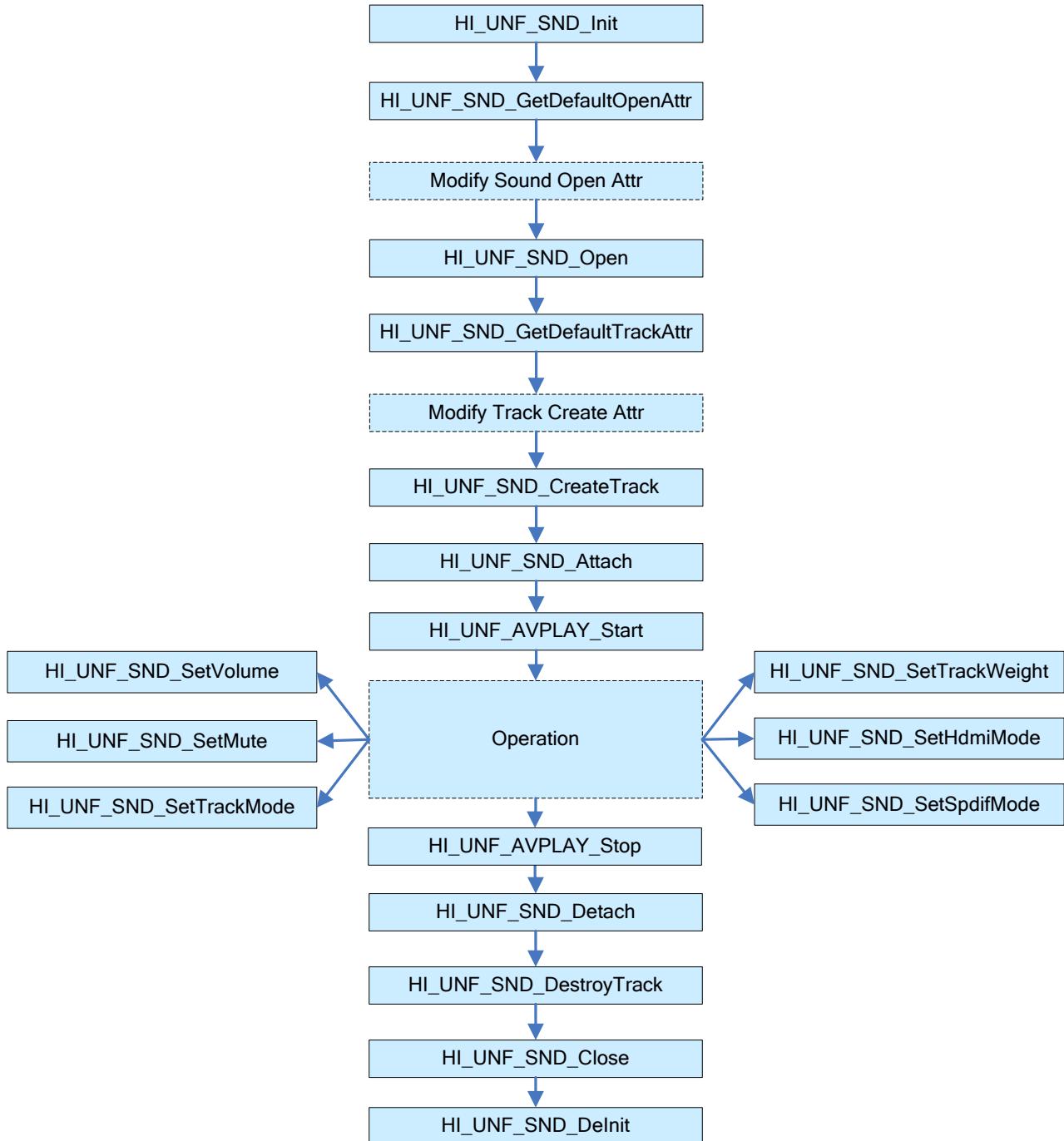
**Step 4** Deinitialize the AO module by calling HI\_UNF\_SND\_DeInit.

----End

[Figure 5-3](#) shows the typical process for playing audio data.



**Figure 5-3** Typical process for playing audio data



## Notes

- You can create multiple tracks at the same time. However, you can create only one master track for each SOUND device.
- If the original master track is in stop state, a new master track can be successfully created. In the meantime, the original master track is switched to the slave track; otherwise, the creation fails.



- Track attributes can be configured. **enTrackType** can be modified only when the created master track is in stop state; otherwise, the setting fails.**u32BufLevelMs** can be set dynamically.**u32FadeinMs**, **u32FadeoutMs**, and **u32OutputBufSize** are reserved.

## Sample

See sample/esplay/esplay.c.

### 5.4.2 Playing External PCM Data

#### Scenario

In this scenario, external PCM data is transmitted to a specific track, processed by the SOUND device, and then output from the physical port that is bound to the SOUND device. This scenario mainly involves the playback of the background music, menu tone, keypad tone, or game music, and the real-time performance is high because the audio decoding module and AVPLAY module are not involved. However, only the external PCM data can be transferred because the audio decoding module is not involved.

#### Working Process

Initialize the AO module before calling other SOUND APIs, and deinitialize the AO module when it is not used.

The initialization procedure is as follows:

- Step 1** Initialize the AO module by calling HI\_UNF\_SND\_Init.
- Step 2** Obtain the SOUND attributes that are enabled by default by calling HI\_UNF\_DMX\_GetDefaultOpenAttr.
- Step 3** Modify the attributes and start the SOUND device by calling HI\_UNF\_SND\_Open.
- Step 4** Obtain the default track attributes by calling HI\_UNF\_SND\_GetDefaultTrackAttr.
- Step 5** Modify the attributes, and create a track by calling HI\_UNF\_SND\_CreateTrack.
- Step 6** Output audio data by calling HI\_UNF\_SND\_SendTrackData repeatedly.

You can dynamically set the attributes of the SOUND device by calling the APIs with the prefix of HI\_UNF\_SND\_Set during output.

----End

The deinitialization procedure is as follows:

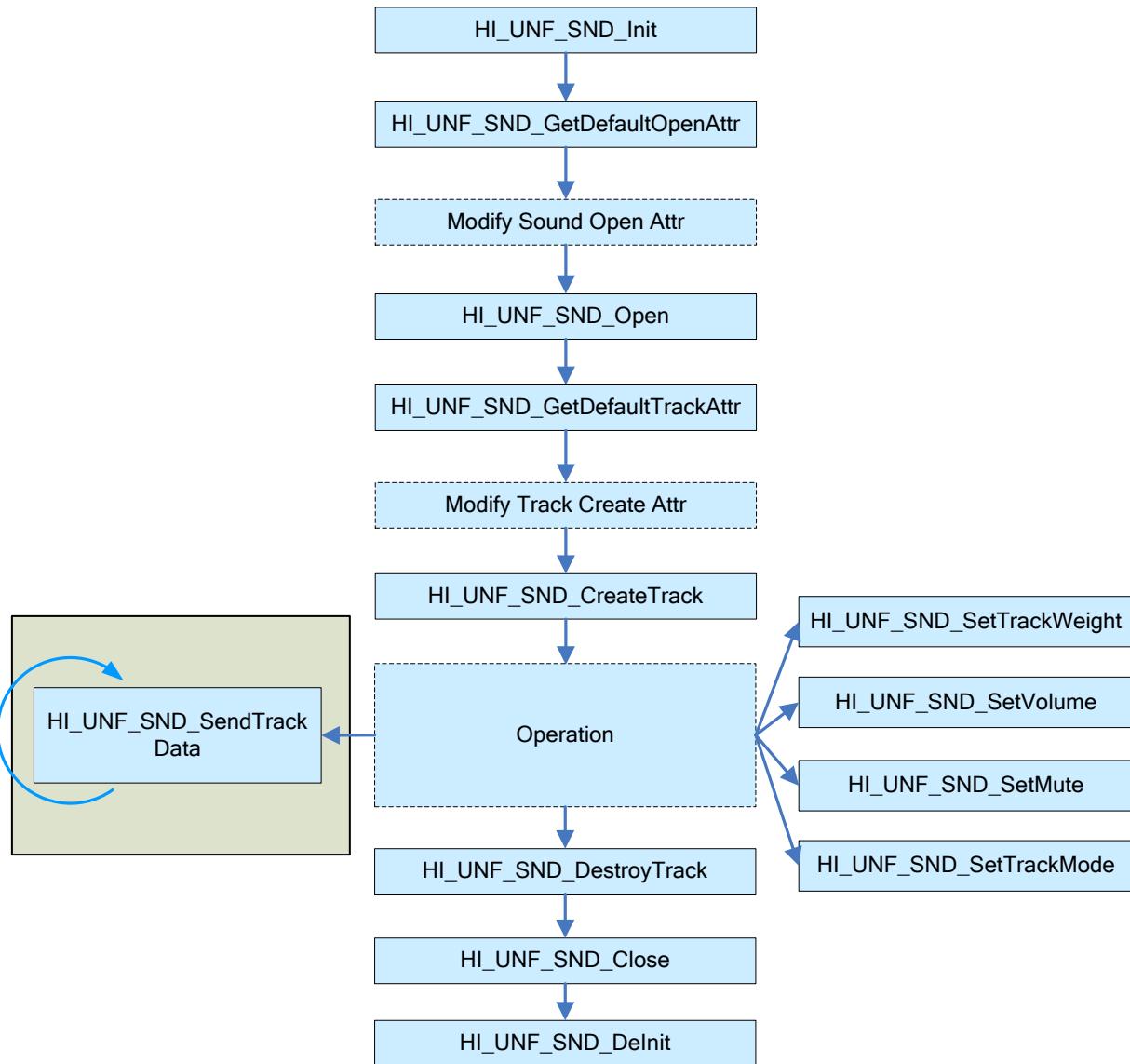
- Step 1** Destroy the track by calling HI\_UNF\_SND\_DestroyTrack.
- Step 2** Stop the SOUND device by calling HI\_UNF\_SND\_Close.
- Step 3** Deinitialize the AO module by calling HI\_UNF\_SND\_DeInit.

----End

[Figure 5-4](#) shows the typical process for playing external PCM data.



**Figure 5-4** Typical process for playing external PCM data



## Notes

- Typically the audio playback scenario and the external PCM data playback scenario can be used together.
- The track created in this scenario is the slave track.

## Sample

See `sample/ao/mix/mixengine.c`.



## 5.4.3 Capturing Track Data

### Scenario

In this scenario, audio data in the track is capture for transcoding (for details about the transcoding sample, see the description of the AENC module).

### Working Process

Initialize the AO module before calling other SOUND APIs, and deinitialize the AO module when it is not used.

The initialization procedure is as follows:

- Step 1** Initialize the AO module by calling HI\_UNF\_SND\_Init.
- Step 2** Obtain the SOUND attributes that are enabled by default by calling HI\_UNF\_DMX\_GetDefaultOpenAttr.
- Step 3** Modify the attributes and start the SOUND device by calling HI\_UNF\_SND\_Open.
- Step 4** Obtain the default track attributes by calling HI\_UNF\_SND\_GetDefaultTrackAttr.
- Step 5** Modify the attributes, and create a virtual track by calling HI\_UNF\_SND\_CreateTrack.
- Step 6** Bind the virtual track to an input source (AVPLAY) by calling HI\_UNF\_SND\_Attach.
- Step 7** Start the input source (AVPLAY).
- Step 8** Obtain and release audio frames by calling HI\_UNF\_SND\_AcquireTrackFrame and HI\_UNF\_SND\_ReleaseTrackFrame repeatedly.

----End

The deinitialization procedure is as follows:

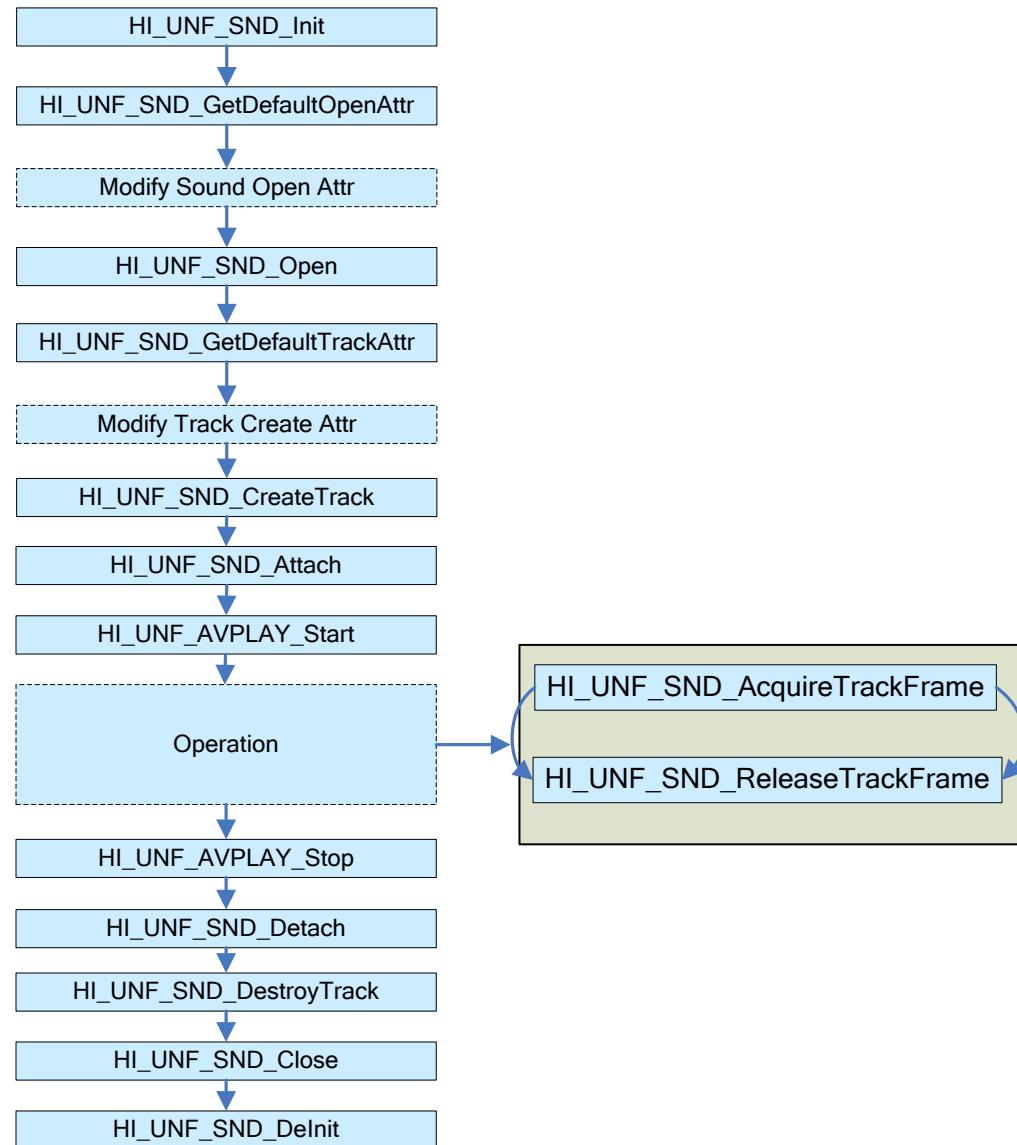
- Step 1** Destroy the virtual track by calling HI\_UNF\_SND\_DestroyTrack.
- Step 2** Stop the SOUND device by calling HI\_UNF\_SND\_Close.
- Step 3** Deinitialize the AO module by calling HI\_UNF\_SND\_DeInit.

----End

[Figure 5-5](#) shows the typical process for capturing track data.



**Figure 5-5** Typical process for capturing track data



## Notes

- You can create a master or slave track besides the virtual track to play the audio while capturing it.
- When a master or slave track and a virtual track are both created, the output of the master or slave track is ensured in priority. Therefore, ensure that data of the virtual track is captured in time to avoid frame loss.

## Sample

See `sample/aenc/aenc_track.c`.



## 5.4.4 Capturing SOUND Output Data

### Scenario

In this scenario, mixed output data from the SOUND device is captured (the cast function) for multi-screen sharing (for details about the multi-screen sharing sample, see the description of the AENC module).

### Working Process

Initialize the AO module before calling other SOUND APIs, and deinitialize the AO module when it is not used.

The initialization procedure is as follows:

- Step 1** Initialize the AO module by calling HI\_UNF\_SND\_Init.
- Step 2** Obtain the SOUND attributes that are enabled by default by calling HI\_UNF\_DMX\_GetDefaultOpenAttr.
- Step 3** Modify the attributes and start the SOUND device by calling HI\_UNF\_SND\_Open.
- Step 4** Obtain the default cast attributes by calling HI\_UNF\_SND\_GetDefaultCastAttr.
- Step 5** Modify the attributes, and create a cast by calling HI\_UNF\_SND\_CreateCast.
- Step 6** Enable the cast function by calling HI\_UNF\_SND\_SetCastEnable.
- Step 7** Obtain and release audio frames by calling HI\_UNF\_SND\_AcquireCastFrame and HI\_UNF\_SND\_ReleaseCastFrame repeatedly.

**----End**

The deinitialization procedure is as follows:

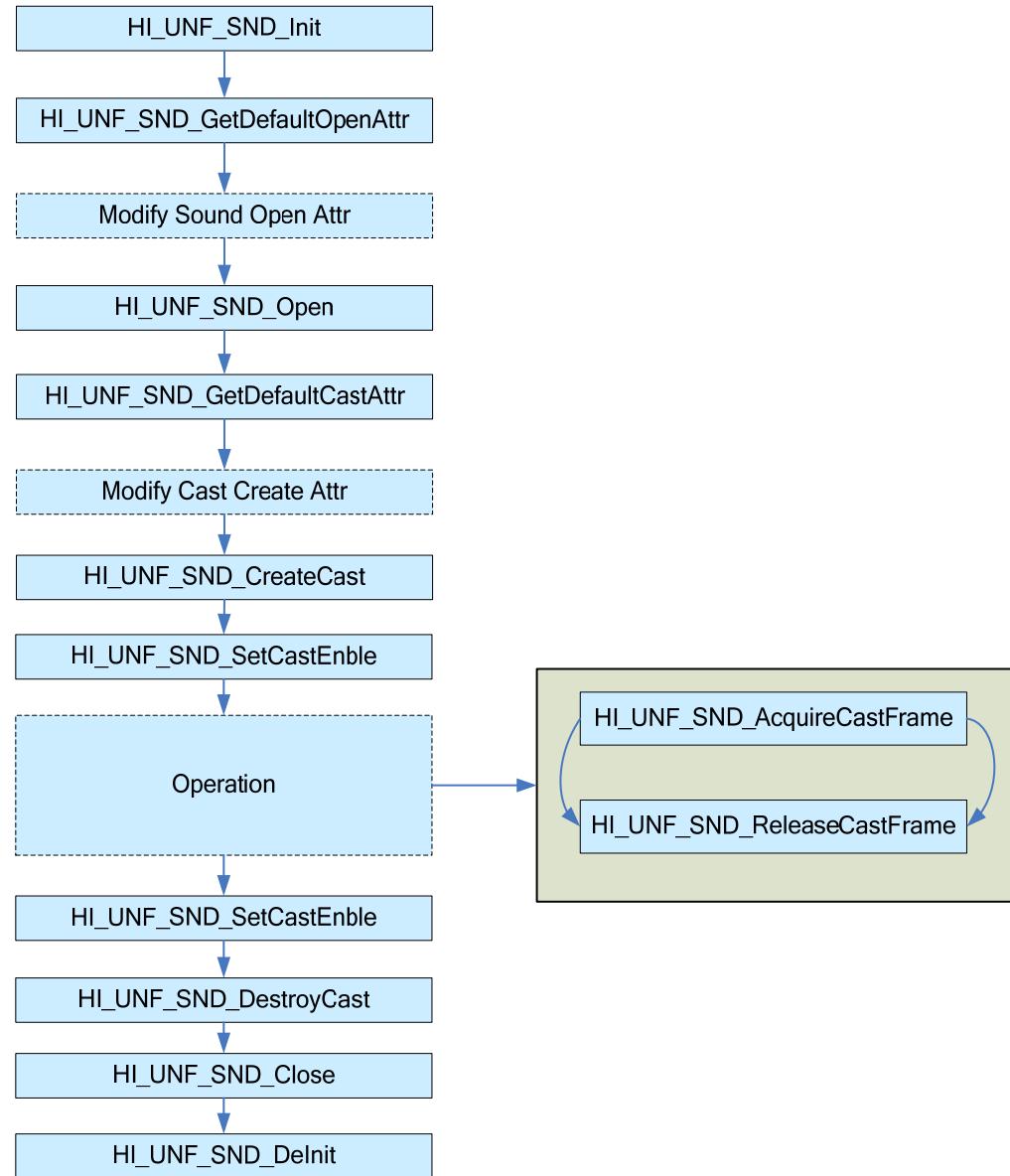
- Step 1** Disable the cast function by calling HI\_UNF\_SND\_SetCastEnable.
- Step 2** Destroy the cast by calling HI\_UNF\_SND\_DestroyCast.
- Step 3** Stop the SOUND device by calling HI\_UNF\_SND\_Close.
- Step 4** Deinitialize the AO module by calling HI\_UNF\_SND\_DeInit.

**----End**

[Figure 5-6](#) shows the typical process for capturing SOUND output data.



**Figure 5-6** Typical process for capturing SOUND output data



## Notes

- Ensure that there is audio track output before starting the cast to obtain audio frames. Otherwise, no data can be obtained.
- Enable the cast function before obtaining cast audio frames, and disable the cast function before destroying the cast handle.

## Sample

See [sample/audiocast/audio\\_cast.c](#).



# Contents

---

<b>6 DISPLAY and WINDOW .....</b>	<b>1</b>
6.1 Overview .....	1
6.2 DISP .....	1
6.2.1 Important Concepts .....	2
6.2.2 Features .....	4
6.2.3 Application Scenarios .....	5
6.3 WINDOW .....	12
6.3.2 Important Concepts .....	13
6.3.3 Feature .....	16
6.3.4 Application Scenario .....	18



# Figures

<b>Figure 6-1</b> Relationships between DISP and WINDOW .....	1
<b>Figure 6-2</b> Relationships between the DISP and other modules.....	2
<b>Figure 6-3</b> Virtual screen .....	3
<b>Figure 6-4</b> Working process of a DISP .....	6
<b>Figure 6-5</b> Process of setting 3D output mode for a DISP .....	8
<b>Figure 6-6</b> Handling process of DISP screen mirroring .....	9
<b>Figure 6-7</b> Handling process of DISP screen mirroring and encoding .....	10
<b>Figure 6-8</b> Screen capture process of a DISP .....	11
<b>Figure 6-9</b> VBI data transmission process of a DISP .....	12
<b>Figure 6-10</b> Relationships between WINDOW and other modules.....	13
<b>Figure 6-11</b> Parameter configurations for an output coordinate are the same as those for the virtual screen.....	14
<b>Figure 6-12</b> Output coordinate going beyond the virtual screen .....	15
<b>Figure 6-13</b> Configuring to make the output coordinate and virtual screen overlap in 3D output mode .....	15
<b>Figure 6-14</b> Comparison between the display effects in 2D and 3D modes .....	15
<b>Figure 6-15</b> Working process of a WINDOW .....	19
<b>Figure 6-16</b> Working process of a virtual WINDOW .....	20
<b>Figure 6-17</b> Working process of a virtual WINDOW and VENC bound together .....	21
<b>Figure 6-18</b> Capture process.....	22
<b>Figure 6-19</b> WINDOW freezing process.....	23
<b>Figure 6-20</b> WINDOW resetting process .....	24
<b>Figure 6-21</b> Quick video output process.....	25



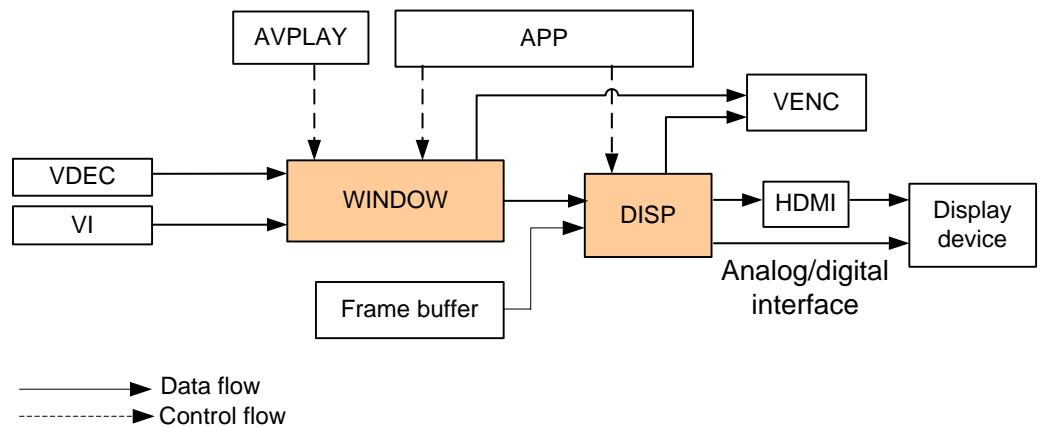
# 6 DISPLAY and WINDOW

## 6.1 Overview

A WINDOW receives visual images from upper-level modules, such as video decoding and video capture modules, adjusts the scale of the received visual images, and improves the quality of the images. After that, the WINDOW sends the processed images to lower-level modules. In addition, a WINDOW allows applications to obtain decoded visual images.

A display module (DISP) receives visual images from WINDOWs and images from frame buffers, overlays images, adjusts image colors, and then outputs the images to a display device through various video output ports. In addition, a DISP can obtain complete images containing videos and graphs on the video output ports.

**Figure 6-1** Relationships between DISP and WINDOW



## 6.2 DISP

A DISP adjusts image display effects and controls video output ports. It implements various functions, such as same-source display, port management, image color adjustment, and screen sharing.

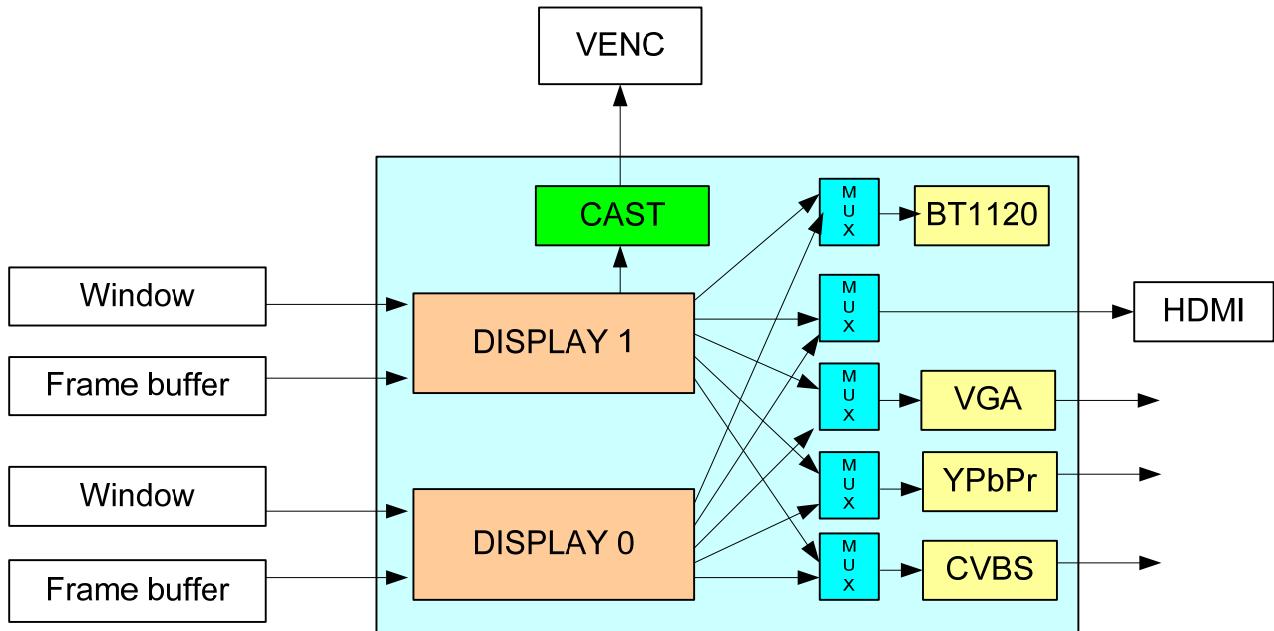
## 6.2.1 Important Concepts

### [DISPLAY]

A DISPLAY refers to a display channel. Images are output through display channels. Multiple video output ports can be bound to each display channel, for example, HDMI, YPbPr, and CVBS. The video output ports bound to the same display channel display the same images.

It is recommended that you select CVBS ports for DISPLAY 0 and select HDMI and YPbPr ports for DISPLAY 1.

**Figure 6-2** Relationships between the DISP and other modules



### [Same-source display]

Same-source display is a work mode of DISPLAY. When you bind DISPLAY m to DISPLAY n, the two display channels work in same-source display mode. In this mode, the source display channel and the target display channel display the same content. Operations performed on the source display channel are also effective on the target display channel.

### [Different-source display]

Different-source display is also a work mode of DISPLAY. Display channels are independent of each other, and upper-layer applications are separately configured and used. An application can create video windows on different display channels for playing different video contents. These display channels do not interfere with each other.

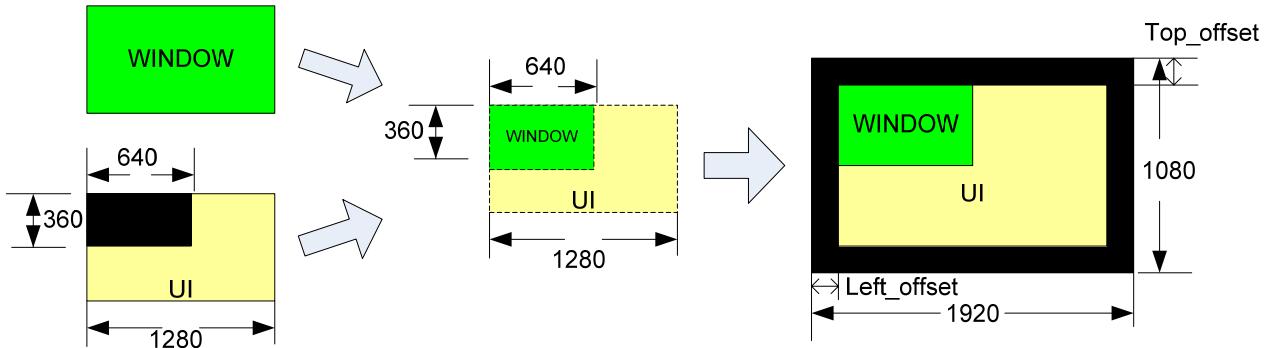
### [Virtual screen]

Virtual screen is the abstract of the real screen of a display device. It has horizontal resolution and vertical resolution. The number of offset pixels between the virtual screen and the real screen in the upward, downward, left, and right directions can be set, implementing complete display of output images on the real screen. Upper-layer applications calculate the display location coordinates of visual images based on the virtual screen. To achieve the optimal



image quality, keep the resolution of the virtual screen consistent with that of the graphical interface.

**Figure 6-3** Virtual screen



#### [Physical screen]

Similar to the virtual screen, a physical screen is also a coordinate system for setting the coordinates, width, and height of video windows or graphics OSDs. The difference is that the physical screen is based on the current physical resolution and is valid for only video windows. A video window can use the coordinate systems of both the virtual screen and the physical screen as references, but a graphics OSD can use only the coordinate system of the virtual screen.

Therefore, the video window and graphics OSD can be separately configured based on different coordinate systems, meeting requirements of some scenarios such as the border cropping scenario.

#### [Aspect ratio of a display device]

Aspect ratio of a display device refers to the proportion of the horizontal length to the vertical length of the physical screen of a device. Generally, the aspect ratio of a display device is 4:3 or 16:9. Whether visual images can be displayed in proper proportion relies on the aspect ratio of the display device. For example, when a traditional standard-definition television program whose images have an aspect ratio of 4:3 is played in full screen on a wide-screen television with an aspect ratio of 16:9, the images are stretched and deformed in horizontal direction.

#### [CAST]

CAST refers to an image collection channel created based on DISPLAY. This channel collects completely displayed images consisting of visual images and overlapping graphs that are collected at regular intervals continually. An application can specify the pixel, format, and target resolution of the images to be provided by CAST.

#### [Snapshot]

Snapshot refers to an image collection service provided based on DISPLAY. The difference between Snapshot and CAST is that Snapshot collects only single-frame images that are completely displayed currently.

#### [VBI]

Vertical Blanking Interval (VBI) data refers to the data transmitted through the VBI protocol during the interval of field blanking in the normal playing process. VBI implements data transmission for display devices. VBI data includes Closed Caption, Teletext, and WSS.



## 6.2.2 Features

The DISP manages video output ports, outputs signal standard configurations, and adjusts image colors. It provides the following application programming interfaces (APIs):

- HI\_UNF\_DISP\_Init: Initializes
- HI\_UNF\_DISP\_DeInit: Deinitializes
- HI\_UNF\_DISP\_Open: Opens the display channel.
- HI\_UNF\_DISP\_Close: Closes the display channel.
- HI\_UNF\_DISP\_SetVirtualScreen: Sets the resolution of the virtual screen.
- HI\_UNF\_DISP\_GetVirtualScreen: Queries the resolution of the virtual screen.
- HI\_UNF\_DISP\_SetScreenOffset: Sets the number of offset pixels between the virtual screen and real screen.
- HI\_UNF\_DISP\_GetScreenOffset: Queries the number of offset pixels between the virtual screen and real screen.
- HI\_UNF\_DISP\_Attach: Sets the binding relationships of display channels (in same-source display scenario).
- HI\_UNF\_DISP\_Detach: Sets the unbinding relationships of display channels (in same-source display scenario).
- HI\_UNF\_DISP\_AttachIntf: Sets the binding relationships between the display channel and output port.
- HI\_UNF\_DISP\_DetachIntf: Sets the unbinding relationships between the display channel and output port.
- HI\_UNF\_DISP\_SetFormat: Sets the format of the output signal.
- HI\_UNF\_DISP\_GetFormat: Queries the format of the output signal.
- HI\_UNF\_DISP\_SetCustomTiming: Sets the user-defined signal output sequence.
- HI\_UNF\_DISP\_GetCustomTiming: Queries the user-defined signal output sequence.
- HI\_UNF\_DISP\_Set3DMode: Sets the signal output mode of 3D images.
- HI\_UNF\_DISP\_Get3DMode: Queries the signal output mode of 3D images.
- HI\_UNF\_DISP\_SetRightEyeFirst: Sets right eye first in 3D mode.
- HI\_UNF\_DISP\_SetLayerZorder: Sets the sequence of the image layers from top to bottom.
- HI\_UNF\_DISP\_GetLayerZorder: Queries the sequence of the image layers from top to bottom.
- HI\_UNF\_DISP\_SetBgColor: Sets the background color.
- HI\_UNF\_DISP\_GetBgColor: Queries the background color.
- HI\_UNF\_DISP\_SetBrightness: Sets the brightness value.
- HI\_UNF\_DISP\_GetBrightness: Queries the brightness value.
- HI\_UNF\_DISP\_SetContrast: Sets the contrast value.
- HI\_UNF\_DISP\_GetContrast: Queries the contrast value.
- HI\_UNF\_DISP\_SetSaturation: Sets the saturation value.
- HI\_UNF\_DISP\_GetSaturation: Queries the saturation value.
- HI\_UNF\_DISP\_SetHuePlus: Sets the hue value.
- HI\_UNF\_DISP\_GetHuePlus: Queries the hue value.
- HI\_UNF\_DISP\_SetColor: Sets color configuration parameters.



- HI\_UNF\_DISP\_GetColor: Queries color configuration parameters.
- HI\_UNF\_DISP\_SetAspectRatio: Sets the aspect ratio of the connected display device.
- HI\_UNF\_DISP\_GetAspectRatio: Queries the aspect ratio of the connected display device.
- HI\_UNF\_DISP\_GetDefaultCastAttr: Queries default configurations of the CAST channel.
- HI\_UNF\_DISP\_CreateCast: Creates a CAST channel.
- HI\_UNF\_DISP\_DestroyCast: Destroys a CAST channel.
- HI\_UNF\_DISP\_SetCastEnable: Enables/disables the CAST channel.
- HI\_UNF\_DISP\_GetCastEnable: Queries the status (enabled or disabled) of the CAST channel.
- HI\_UNF\_DISP\_AcquireCastFrame: Requests a CAST image.
- HI\_UNF\_DISP\_ReleaseCastFrame: Releases a CAST image.
- HI\_UNF\_DISP\_AcquireSnapshot: Requests a snapshot image.
- HI\_UNF\_DISP\_ReleaseSnapshot: Releases a snapshot image.
- HI\_UNF\_DISP\_CreateVBI: Creates a VBI channel.
- HI\_UNF\_DISP\_DestroyVBI: Destroys a VBI channel.
- HI\_UNF\_DISP\_SendVBIData: Sends VBI data.
- HI\_UNF\_DISP\_SetWss: Sets the WSS.
- HI\_UNF\_DISP\_GetMacrovisionSupport: Queries whether Macrovision is supported.
- HI\_UNF\_DISP\_SetMacrovision: Sets the Macrovision mode.
- HI\_UNF\_DISP\_SetCgms: Sets the CGMS-A mode.
- HI\_UNF\_DISP\_SetIsogenyAttr: Sets the standards of the two channels in the same-source scenario.

### 6.2.3 Application Scenarios

The typical application scenarios of a DISP are as follows:

- Same-source/different-source scenario (basic scenario)
- 3D display scenario
- Screen image (CAST) scenario
- Screen capture scenario
- VBI data output scenario

#### 6.2.3.1 Same-Source/Different-Source Scenario

##### Scenario

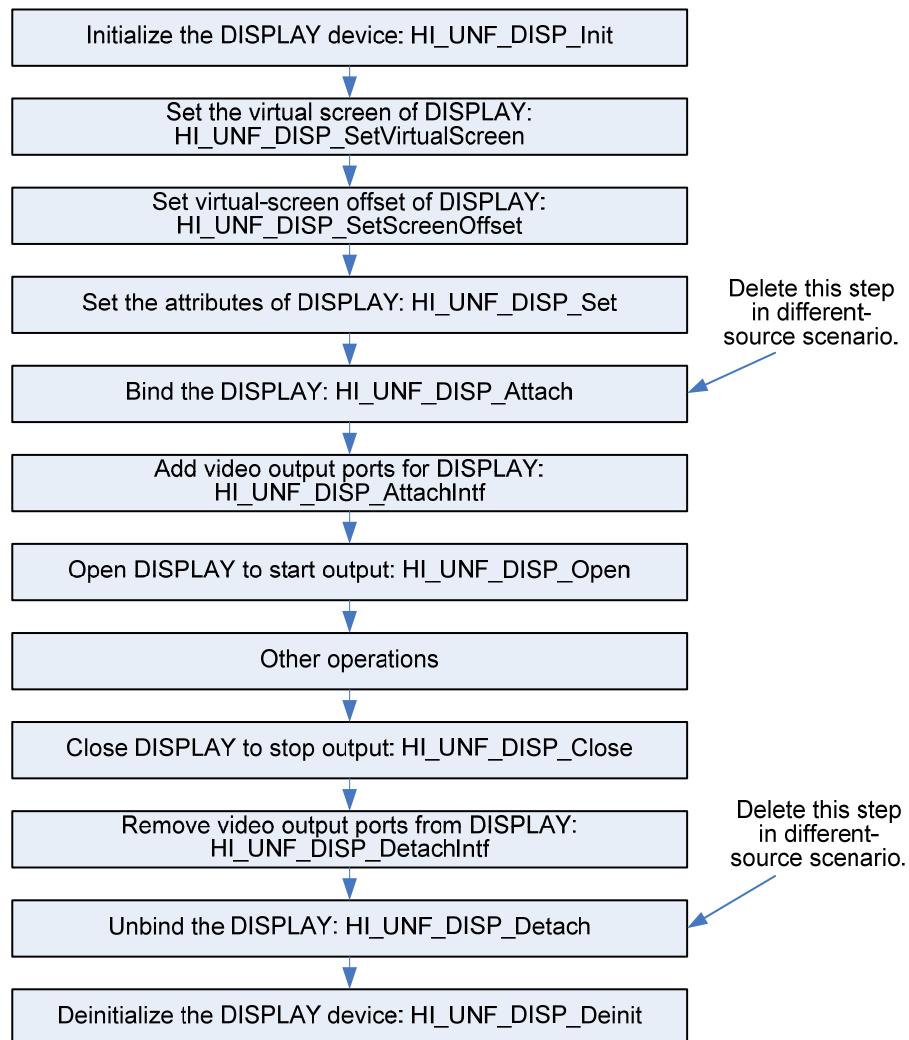
- Same-source scenario of DISPLAY  
Bind DISPLAY 0 to DISPLAY 1 to make the two display channels output the same image contents.
- Different-source scenario of DISPLAY  
Do not bind DISPLAY 0 and DISPLAY 1. The two display channels work independently and output different image contents.



## Working Process

Figure 6-4 shows the working process of a DISP.

**Figure 6-4** Working process of a DISP



## Notes

- Before invoking the modules that need to be displayed, such as VO and HiGo, you must initialize a DISP.
- When relevant modules are no longer used, close them and then initialize the DISP.
- If baseparam is burnt, the same-source and different-source modes cannot be dynamically switched.  
If baseparam is not burnt, the same-source and different-source modes can be dynamically switched by calling HI\_UNF\_DISP\_Attach and HI\_UNF\_DISP\_Detach.

Same-source mode:

- Open the source DISPLAY and then the destination DISPLAY.



- To achieve the optimal image synchronization effects, set display standards with the same refresh rate for the bound display channels. For example, if you set 720P\_50 for DISPLAY 1, you need to set a standard of 50 Hz for DISPLAY 0, such as PAL. If the refresh rates are inconsistent, the video fluency of the destination DISPLAY may be slightly affected. For example, if you set 1080P\_25 for DISPLAY 1, set PAL for DISPLAY 0.
- To achieve the optimal image synchronization effects, you are advised to open the bound display channels simultaneously. For example, configure the attributes of DISPLAY 0 and DISPLAY 1 and then run two neighboring commands to open DISPLAY 0 and DISPLAY 1.
- When the following interfaces are used to perform operations on the source display channel, these operations are automatically performed on the destination display channel.
  - HI\_UNF\_DISP\_SetLayerZorder
  - HI\_UNF\_DISP\_SetBgColor
  - HI\_UNF\_DISP\_SetBrightness
  - HI\_UNF\_DISP\_SetContrast
  - HI\_UNF\_DISP\_SetSaturation
  - HI\_UNF\_DISP\_SetHuePlus
- The new API HI\_UNF\_DISP\_SetIsogenyAttr is used to set the standards of HD and SD channels in the same-source scenario. The original API HI\_UNF\_DISP\_SetFormat is still valid, but the two APIs cannot be used at the same time. You are advised to use HI\_UNF\_DISP\_SetIsogenyAttr in later versions.

Different-source mode:

Relevant attributes, such as the chromaticity, brightness, saturation, and contrast, must be set separately.

Dual-channel HD/SD output select:

- In the same-source and different-source mode, one HD video output and one SD video output are used by default.
- To enable two HD video outputs, run **make menuconfig** in the SDK root directory, choose **Msp > VO Config**, and select **QPLL Be Used By Display Solely**.
- Dual-channel HD outputs cannot coexist with the QAM module.

## Sample

For the display in same-source mode, refer to **sample/tsplay/tsplay.c**.

For the display in different-source mode, refer to  
**sample/separateoutput/videoseparateoutput.c**.

### 6.2.3.2 3D Display

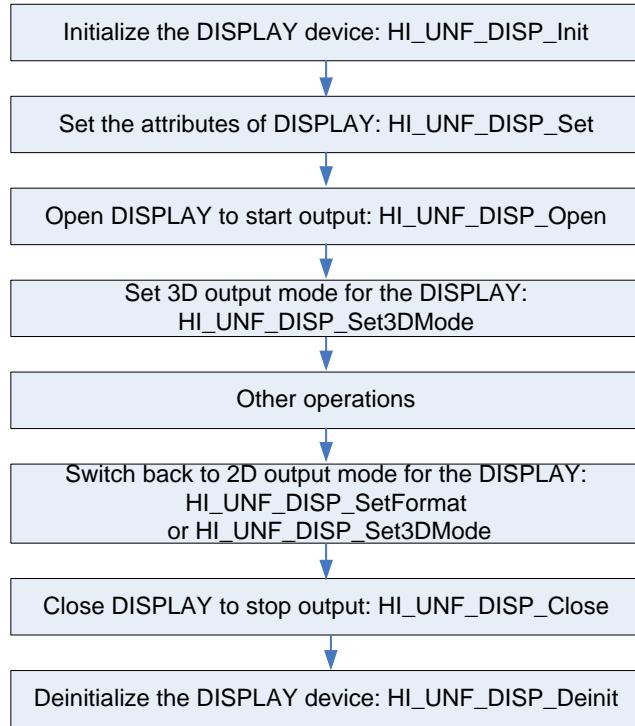
#### Scenario

DISPLAY supports output of 3D images through HDMI TX ports, including side-by-side half, top-and-bottom, and frame packing. The format of 3D images must comply with the HDMI TX 1.4a protocol. DISPLAY supports switching between output of 2D and 3D images.



## Working Process

**Figure 6-5** Process of setting 3D output mode for a DISP



## Notes

Before enabling the 3D function, verify that the display device supports display of 3D images based on the capability set of the display device obtained by the HDMI TX.

## Sample

None

### 6.2.3.3 Screen Mirroring Scenario

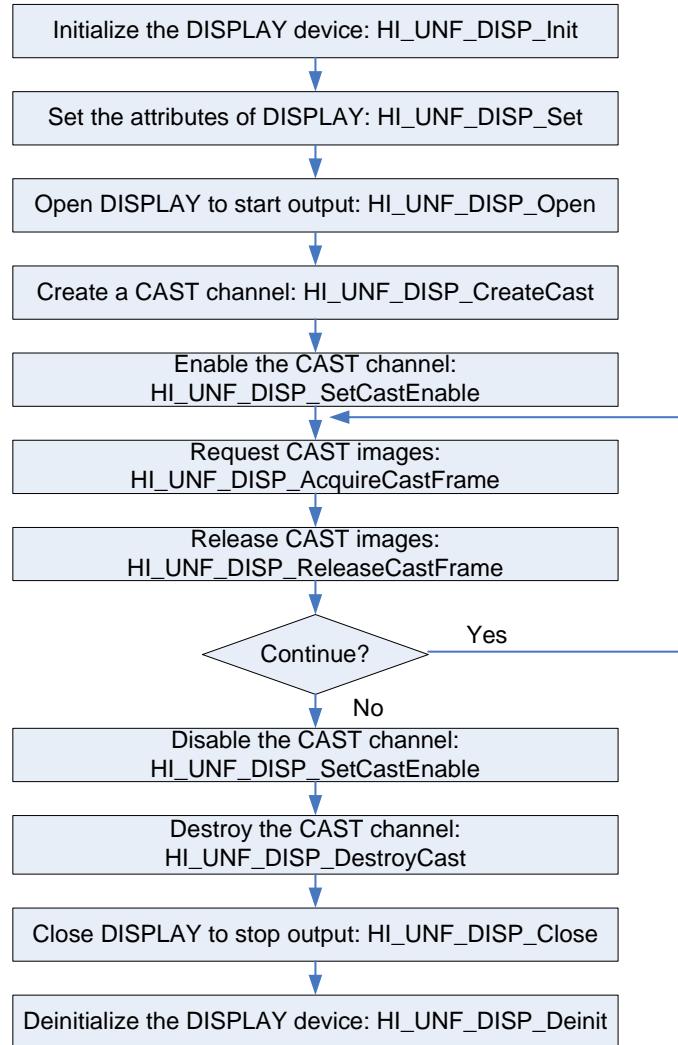
#### Scenario

DISPLAY 1 allows you to create a CAST channel. The CAST channel can capture images output by the display channel continuously. The captured images can serve as the screen images of the display device. Applications can obtain these images. These images can also be sent to other display devices to implement image sharing. DISP allows you to bind a CAST channel to a video encoder (VENC), and encode the captured images directly.



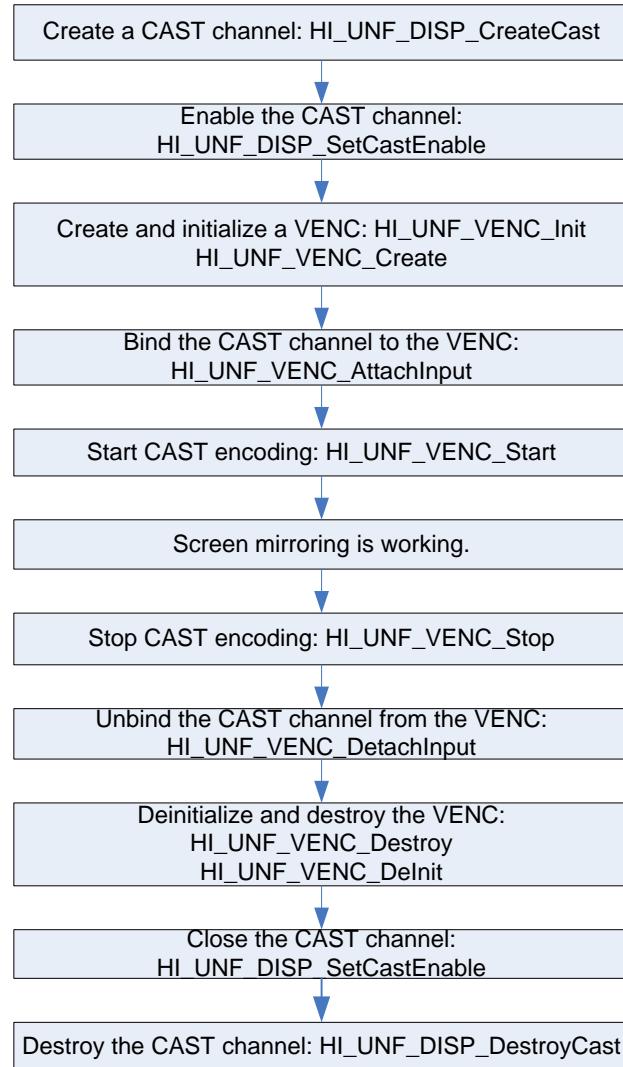
## Working Process

**Figure 6-6** Handling process of DISP screen mirroring





**Figure 6-7** Handling process of DISP screen mirroring and encoding



## Notes

- Only DISPLAY 1 allows you to create a CAST channel.
- If the video frames collected by a CAST channel are not processed by an application or VENC timely, the image buffer will be filled up. In this case, image collection stops. When a new buffer is available, image collection starts again.

## Sample

Binding mode of the CAST channel and VENC:

sample/avcast/ ts\_cast.c



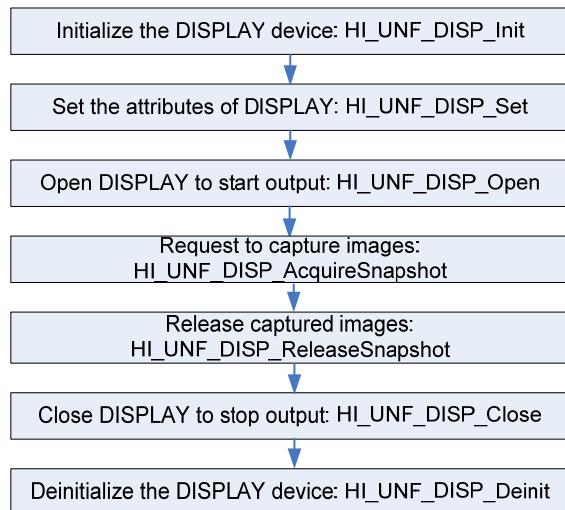
### 6.2.3.4 Screen Capture Scenario

#### Scenario

The screen capture scenario is similar to the screen mirroring scenario. The only difference is that the screen capture function captures the only image being output by a DISPLAY in screen capture scenario.

#### Working Process

**Figure 6-8** Screen capture process of a DISP



#### Notes

The resolution of a captured image is the same as that of the image being output by the DISPLAY currently.

#### Sample

None

### 6.2.3.5 VBI Data Transmission Scenario

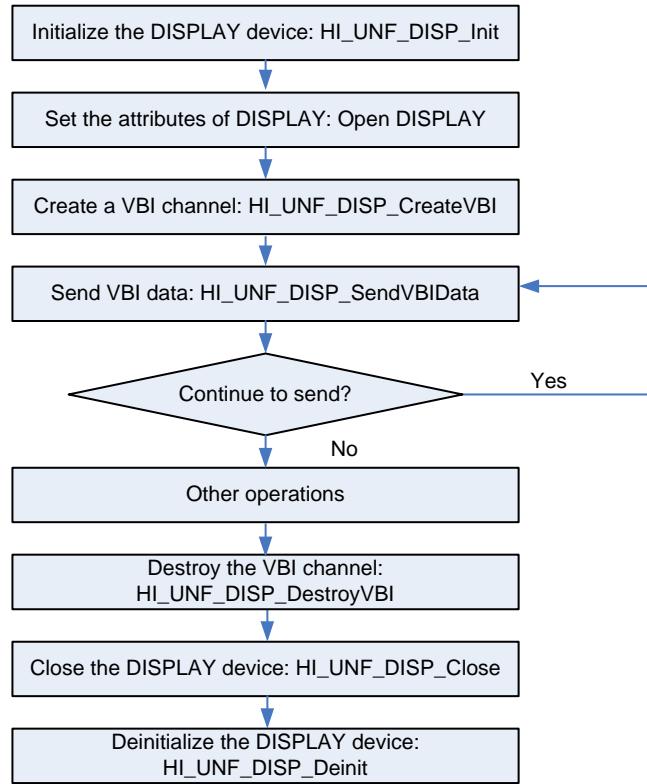
#### Scenario

Some television systems provide teletext services, for example, the hidden caption service provided to people having hearing obstacle, which needs the support of the VBI function.



## Working Process

**Figure 6.9** VBI data transmission process of a DISP



## Notes

- CVBS ports are required for transmitting VBI data; therefore, CVBS ports need to be added to a DISPLAY before transmitting VBI data.
- Only one channel can be created for each type of VBI data at the same time.
- After VBI data is transmitted, destroy the corresponding channel.

## Sample

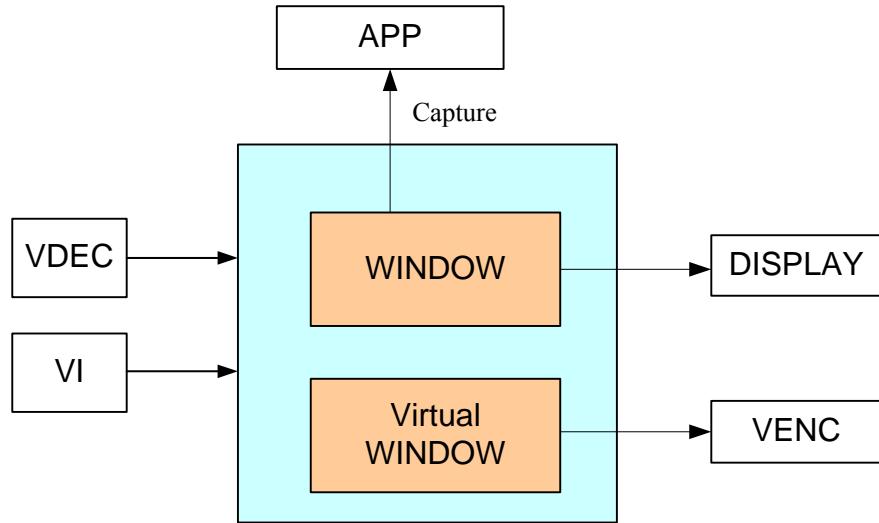
None

## 6.3 WINDOW

A WINDOW implements various functions, such as video output window control, picture in picture (PIP), and multi-picture display. In addition, a virtual WINDOW transmits visual images to a VENC for processing.



**Figure 6-10** Relationships between WINDOW and other modules



### 6.3.2 Important Concepts

#### [WINDOW]

A WINDOW refers to a video display window and is created based on a DISPLAY. Each WINDOW can only play visual images of one channel. A maximum of 17 WINDOWS can be created on one DISPLAY. An application controls a WINDOW to control the video display screen.

When multiple WINDOWS are created, only one WINDOW is allowed to overlap with other WINDOWS, and the Z order of this WINDOW cannot be between the Z orders of the overlaid WINDOWS. That is, the Z order of the WINDOW is greater than or less than the Z orders of all the overlaid WINDOWS.

#### [Z order]

The Z order indicates the display sequence of WINDOWS on the screen. For two adjacent WINDOWS, the one with the larger Z order value is displayed in front of the other one. The newly created WINDOW has the largest Z order and therefore is displayed on the top.

The Z order is dynamically adjusted within a certain range as the WINDOWS are created and destroyed. For example, if N WINDOWS are created, the Z order range is [0, N-1].

#### [Virtual WINDOW]

A virtual WINDOW is a special WINDOW. Images processed by a virtual WINDOW are not displayed on a display device through a display channel. Applications can fetch the images processed by a virtual WINDOW for video encoding. Unlike physical WINDOWS, virtual WINDOWS are not created based on a DISPLAY. A maximum of 16 virtual WINDOWS can be created.

#### [Freeze]

Freeze is a function of a WINDOW, which involves two modes, that is, blank screen and frame freezing. After Freeze is enabled, the corresponding WINDOW is in frozen state, and the screen displays blank screen or stops the current picture. However, the WINDOW still processes the visual images provided by upper-level modules at normal display speed,



bringing about no reverse pressure to upper-level modules. When you disable Freeze, the WINDOW restores to the normal playing status, the all-black or frozen picture disappears, and the visual images being processed by the WINDOW continue to be displayed.

A virtual WINDOW does not support the Freeze function.

[Reset]

Reset is a function of a WINDOW, which involves two modes, that is, blank screen and frame freezing. After you perform the reset operation, the screen displays blank screen or stops the current picture, and the WINDOW clears the visual images waiting to be displayed in the buffer. After that, the WINDOW restores to the normal playing status and continues to receive visual images provided by upper-level modules for processing and display.

A virtual WINDOW does not support the Reset function.

[Capture]

Capture is a function of a WINDOW. Applications request to capture and release the visual image being displayed currently through Capture interfaces.

A virtual WINDOW does not support the Capture function.

[QuickOutput]

QuickOutput is a function of a WINDOW. When you enable the QuickOutput function, the WINDOW processes the last received frame of visual image immediately and releases other images in the buffer, and repeats these operations subsequently. After you disable the QuickOutput function, the WINDOW restores to the normal playing status.

A virtual WINDOW does not support the QuickOutput function.

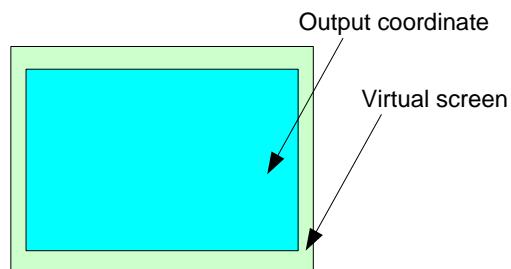
[Aspect ratio switch]

Aspect ratio switch is a function of a WINDOW. This function allows you to adjust the horizontal and vertical lengths of visual images to fit the WINDOW specified by an application while ensuring proper proportion of the horizontal length to vertical length of the source visual images. Note that the scaling ratio in the horizontal direction is not always the same as that in the vertical direction of visual images.

[Output coordinate]

An output coordinate of a WINDOW consists of the horizontal axis (X), vertical axis (Y), width (W) of window, and height (H) of window. The output coordinate is defined based on the virtual screen of a DISP. When the parameter configurations for an output coordinate are the same as those for the virtual screen, the corresponding WINDOW displays images in full screen. See [Figure 6-11](#).

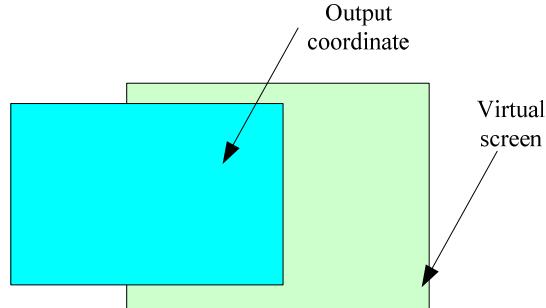
**Figure 6-11** Parameter configurations for an output coordinate are the same as those for the virtual screen





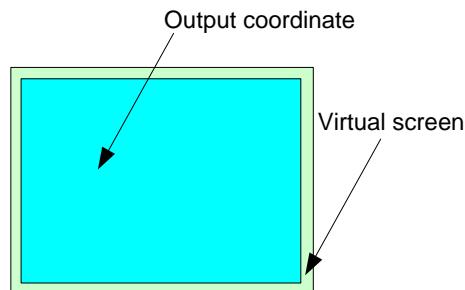
A WINDOW allows the output coordinate to go beyond the virtual screen. See [Figure 6-12](#).

**Figure 6-12** Output coordinate going beyond the virtual screen



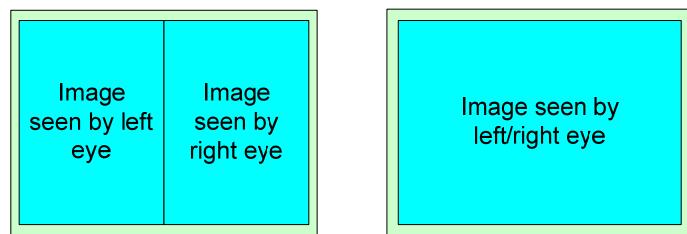
When a WINDOW works in 3D output mode, set the parameters related to the output coordinate based on the final display effects of the images seen by your left eye. For example, when the SBS video source is displayed in full screen, configure relevant parameters to make the output coordinate and virtual screen overlap, as shown in [Figure 6-13](#).

**Figure 6-13** Configuring to make the output coordinate and virtual screen overlap in 3D output mode



The left part of [Figure 6-14](#) shows the display effects of a television in 2D mode, and the right part of [Figure 6-14](#) shows the display effects in 3D mode.

**Figure 6-14** Comparison between the display effects in 2D and 3D modes





### 6.3.3 Feature

A WINDOW manages video windows and provides the following APIs:

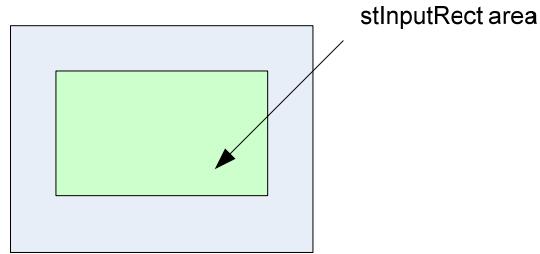
- HI\_UNF\_VO\_Init: Initializes a device.
- HI\_UNF\_VO\_DeInit: Deinitializes the device.
- HI\_UNF\_VO\_CreateWindow: Creates a WINDOW.
- HI\_UNF\_VO\_DestroyWindow: Destroys a WINDOW.
- HI\_UNF\_VO\_SetWindowAttr: Sets the attributes of a WINDOW.
- HI\_UNF\_VO\_GetWindowAttr: Queries the attributes of a WINDOW.
- HI\_UNF\_VO\_SetWindowEnable: Enables/disables a WINDOW.
- HI\_UNF\_VO\_GetWindowEnable: Queries the status (enabled or disabled) of a WINDOW.
- HI\_UNF\_VO\_AttachWindow: Sets the video source binding relationships.
- HI\_UNF\_VO\_DetachWindow: Removes the video source binding relationships.
- HI\_UNF\_VO\_AcquireFrame: Requests images from a virtual WINDOW.
- HI\_UNF\_VO\_ReleaseFrame: Releases images to a virtual WINDOW.
- HI\_UNF\_VO\_SetWindowZorder: Sets the display sequence (from top to bottom) for a WINDOW.
- HI\_UNF\_VO\_FreezeWindow: Freezes a WINDOW, and displays the last frame or black frame.
- HI\_UNF\_VO\_GetWindowFreezeStatus: Obtains the freeze state of a WINDOW.
- HI\_UNF\_VO\_ResetWindow: Resets a WINDOW.
- HI\_UNF\_VO\_CapturePicture: Captures the current frame image.
- HI\_UNF\_VO\_CapturePictureRelease: Releases the current frame image.
- HI\_UNF\_VO\_SetQuickOutputEnable: Sets the quick output mode.
- HI\_UNF\_VO\_GetQuickOutputStatus: Obtains the quick output mode.
- HI\_UNF\_VO\_AttachExternBuffer: Configures frame buffers provided by applications for a virtual WINDOW.
- HI\_UNF\_VO\_SetStereoDtpth: Sets the depth of field (valid in only 3D display mode).
- HI\_UNF\_VO\_GetStereoDtpth: Obtains the depth of field.
- HI\_UNF\_VO\_CreateWindowExt: Creates a WINDOW based on the physical screen.

Among these APIs, HI\_UNF\_VO\_SetWindowAttr can be used to implement the following attribute configurations:

- Display area (stInputRect) of visual images:

A rectangle area is specified for visual images. Images in this area are displayed.

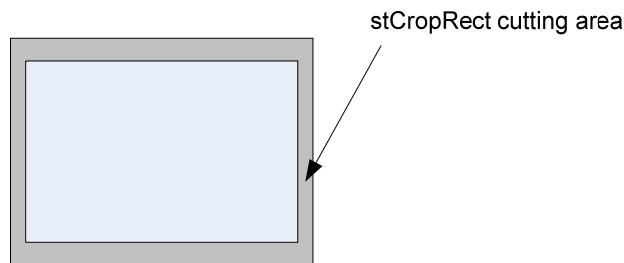
- If this rectangle area is larger than the video source or all parameters related to this area are set to 0, visual images need to be displayed completely.
- The width and height of this area must not be smaller than 64. Otherwise, they are both adjusted to 64 automatically.
- When **bUseCropRect** is set to **HI\_FALSE**, the **stInputRec** parameter is valid.



- Cutting area (stCropRect) of visual images:

A rectangle area is specified for visual images. Images out of this area are cut off.

- If all parameters related to this area are set to 0, visual images need to be displayed completely.
- The cutting area must not be larger than visual images.
- When **bUseCropRect** is set to **HI\_TRUE**, the **stCropRect** parameter is valid.

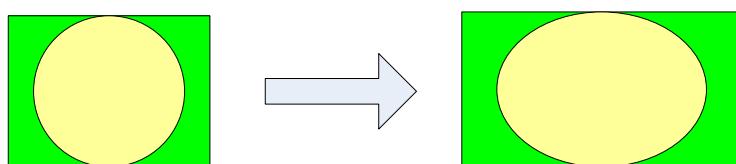


- Display area (stOutputRect) of a WINDOW on the virtual screen

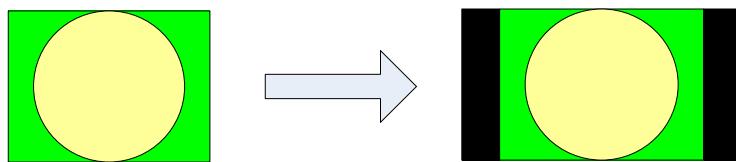
If all parameters related to this area are set to 0, the WINDOW covers the whole screen. You can also set relevant parameters to make a part or the whole area go beyond the virtual screen. The part going beyond the virtual screen will not be displayed.

- Aspect ratio mode configuration (stWinAspectAttr. enAspectCvrs):

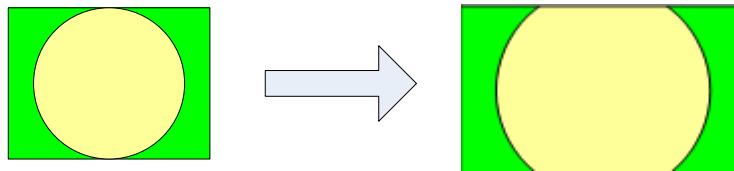
- IGNORE: stretches the video to fill the window.



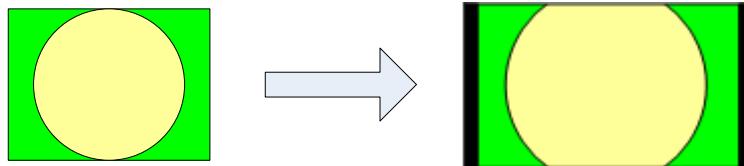
- LETTERBOX: keeps the aspect ratio by adding black bands.



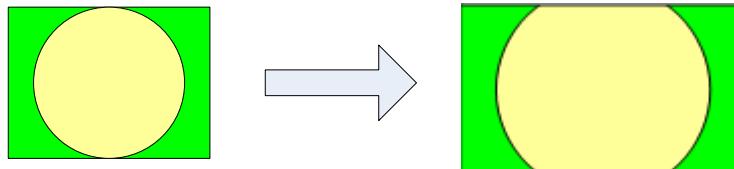
- PAN\_SCAN: keeps the aspect ratio by cutting the video source.



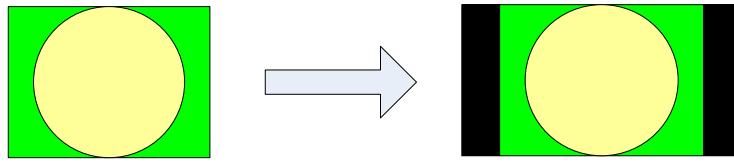
- COMBINED: keeps the aspect ratio by adding black bands and cutting the video source.



- HORIZONTAL\_FULL: keeps the aspect ratio by stretching the video to fill the window in horizontal direction.



- VERTICAL\_FULL: keeps the aspect ratio by stretching the video to fill the window in vertical direction



- stWinAspectAttr.bUserDefAspectRatio: displays a video at the user-defined aspect ratio that ranges from 1/16 to 16.

### 6.3.4 Application Scenario

The typical application scenarios of a WINDOW are as follows:

- Video display
- Virtual WINDOW
- Capturing video frames
- Freezing a WINDOW
- Resetting a WINDOW
- Quick video output



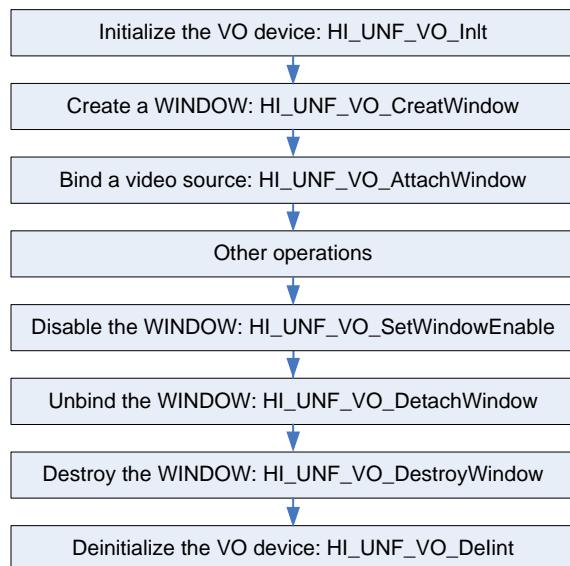
### 6.3.4.1 Video Display

#### Scenario

Video display is a basic application scenario of a WINDOW. An application can create one or multiple WINDOWS based on a certain DISPLAY. You can bind a video source for each WINDOW to implement video output on a DISPLAY. An application can control the display location and display effects of visual images and the hierarchy relationships of multiple video pictures based on WINDOWS.

#### Working Process

**Figure 6-15** Working process of a WINDOW



#### Notes

- Before initializing a WINDOW, initialize the corresponding DISPLAY.
- In same-source scenario, the destination DISPLAY automatically displays contents the same as those displayed by the source DISPLAY, and applications do not need to create a WINDOW on the destination DISPLAY.
- When invoking HI\_UNF\_VO\_CreateWindow, to implement full-screen display, you can set all output range parameters to 0.
- Video sources can be bound or unbound only when the WINDOW is in Disable state.
- The overlapping relationship of WINDOWS must meet the requirements described in section [6.3.2 "Important Concepts"](#), and the Z order must also meet the requirements.
- WINDOWS that are disabled are not restricted by the WINDOW overlapping conditions.

#### Sample

- For the one-channel video display scenario, refer to [sample/dvbplay/ dvbplay.c](#).
- For the PIP video display scenario, refer to [sample/pip/ sample\\_pipdvb.c](#).



- For the usage in MOSAIC mode, refer to [sample/mosaic/dvb\\_mosaic.c](#).

### 6.3.4.2 Virtual WINDOW

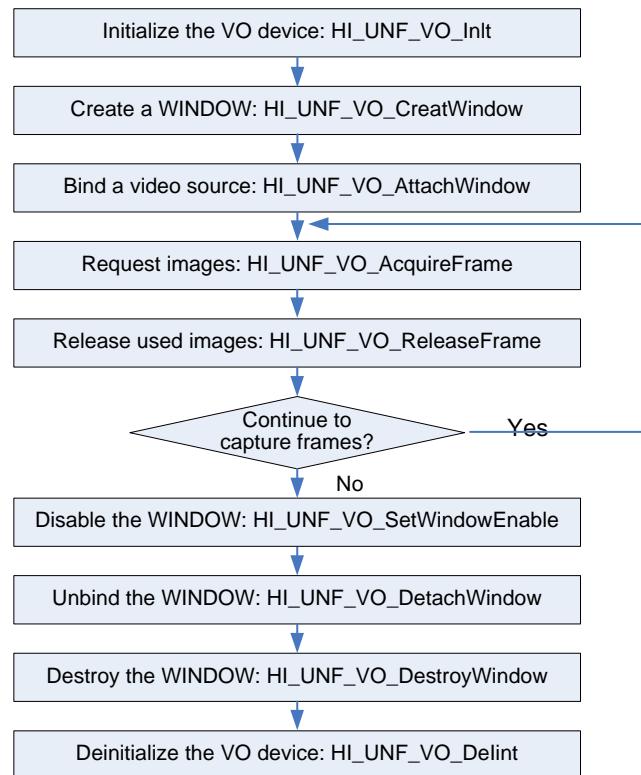
#### Scenario

A virtual WINDOW is a special WINDOW. It only simulates operations of a real WINDOW and processes input images. The images processed by a virtual WINDOW are not displayed, but provided for the use of lower-level modules.

A virtual WINDOW can be bound to a VENC to work together with the VENC. The images output by a virtual WINDOW are directly transmitted to the bound VENC to implement video transcoding.

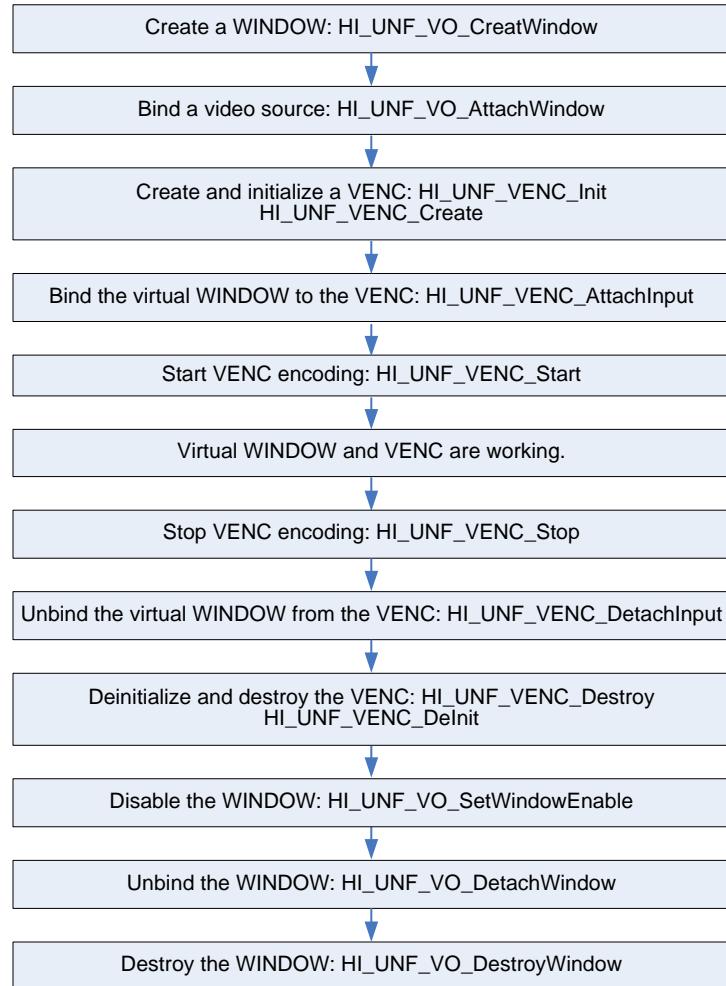
#### Working Process

**Figure 6-16** Working process of a virtual WINDOW





**Figure 6-17** Working process of a virtual WINDOW and VENC bound together



## Notes

- Virtual WINDOWS do not need to be created based on a DISPLAY.
- Images output by a virtual WINDOW cannot be displayed.
- A created virtual WINDOW cannot be converted into a physical WINDOW. In addition, the attributes of a virtual WINDOW cannot be changed.
- The format of the data output by a virtual WINDOW is determined by the chip.

## Sample

For the usage of a virtual WINDOW, refer to [sample/transcode/sample\\_transcode.c](#).



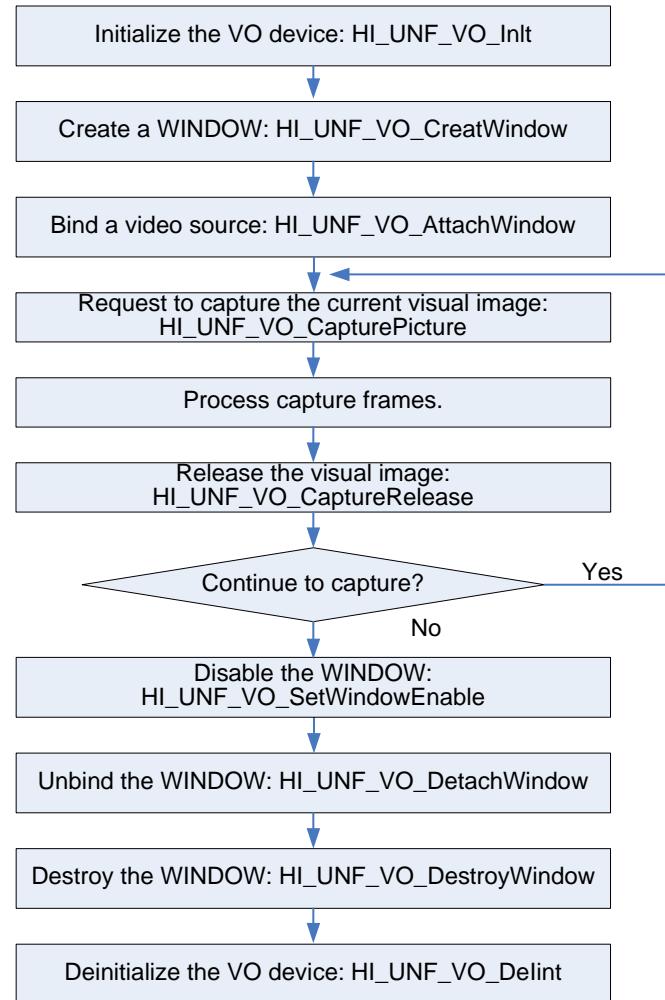
### 6.3.4.3 Capturing Video Frames

#### Scenario

During video playing, applications can capture frames from a current video playing window and then post-process the captured frames. For example, an application captures a visual image and then invokes HiGo to convert the format of the image to .jpeg for saving it.

#### Working Process

**Figure 6-18** Capture process



#### Notes

- Visual images can be captured from only non-virtual WINDOWS.
- Frame capture information HI\_UNF\_VO\_CapturePicture and HI\_UNF\_VO\_CapturePictureRelease must appear simultaneously for timely release of frame storage space.
- Difference from HI\_UNF\_DISP\_Snapshot of DISPLAY:



The capture function captures contents from the current playing window, whereas the snapshot function captures the contents currently displayed on a screen.

## Sample

For the usage of the capture function, refer to [sample/capture/capture.c](#).

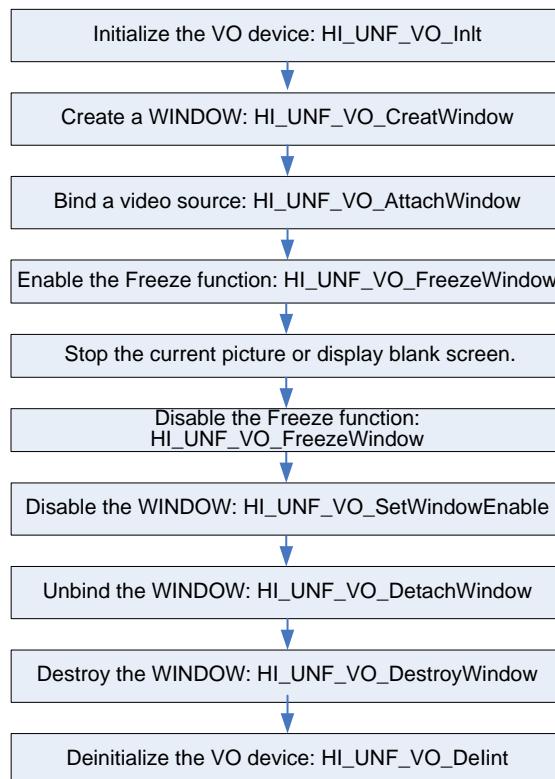
### 6.3.4.4 Freezing a WINDOW

#### Scenario

During video playback, applications can configure relevant parameters through Freeze interfaces to make a WINDOW display blank screen or stop the current picture. The WINDOW still processes the visual images provided by upper-level modules at normal display speed.

#### Working Process

**Figure 6-19** WINDOW freezing process



#### Notes

None

## Sample

None



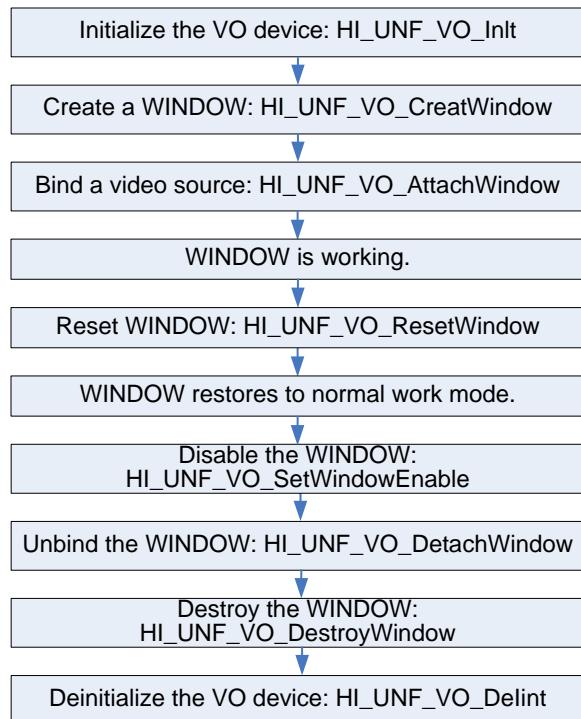
### 6.3.4.5 Resetting a WINDOW

#### Scenario

During video playing, applications can configure relevant parameters by invoking Reset interfaces to make a WINDOW display blank screen or stop the current picture and return other buffered visual images to upper-level modules.

#### Working Process

**Figure 6-20** WINDOW resetting process



#### Notes

Reset is an operation performed on a WINDOW. It is not used to set a WINDOW to a certain state. That is, after the reset operation is performed on a WINDOW, the WINDOW still works properly.

#### Sample

None

### 6.3.4.6 Quick Video Output

#### Scenario

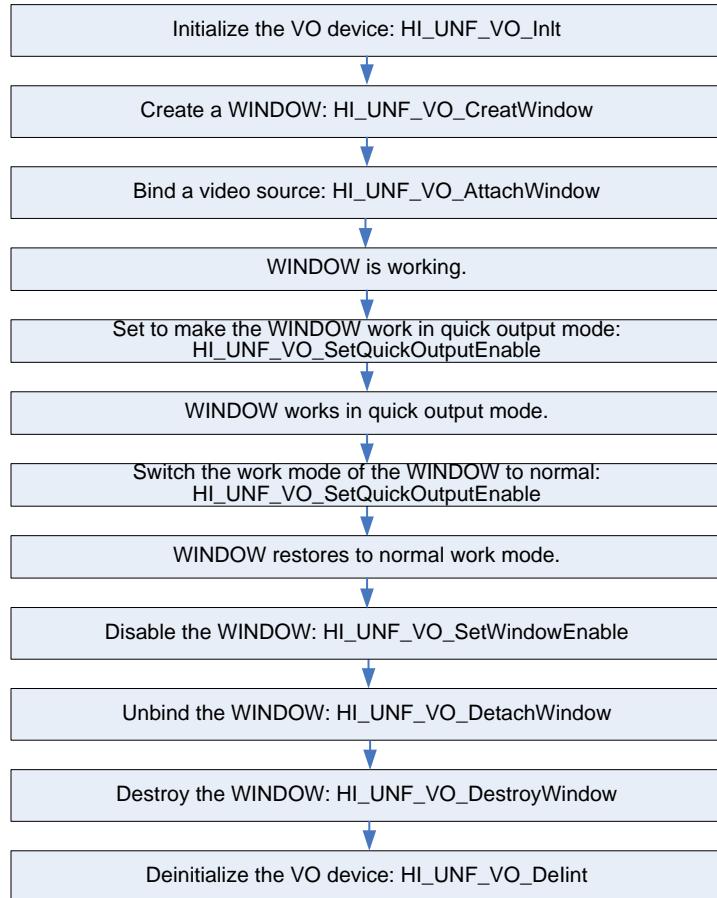
In application scenarios like cloud-based games or video calling, WINDOWS need to display the last frame of visual images provided by upper-level modules as quick as possible. You can



perform relevant configurations to make the WINDOWS work in quick video output mode to achieve this purpose.

## Working Process

**Figure 6-21** Quick video output process



## Notes

The quick video output mode aims to achieve the effect of low-delay display of visual images. Buffered video frames may be directly released to upper-level modules. In this case, the fluency of visual images is reduced.

## Sample

None



# Contents

---

<b>7 HDMI .....</b>	<b>1</b>
7.1 Overview .....	1
7.2 Important Concepts .....	1
7.3 Features .....	2
7.4 Development Guide .....	5
7.4.1 Detecting HDMI Hot Plug .....	5
7.4.2 Sending HDMI CEC Messages.....	7



# Figures

<b>Figure 7-1</b> Structure of the HDMI module .....	1
<b>Figure 7-2</b> HDMI functions.....	2
<b>Figure 7-3</b> Process for using the HDMI module .....	6
<b>Figure 7-4</b> Working process for the HDMI CEC.....	7



# Tables

<a href="#">Table 7-1 Major attributes of HI_UNF_HDMI_CALLBACK_FUNC_S .....</a>	2
<a href="#">Table 7-2 Major attributes of HI_UNF_HDMI_OPEN_PARA_S.....</a>	2
<a href="#">Table 7-3 Major attributes of HI_UNF_EDID_BASE_INFO_S .....</a>	3
<a href="#">Table 7-4 Major attributes of HI_UNF_HDMI_ATTR_S .....</a>	4
<a href="#">Table 7-5 Major attributes of HI_UNF_HDMI_INFOFRAME_S .....</a>	4
<a href="#">Table 7-6 Major attributes of HI_UNF_HDMI_STATUS_S .....</a>	5
<a href="#">Table 7-7 Major attributes of HI_UNF_HDMI_DELAY_S .....</a>	5



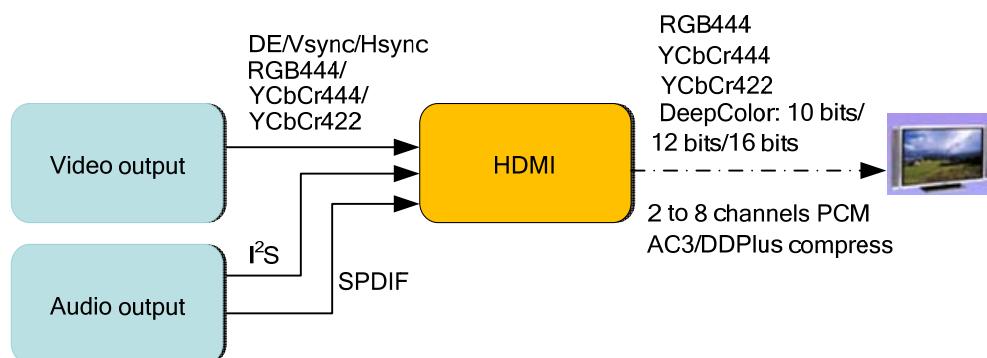
# 7 HDMI

## 7.1 Overview

The HDMI is a dedicated digital interface for video/audio transmission, which uses the digital video/audio interface technology. It can transmit audio and video signals simultaneously, and requires no digital-to-analog or analog-to-digital conversion before signal transmission. The high-bandwidth digital content protection (HDCP) can be used along with the HDMI to prevent video contents with copyright from being copied by unauthorized parties.

**Figure 7-1** Structure of the HDMI module

### HDMI structure



## 7.2 Important Concepts

[CEC]

Consumer electronics control

[TMDS]

Transition-minimized differential signaling

[EDID]

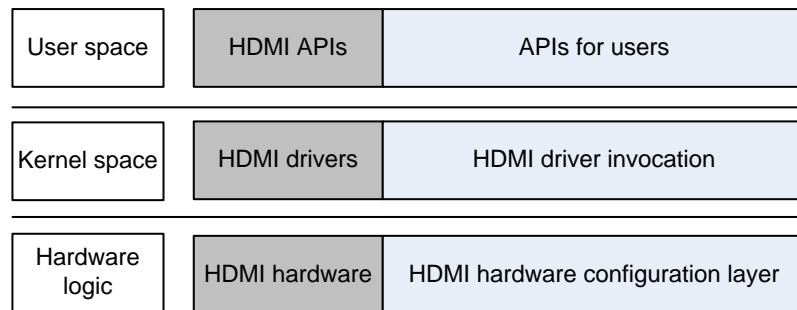


- Extended display identification data
  - [DDC]
- Display data channel
  - [HDCP]
- High-bandwidth digital content protection
  - [HPD]
- Hot plug detect

## 7.3 Features

[Figure 7-2](#) shows the HDMI functions.

**Figure 7-2** HDMI functions



The common interface for setting HDMI parameters is **hi\_unf\_hdmi.h**.

[Table 7-1](#) to [Table 7-5](#) describe the major attributes of callback function parameters.

**Table 7-1** Major attributes of HI\_UNF\_HDMI\_CALLBACK\_FUNC\_S

Parameter	Default Value	Description
pfnHdmiEventCallback	None	Callback function for registration
pPrivateData	None	Parameters of the registered callback function

**Table 7-2** Major attributes of HI\_UNF\_HDMI\_OPEN\_PARA\_S

Parameter	Default Value	Description
enDefaultMode	HI_UNF_HDMI_FORCE_NULL	HDMI output mode. The mode of HI_UNF_HDMI_FORCE_HDMI is recommended.



**Table 7-3** Major attributes of HI\_UNF\_EDID\_BASE\_INFO\_S

Parameter	Default Value	Description
bSupportHdmi	HI_FALSE	Whether the HDMI is supported by the device. If the HDMI is not supported by the device, the output mode is digital visual interface (DVI) output.
enNativeFormat	BUTT	Optimal program aspect ratio supported by the display device
bSupportFormat	HI_FALSE	Video formats supported by the display device
st3DInfo	NULL	3D display capability of the display device
stDeepColor	NULL	Color depth capability set supported by the display device
stColorMetry	NULL	Chrominance capability set supported by the display device
stColorSpace	NULL	Color space capability set supported by the display device
stAudioInfo	NULL	Audio capability set supported by the display device
u32AudioInfoNum	0	Number of valid audio files in the <b>stAudioInfo</b> structure
u8ExtBlockNum	0	Number of EDID extended blocks
stMfrsInfo	NULL	Vendor information of the display device
stCECAddr	NULL	CEC information
stPerferTiming	NULL	Optimal aspect ratio supported by the display device
u32MaxTMDSClock	0	Maximum TMDS clock
bSupportY420Format	HI_FALSE	Whether YUV420 is supported
stY420DeepColor	HI_FALSE	Whether YUV420 deep color is supported
bDolbySupport	HI_FALSE	Whether Dolby is supported
bHdrSupport	HI_FALSE	Whether HDR is supported



**Table 7-4** Major attributes of HI\_UNF\_HDMI\_ATTR\_S

Parameter	Default Value	Description
bEnableHdmi	HI_FALSE	HI_TRUE indicates HDMI, and other values indicate DVI.
bEnableVideo	HI_FALSE	Video enable
enVidOutMode	VIDEO_MODE_RGB444	HDMI video output mode, including VIDEO_MODE_YCBCR444, VIDEO_MODE_YCBCR422, and VIDEO_MODE_RGB444
enDeepColorMode	HI_UNF_HDMI_DEEP_COLOR_24BIT	Output mode of DeepColor
bXvYCCMode	HI_FALSE	xvYCC output mode enable
bEnableAudio	HI_FALSE	Audio enable
bEnableSpdInfoFrame	HI_FALSE	Whether to enable the information frame of Source Product Description
bEnableMpegInfoFrame	HI_FALSE	Whether to enable the information frame of MPEG Source
bHDCPEnable	HI_FALSE	Whether to transfer data in HDCP encryption mode  This mode is valid only after a valid HDCP key is burnt to the OTP. 0: disabled 1: enabled

**Table 7-5** Major attributes of HI\_UNF\_HDMI\_INFOFRAME\_S

Parameter	Default Value	Description
enInfoFrameType	0	Type of the information frame, including auxiliary video information (AVI) and audio (AUD).
unInforUnit	None	Union of information frames, including the structure of AVI and AUD information frames.



**Table 7-6** Major attributes of HI\_UNF\_HDMI\_STATUS\_S

Parameter	Default Value	Description
bConnected	0	Whether the device is connected
bSinkPowerOn	NULL	Whether the sink device is powered on

**Table 7-7** Major attributes of HI\_UNF\_HDMI\_DELAY\_S

Parameter	Default Value	Description
u32MuteDelay	0	Avmute delay
u32FmtDelay	0	Standard switching delay
bForceFmtDelay	FALSE	Forcible standard switching delay
bForceMuteDelay	FALSE	Forcible avmute delay

## 7.4 Development Guide

The HDMI module is used in the following scenarios:

- Detecting HDMI hot plug
- Sending CEC messages

### 7.4.1 Detecting HDMI Hot Plug

#### Scenario

In this scenario, the connection status of the RX end device is detected and the corresponding HDMI status is configured.

When the HDMI device connects to the RX device, it parses the capability set of the RX device for playing.

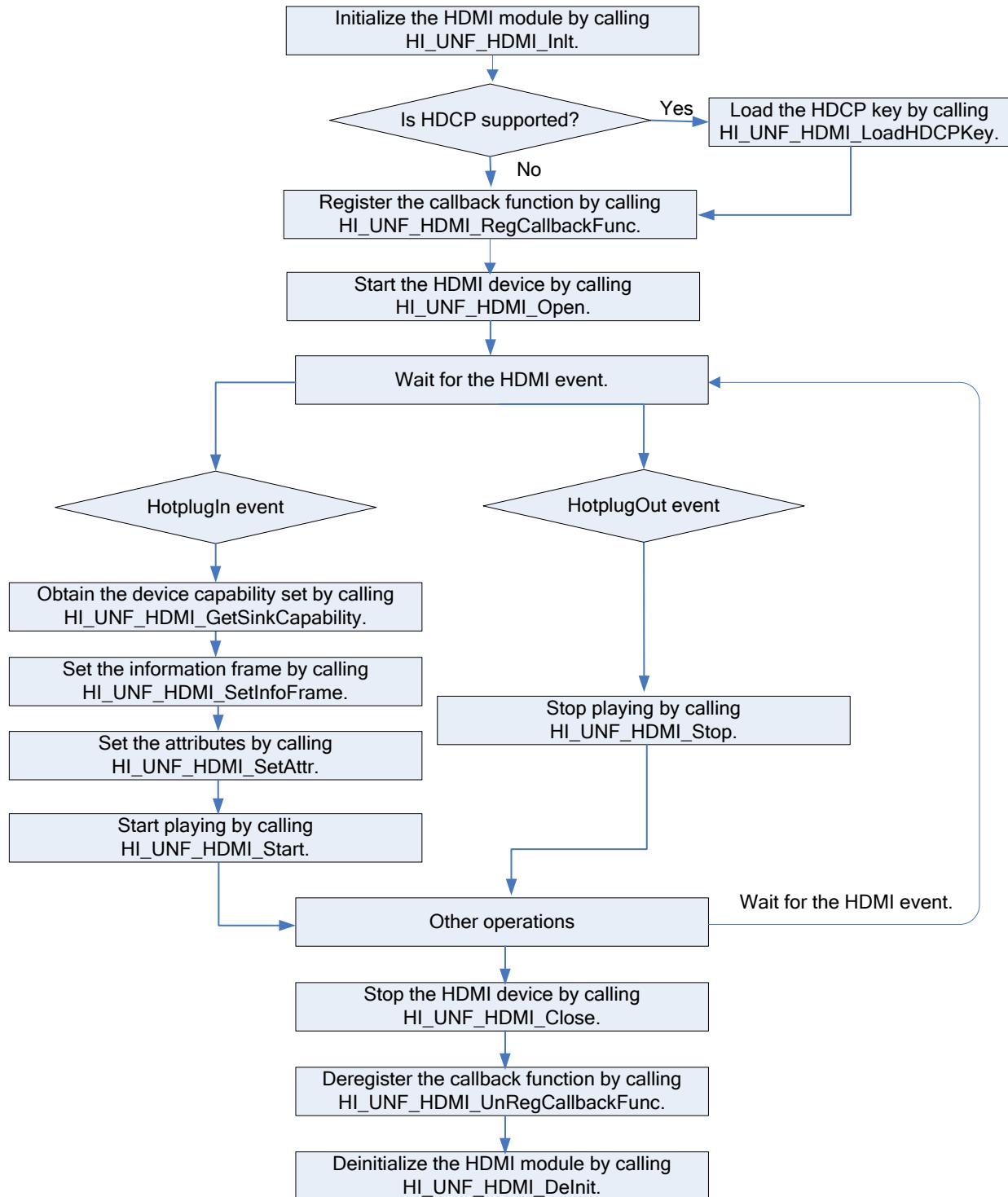
If the HDMI device disconnects from the RX device, the playback is stopped.

#### Working Process

[Figure 7-3](#) shows the process for using the HDMI module.



**Figure 7-3** Process for using the HDMI module



## Notes

- When the HDMI device is started, the HDMI output is normal only after the HDMI is configured in the processing function for the triggered HotPlug event and `HI_UNF_HDMI_Start` is called.



- HI\_UNF\_HDMI\_RegCallbackFunc and HI\_UNF\_HDMI\_LoadHDCPKey must be called before HI\_UNF\_HDMI\_Open.
- The HDMI can be called by multiple processes or threads on Linux. If multiple applications use the HDMI at the same time, each application must call HI\_UNF\_HDMI\_RegCallbackFunc before calling HI\_UNF\_HDMI\_Open. Before calling HI\_UNF\_HDMI\_Open, each application can call HI\_UNF\_HDMI\_RegCallbackFunc for multiple times to register multiple callback functions.
- It is recommended that the HDMI device attributes be set in the processing function of the HotPlug event.

## Samples

See \sample\common\hi\_adp\_hdmi.c, \sample\hdmi\_tsplay\hdmi\_tsplay.c, and \sample\hdmi\_tsplay\hdmi\_test\_cmd.c.

## 7.4.2 Sending HDMI CEC Messages

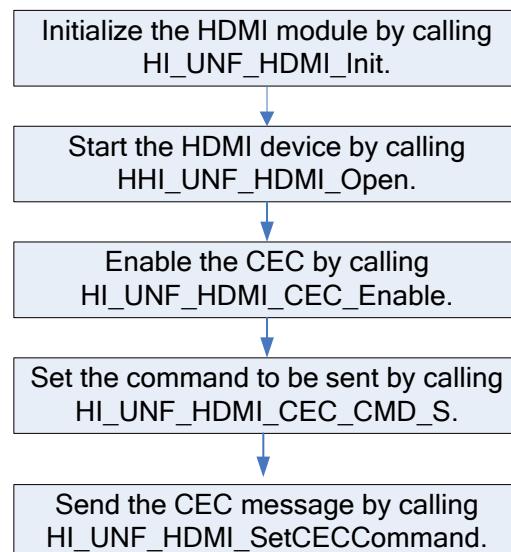
### Scenario

In this scenario, functions such as one-key standby or one-key wakeup are implemented by sending specific messages to the RX end based on the HDMI extended protocol customer electronics control (CEC).

### Working Process

Figure 7-4 shows the working process for the HDMI CEC.

**Figure 7-4** Working process for the HDMI CEC



### Notes

- The corresponding compilation option must be selected to support the CEC.



Run make menuconfig, choose MSP > HDMI Config, and select CEC Support.

- Some devices do not support the CEC function. Check whether the RX end device supports the CEC or specific CEC commands first.

## Samples

See `\sample\common\hi_adp_hdmi.c`, `\sample\hdmi_tsplay\hdmi_tsplay.c`, and `\sample\hdmi_tsplay\hdmi_test_cmd.c`.



# Contents

---

<b>8 HiFB .....</b>	<b>1</b>
8.1 Overview .....	1
8.1.1 Introduction .....	1
8.1.2 Comparison Between the HiFB and the Linux Framebuffer.....	2
8.1.3 Related Document.....	4
8.1.4 Loading Drivers .....	4
8.2 Important Concepts .....	6
8.3 Features .....	6
8.4 Development Guide .....	7
8.4.1 Displaying 2D Graphics.....	7
8.4.2 Displaying Index Pictures by Using the Palette .....	8
8.4.3 Displaying Pictures Dynamically by Using PAN_DISPLAY .....	12



## Figures

---

<b>Figure 8-1</b> Architecture of the HiFB.....	1
<b>Figure 8-2</b> Process for developing the HiFB .....	7



# Tables

---

<b>Table 8-1</b> Pixel formats supported by the HiFB on the HD platform .....	3
<b>Table 8-2</b> Mapping between vramn_size and the overlapped graphics layer.....	5
<b>Table 8-3</b> Tasks in each development phase of the HiFB .....	8



# 8 HiFB

## 8.1 Overview

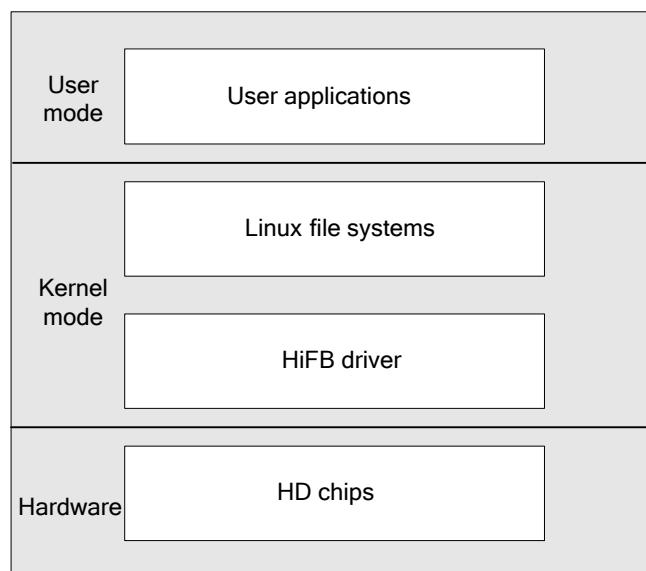
### 8.1.1 Introduction

As a module of HiSilicon digital MSP, the HiFB is used to manage overlapped graphics layers. In addition to the basic functions of the Linux framebuffer, the HiFB also provides extended functions, such as interlayer colorkey, interlayer colorkey mask, interlayer alpha, and origin offset.

## Architecture

Applications use the HiFB based on the Linux file system. [Figure 8-1](#) shows the architecture of the HiFB.

**Figure 8-1** Architecture of the HiFB





## Application Scenarios

The HiFB applies to the following scenarios:

- DirectFB system
  - The DirectFB system is compatible with the Linux framebuffer.
- Applications based on the Linux framebuffer
  - These applications can be rapidly ported to HiSilicon chips after a few changes or without any change.

### 8.1.2 Comparison Between the HiFB and the Linux Framebuffer

#### Management of Overlapped Graphics Layers

In the Linux framebuffer, a sub device ID corresponds to a video card. In the HiFB, a sub device ID corresponds to an overlapped graphics layer. The HiFB can manage multiple overlapped graphics layers. The number of managed graphics layers varies according to chips.

##### NOTE

Each device name of HiFB corresponds to a physical graphics layer. To be specific, /dev/fb0, /dev/fb1, and /dev/fb2 correspond to HD graphics layer 0, layer 1, and layer 2 respectively, /dev/fb3 corresponds to the hardware mouse layer, and /dev/fb4 corresponds to SD graphics layer 0. Typically, the default mapping between device names and physical graphics layers is used. Some platforms can dynamically bind graphics layers to different output devices. For example, an SD graphics layer can be bound to an HD output device. The default graphics layer, however, is still an SD graphics layer.

By setting module loading parameters, you can control one or more overlapped graphics layers managed by the HiFB, and operate the layers in the same way as operating common files.

#### Timing Control

The Linux framebuffer displays the contents of the physical display buffer on different output devices (such as the PC monitor, TV set, and LCD display) in multiple control modes. The control modes include sync timing mode, scanning mode, and sync signal mode and depend on the hardware configuration. At present, the HiFB does not support the preceding control modes.

#### Basic Functions and Extended Functions

The HiFB has the following basic functions:

- Maps/unmaps the physical display buffer to/from the virtual memory.
- Operates the physical display buffer in the same way as operating common files.
- Sets the display resolution and pixel format of the hardware. You can call the capability interface to obtain the information about the maximum resolution and pixel formats supported by each overlapped graphics layer. For details about the supported pixel formats, see [Table 8-1](#).
- Performs read, write, and display operations in any position of the physical display buffer.
- Sets and obtains 256-color palette if the overlapped graphics layer supports the index format.



**Table 8-1** Pixel formats supported by the HiFB on the HD platform

Overlapped Graphics Layer	Picture Type	Pixel Format
Overlapped graphics layer 0 (SD)	Index format	8 bpp
	16-bit format	RGB444, ARGB4444, RGB555, RGB565, ARGB1555
	32-bit format	RGB888, ARGB8888
Overlapped graphics layer 0 (HD)	Index format	8 bpp
	16-bit format	RGB444, ARGB4444, RGB555, RGB565, ARGB1555
	32-bit format	RGB888, ARGB8888
Overlapped graphics layer 1 (HD)	Index format	Not supported
	16-bit format	RGB444, ARGB4444, RGB555, RGB565, ARGB1555
	32-bit format	RGB888, ARGB8888
Overlapped graphics layer 2 (HD)	Index format	Not supported
	16-bit format	RGB444, ARGB4444, RGB555, RGB565, ARGB1555
	32-bit format	RGB888, ARGB8888

The HiFB has the following extended functions:

- Sets and obtains the alpha value of an overlapped graphics layer.
- Sets and obtains the colorkey value of an overlapped graphics layer.
- Sets the start position of the current overlapped graphics layer (relative to the origin offset of the screen).
- Sets and obtains the display status (displayed or hidden) of the current overlapped graphics layer.
- Obtains the vertical blanking interval of the current overlapped graphics layer.
- Sets the size of the physical display buffer and the number of managed overlapped graphics layers based on the module loading parameters.

The HiFB does not support the following basic functions of the Linux framebuffer:

- Sets and obtains the Linux framebuffer corresponding to the console.
- Obtains the real-time scanning information about the hardware.
- Obtains the information about the hardware.
- Sets the sync timing of the hardware.
- Sets the sync signal mechanism of the hardware.



## 8.1.3 Related Document

See the *HMS API Development Reference*.

## 8.1.4 Loading Drivers

### Principles

When some Linux framebuffer drivers such as versa are running, you cannot change the display attributes such as resolution, color depth, and timing. To solve this problem, Linux provides a mechanism for transferring corresponding parameters to the Linux framebuffer when the kernel is booted or drivers are loaded. The kernel boot parameters can be set in the kernel loader. When HiFB drivers are being loaded, only the size of the physical display buffer can be set.

Before loading the HiFB driver **hifb.ko**, you must load the standard frame buffer driver **fb.ko** to the kernel by running **modprobe fb** and load the TDE driver **tde.ko**.



**fb.ko** is a driver inherent in the Linux framebuffer. You can compile **fb.ko** in the kernel by using the kernel compilation option. In this way, you do not need to load **fb.ko** manually before loading **hifb.ko**.

### Setting Parameters

The HiFB can set the size of the physical display buffer corresponding to each overlapped graphics layer. The size of the physical display buffer determines the maximum size of the physical display buffer for the HiFB and the maximum virtual resolution. If the size of the physical display buffer is set by transferring parameters when HiFB drivers are being loaded, the size cannot be changed after being set.

The following is the syntax for setting the size of the physical display buffer corresponding to each overlapped graphics layer:

```
video="hifb:vram0_size:xxx, vram2_size:xxx,..."
```



#### NOTE

- Options are separated with commas (,).
- The option and option value are separated with a colon (:).
- If the physical display buffer size is not set in parameter transfer mode when the HiFB drivers are being loaded, the system allocates the display buffer by default. For the Hi379X, you can dynamically set the frame buffer size by running the **insmod** command, or set the default display buffer size for the graphics layer by compiling the scripts using **cfg.mak** or running **make menuconfig**. If a graphics layer is not used, you can set its buffer size to **0** to reduce the memory occupied. In addition, you can check whether the buffer is allocated to a graphics layer by running **\cat\proc\media-mem**. For example, **hifb\_layer0** corresponds to the frame buffer for HD graphics layers, and **hifb\_layer4** corresponds to the frame buffer for SD graphics layers.

**vram $n$ \_size:xxx** indicates that  $xxx$  KB physical display buffer is configured for the overlapped graphics layer  $n$ .

For a standard frame buffer, the relationship between **vram $n$ \_size** and virtual display resolution is as follows:

$$\text{vram}(n)_{\text{size}} \geq \text{xres\_virtual} \times \text{yres\_virtual} \times \text{bpp}$$

Where, **xres\_virtual** **yres\_virtual** indicates the virtual resolution, and **bpp** indicates the number of bytes for each pixel.



For an extended frame buffer, the relationship between `vramn_size` and display resolution is as follows (the required memory depends on the value of `displaySize`, pixel format of the graphics layer, and refresh mode):

$$\text{Vramn\_size} \geq \text{displayWidth} \times \text{displayHeight} \times \text{bpp} \times \text{BufferMode}$$

**BufferMode** indicates the number of display buffers. It is **0** when the HiGo is in single-buffer mode, **1** when the HiGo is in double-buffer mode, and **2** when the HiGo is in the triple-buffer mode or over mode.

For example, if the resolution of an HD graphics layer is 1280 x 720, and the pixel format in dual-buffer mode is ARGB8888, the required memory is calculated as follows:

$$\text{vram2\_size} = 1280 \times 720 \times 4 \times 2 = 7200 \text{ KB}$$

In most cases, if the interfaces related to HiGo are used, the required memory is configured based on the extended frame buffer.

#### NOTE

The value of `vramn_size` must be a multiple of `PAGE_SIZE` (4 KB). Otherwise, the HiFB rounds up the value to a multiple of `PAGE_SIZE`.

[Table 8-2](#) describes the mapping between `vramn_size` and the overlapped graphics layer.

**Table 8-2** Mapping between `vramn_size` and the overlapped graphics layer

Overlapped Graphics Layer	Character String	Description
Overlapped graphics layer 0	<code>vram0_size</code>	Size (in KB) of the physical display buffer corresponding to overlapped graphics layer 0 (main HD layer)
Overlapped graphics layer 1	<code>vram1_size</code>	Size (in KB) of the physical display buffer corresponding to overlapped graphics layer 1 (HD hardware scrolling teletext layer)
Overlapped graphics layer 2	<code>vram2_size</code>	Size (in KB) of the physical display buffer corresponding to overlapped graphics layer 2 (HD hardware scrolling caption layer)
Overlapped graphics layer 3	<code>vram3_size</code>	Size (in KB) of the physical display buffer corresponding to overlapped graphics layer 3 (HD hardware mouse layer)
Overlapped graphics layer 4	<code>vram4_size</code>	Size (in KB) of the physical display buffer corresponding to overlapped graphics layer 4 (SD graphics layer)

You need to set the number of overlapped graphics layers managed by the HiFB from a global perspective, and allocate appropriate display buffers for each overlapped graphics layer.



## Configuration Examples

This section describes the examples of enabling the HiFB to manage one or more overlapped graphics layers.

### NOTE

The driver of the HiFB is **hifb.ko**.

- Enabling the HiFB to manage a single overlapped graphics layer

If only overlapped graphics layer 4 (SD graphics layer) needs to be managed by the HiFB, the maximum virtual resolution is 720 x 576, and the pixel format is ARGB8888, the minimum display buffer for overlapped graphics layer 4 is 1620 KB (720 x 4 x 576). In this case, the parameters are set as follows:

```
insmod hifb.ko video="hifb:vram4_size:1620"
```

If the HiGo is in triple-buffer mode or over mode, the value of **vram0\_size** must be multiplied by 2. That is, the parameters are set as follows:

```
insmod hifb.ko video="hifb:vram4_size:3640"
```

- Enabling the HiFB to manage multiple overlapped graphics layers

If overlapped graphics layer 0 (HD graphics layer) and layer 4 (SD graphics layer) need to be managed by the HiFB, the maximum virtual resolution is 720 x 576 and the pixel format is ARGB8888, and the maximum virtual resolution is 1280 x 720 and the pixel format is ARGB8888 for layer 0, the parameter is set as follows:

```
insmod hifb.ko video="hifb:vram0_size:1620, vram2_size: 3600"
```

If graphics layer 2 uses dual buffers, the parameters are set as follows:

```
insmod hifb.ko video="hifb:vram0_size:3600, vram2_size: 7200"
```

## 8.2 Important Concepts

None

## 8.3 Features

The HiFB is mainly used to display 2D pictures (directly operating the physical display buffer).

The common APIs are as follows:

- FBIOPUT\_VSCREENINFO**: Sets the variable screen information by calling the ioctl function.
- FBIOGET\_VSCREENINFO**: Obtains the fixed screen information by calling the ioctl function.
- FBIOPUT\_ALPHA\_HIFB**: Sets the alpha for a graphics layer by calling the ioctl function.



## 8.4 Development Guide

The HiFB module is used in the following scenarios:

- Displaying 2D pictures
- Displaying index pictures by using the palette
- Displaying pictures dynamically by using PAN\_DISPLAY

### 8.4.1 Displaying 2D Graphics

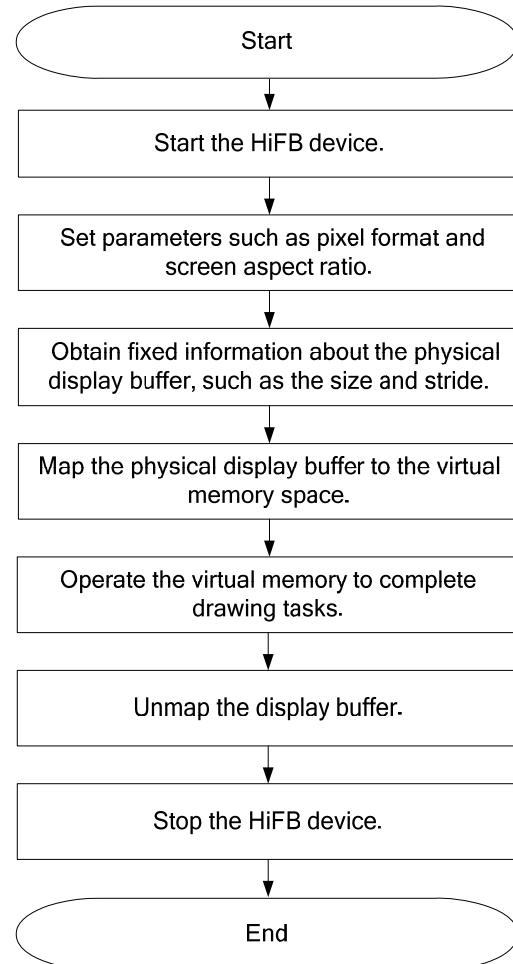
#### Scenario

The HiFB is used to display 2D graphics.

#### Working Process

[Figure 8-2](#) shows the process for developing the HiFB.

**Figure 8-2** Process for developing the HiFB





**NOTE**

Stride = Row width x Number of bytes of each pixel

To develop the HiFB, perform the following steps:

- Step 1** Start the specific HiFB device by calling the open function.
- Step 2** Set corresponding parameters such as the pixel format of the HiFB, screen height, and screen width by calling the ioctl function. For details, see the HiFB section in the *HMS API Development Reference*.
- Step 3** Obtain the fixed information such as the size of the physical display buffer allocated by the HiFB and the stride by calling the ioctl function. You can also call the ioctl function to enable the extended functions of the HiFB such as interlayer colorkey, interlayer colorkey mask, interlayer alpha, and origin offset.
- Step 4** Map the physical display buffer to the virtual memory by calling the mmap function.
- Step 5** Operate the virtual memory to complete specific drawing tasks. In this step, you can enable the page flip with the dual buffers function of the HiFB to achieve certain drawing effects (such as book flipping).
- Step 6** Unmap the display buffer by calling the munmap function.
- Step 7** Stop the HiFB device by calling the close function.

----End

[Table 8-3](#) describes the task in each development phase of the HiFB.

**Table 8-3** Tasks in each development phase of the HiFB

Phase	Task
Initialization phase	Sets display attributes and maps the physical display buffer to the virtual memory.
Drawing phase	Performs specific drawing tasks.
End phase	Clears resources.

## Notes

If the virtual resolution is changed, the value of the parameter `fb_fix_screeninfo::line_length` (stride) of the HiFB is changed accordingly. To ensure the normal running of the drawing program, you are advised to set the variable parameter `fb_var_screeninfo` of the HiFB before obtaining the value of the fixed parameter `fb_fix_screeninfo::line_length`.

## Sample

See `sample\fb\sample_fb.c`.

### 8.4.2 Displaying Index Pictures by Using the Palette

This instance describes how to display a 720 x 576 index picture with 256 colors by using the palette.



In the following instance, the first 256 x 4 bytes of the **colormap.bits** file store palette data, and the rest bytes store index picture data.

[Reference Code]

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <linux/fb.h>
#include "hifb.h"

#define CMAP_DATA      "./res/colormap.bits"

unsigned short cmap_red[256];
unsigned short cmap_green[256];
unsigned short cmap_blue[256];

int hifb_PaletteCreate(void)
{
    FILE *fp;
    int i;
    int cmap[256];
    unsigned char r, g, b;
    fp = fopen(CMAP_DATA, "rb");
    if(NULL == fp)
    {
        printf("open %s failed!\n", CMAP_DATA);
        return -1;
    }

    fread(cmap, 4, 256, fp);
    fclose(fp);
    for(i = 0; i < 256; i++)
    {
        b = cmap[i];
        g = cmap[i]>>8;
        r = cmap[i]>>16;
        cmap_red[i] = r;
        cmap_green[i] = g;
        cmap_blue[i] = b;
    }
    return 0;
}
```



```
int hifb_GetImageData(unsigned char *ptr)
{
    FILE *fp;
    fp = fopen(CMAP_DATA, "rb");
    if(NULL == fp)
    {
        printf("open %s failed!\n", CMAP_DATA);
        return -1;
    }

    fseek(fp, 256*4, SEEK_SET);
    fread(ptr, 1, 720*576, fp);
    fclose(fp);

    return 0;
}

int main()
{
    int fd;
    struct fb_fix_screeninfo fix;
    struct fb_var_screeninfo var;
    unsigned char *pShowScreen;
    unsigned char *pHideScreen;
    HIFB_POINT_S stPoint = {0, 0};
    struct fb_cmap himap;

    /*1. open Framebuffer device overlay 0*/
    fd = open("/dev/fb0", O_RDWR);
    if(fd < 0)
    {
        printf("open fb0 failed!\n");
        return -1;
    }

    /*2. set the screen original position*/
    if (ioctl(fd, FBIOPUT_SCREEN_ORIGIN_HIFB, &stPoint) < 0)
    {
        printf("set screen original show position failed!\n");
        return -1;
    }

    /*3. get the variable screen info*/
    if (ioctl(fd, FBIOGET_VSCREENINFO, &var) < 0)
```



```
{  
    printf("Get variable screen info failed!\n");  
    close(fd);  
    return -1;  
}  
  
/*4. modify the variable screen info*/  
var.xres = var.xres_virtual = 720;  
var.yres = var.yres_virtual = 576;  
var.bits_per_pixel = 8;  
  
/*5. set the variable screeninfo*/  
if (ioctl(fd, FBIOPUT_VSCREENINFO, &var) < 0)  
{  
    printf("Put variable screen info failed!\n");  
    close(fd);  
    return -1;  
}  
  
/*6. get the fix screen info and map the physical video memory for  
user use*/  
if (ioctl(fd, FBIOGET_FSCREENINFO, &fix) < 0)  
{  
    printf("Get fix screen info failed!\n");  
    close(fd);  
    return -1;  
}  
  
pShowScreen = mmap(NULL, fix.smem_len, PROT_READ|PROT_WRITE,  
MAP_SHARED, fd, 0);  
  
/*7. set color map*/  
hifb_PaletteCreate();  
himap.start = 0;  
himap.len = 256;  
himap.red = cmap_red;  
himap.green = cmap_green;  
himap.blue = cmap_blue;  
himap.transp = 0;  
  
if (ioctl(fd, FBIOPUTCMAP, &himap) < 0)  
{  
    printf("fb ioctl put cmap err!\n");  
    close(fd);  
}
```



```
        return -1;
    }

    /*8. load the image data to screen*/
    hifb_GetImageData(pShowScreen);

    printf("Enter to quit!\n");
    getchar();
    /*9. close the Framebuffer device*/
    close(fd);

    return 0;
}
```

### 8.4.3 Displaying Pictures Dynamically by Using PAN\_DISPLAY

This instance describes how to display fifteen 640 x 352 pictures consecutively by using PAN\_DISPLAY to achieve dynamic display effect.

In the following instance, each file stores pure data in the pixel format of ARGB1555 (picture data without attached information).

[Reference Code]

```
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <linux/fb.h>
#include "hifb.h"

#define IMAGE_WIDTH      640
#define IMAGE_HEIGHT     352
#define IMAGE_SIZE       (640*352*2)
#define IMAGE_NUM        14
#define IMAGE_PATH       "./res/%d.bits"

static struct fb_bitfield g_r16 = {10, 5, 0};
static struct fb_bitfield g_g16 = {5, 5, 0};
static struct fb_bitfield g_b16 = {0, 5, 0};
static struct fb_bitfield g_a16 = {15, 1, 0};

int main()
{
    int fd;
    int i;
    struct fb_fix_screeninfo fix;
```



```
struct fb_var_screeninfo var;
unsigned char *pShowScreen;
unsigned char *pHideScreen;
HIFB_POINT_S stPoint = {40, 112};
FILE *fp;
char image_name[128];

/*1. open Framebuffer device overlay 0*/
fd = open("/dev/fb0", O_RDWR);
if(fd < 0)
{
    printf("open fb0 failed!\n");
    return -1;
}

/*2. set the screen original position*/
if (ioctl(fd, FBIOPUT_SCREEN_ORIGIN_HIFB, &stPoint) < 0)
{
    printf("set screen original show position failed!\n");
    return -1;
}

/*3. get the variable screen info*/
if (ioctl(fd, FBIOGET_VSCREENINFO, &var) < 0)
{
    printf("Get variable screen info failed!\n");
    close(fd);
    return -1;
}

/*4. modify the variable screen info
   the screen size: IMAGE_WIDTH*IMAGE_HEIGHT
   the virtual screen size: IMAGE_WIDTH*(IMAGE_HEIGHT*2)
   the pixel format: ARGB1555
*/
var.xres = var.xres_virtual = IMAGE_WIDTH;
var.yres = IMAGE_HEIGHT;
var.yres_virtual = IMAGE_HEIGHT*2;

var.transp= g_a16;
var.red = g_r16;
var.green = g_g16;
var.blue = g_b16;
var.bits_per_pixel = 16;
```



```
/*5. set the variable screeninfo*/
if (ioctl(fd, FBIOPUT_VSCREENINFO, &var) < 0)
{
    printf("Put variable screen info failed!\n");
    close(fd);
    return -1;
}

/*6. get the fix screen info*/
if (ioctl(fd, FBIOGET_FSCREENINFO, &fix) < 0)
{
    printf("Get fix screen info failed!\n");
    close(fd);
    return -1;
}

/*7. map the physical video memory for user use*/
/* pShowScreen is the start address of the frame buffer, that is, display
buffer 1*/
pShowScreen = mmap(NULL, fix.smem_len, PROT_READ|PROT_WRITE, MAP_SHARED,
fd, 0);
/* pHideScreen is the start address of display buffer 2.
pHideScreen = pShowScreen + IMAGE_SIZE;
memset(pShowScreen, 0, IMAGE_SIZE);

/*8. Load the bitmaps from file to hide screen and set pan display the
hide screen*/
for(i = 0; i < IMAGE_NUM; i++)
{
    sprintf(image_name, IMAGE_PATH, i);
    fp = fopen(image_name, "rb");
    if(NULL == fp)
    {
        printf("Load %s failed!\n", image_name);
        close(fd);
        return -1;
    }
    /*Read data into the display buffer. */
    fread(pHideScreen, 1, IMAGE_SIZE, fp);
    fclose(fp);
    usleep(10);
    /*Switch the display buffer.*/
    if(i%2)
```



```
{  
    var.yoffset = 0;  
    pHideScreen = pShowScreen + IMAGE_SIZE;  
}  
else  
{  
    var.yoffset = IMAGE_HEIGHT;  
    pHideScreen = pShowScreen;  
}  
/*Refresh the display. Contents in the display buffer is read and  
displayed on the screen by frame.  
if (ioctl(fd, FBIOPAN_DISPLAY, &var) < 0)  
{  
    printf("FBIOPAN_DISPLAY failed!\n");  
    close(fd);  
    return -1;  
}  
}  
  
printf("Enter to quit!\n");  
getchar();  
  
/*9. close the Framebuffer device*/  
close(fd);  
  
return 0;
```



# Contents

---

<b>9 TDE .....</b>	<b>1</b>
9.1 Overview .....	1
9.2 Important Concepts .....	1
9.3 Features .....	1
9.4 Development Guide .....	3
9.4.1 Working Process .....	3
9.4.2 Programming Guide.....	4
9.4.3 Application Scenarios .....	4



# Figures

<b>Figure 9-1</b> Working process of the TDE.....	3
<b>Figure 9-2</b> Background bitmap (corresponding to stSrc1 described in the example).....	12
<b>Figure 9-3</b> Foreground bitmap (corresponding to stSrc2 described in the example).....	12
<b>Figure 9-4</b> Mask bitmap (corresponding to stMask in the format of A1 described in the example) .....	13
<b>Figure 9-5</b> Output bitmap (corresponding to stDst described in the example).....	13



# 9 TDE

## 9.1 Overview

The TDE provides graphics drawing functions using hardware, which significantly reduces the CPU usage and makes full use of the bandwidth of the memory. In addition, the TDE supports operations on raster or macroblock bitmaps, such as transferring, filling, drawing lines or rectangles, anti-flicker, scaling, and color space conversion (CSC) or color format conversion.

## 9.2 Important Concepts

None

## 9.3 Features

The TDE provides the following APIs to implement corresponding functions:

- `HI_TDE2_BeginJob`: Returns the identifier of a created TDE task. This identifier must be stored for adding TDE operations to the task.
- `HI_TDE2_Bitblit`: Adds bit block transfer (Blit) operations (performed on raster bitmaps) to a created TDE task. The required information is as follows:
  - Identifier of the created task to which operations are to be added. You can obtain this identifier by calling `HI_TDE2_BeginJob`.
  - Information about source bitmap 1 (background bitmap), source bitmap 2 (foreground bitmap), target bitmap, and corresponding operating areas.
  - Configured functions and parameters, such as whether to perform the operations such as ROP, alpha blending, anti-flicker, and scaling, and the colorkey configuration information.
- `HI_TDE2_MbBlit`: Adds Blit operations (performed on macroblock bitmaps) to a created TDE task. The required information is as follows:
  - Identifier of the created task to which operations are to be added. You can obtain this identifier by calling `HI_TDE2_BeginJob`.
  - Luminance and chrominance of the source bitmap, information about the target bitmap, and corresponding operating areas.



- Configured functions and parameters.
- HI\_TDE2\_SolidDraw: Adds the line or rectangle drawing operations to a created TDE task. The required information is as follows:
  - Identifier of the created task to which operations are to be added. You can obtain this identifier by calling HI\_TDE2\_BeginJob.
  - Information about the target bitmap on which lines are to be drawn and corresponding operating areas.
  - Drawing color, drawing width, and drawing height.
  - Configured functions and parameters.
- HI\_TDE2\_QuickCopy: Adds the DMA operation to a created TDE task. The required information is as follows:
  - Identifier of the created task to which operations are to be added. You can obtain this identifier by calling HI\_TDE2\_BeginJob.
  - Information about the source bitmap, target bitmap, and corresponding operating areas.
- HI\_TDE2\_QuickFill: Adds the fast filling operation to a created TDE task. The required information is as follows:
  - Identifier of the created task to which operations are to be added. You can obtain this identifier by calling HI\_TDE2\_BeginJob.
  - Information about the target bitmap and corresponding operating area.
  - Required filling value.
- HI\_TDE2\_QuickResize: Adds the fast scaling operation to a created TDE task. The required information is as follows:
  - Identifier of the created task to which operations are to be added. You can obtain this identifier by calling HI\_TDE2\_BeginJob.
  - Information about the source bitmap, target bitmap, and corresponding operating areas.
  - Scaling ratio that depends on the input operating area and output operating area.
- HI\_TDE2\_QuickDeflicker: Adds the fast anti-flicker operation to a created TDE task. The required information is as follows:
  - Identifier of the created task to which operations are to be added. You can obtain this identifier by calling HI\_TDE2\_BeginJob.
  - Information about the source bitmap, target bitmap, and corresponding operating areas.
- HI\_TDE2\_BitmapMaskRop: Adds the mask ROP operation to a created TDE task. The required information is as follows:
  - Identifier of the created task to which operations are to be added. You can obtain this identifier by calling HI\_TDE2\_BeginJob.
  - Information about source bitmap 1 (background bitmap), source bitmap 2 (foreground bitmap), mask bitmap (bitmap involved in the mask operation), target bitmap after the mask operation, and corresponding operating areas.
- HI\_TDE2\_BitmapMaskBlend: Adds the mask blend operation to a created TDE task. The required information is as follows:
  - Identifier of the created task to which operations are to be added. You can obtain this identifier by calling HI\_TDE2\_BeginJob.



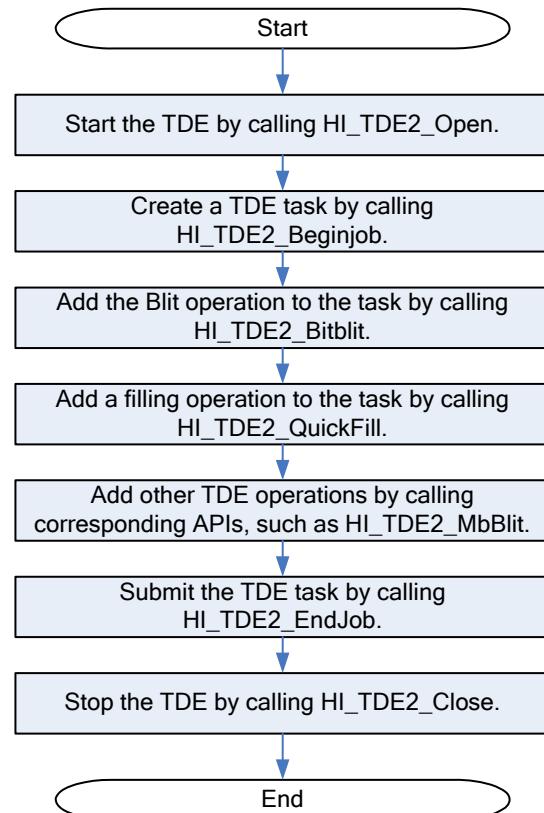
- Information about source bitmap 1 (background bitmap), source bitmap 2 (foreground bitmap), mask bitmap (bitmap involved in the mask operation), target bitmap after the mask operation, and corresponding operating areas.
- `HI_TDE2_EndJob`: Submits a created TDE task. The operations that are added before `HI_TDE2_EndJob` is called are temporarily suspended. That is, the TDE operations are performed only after `HI_TDE2_EndJob` is called. The required information is as follows:
  - Identifier of a created task. The identifier shows whether the task is a block task. If the task is a block task, the identifier also shows the timeout period.If you submit a non-block task by calling `HI_TDE2_EndJob`, you can check whether the task is complete by calling `HI_TDE2_WaitForDone`. The required parameter is the identifier of the specified task handle.
- A TDE task is performed when the TDE is idle (asynchronization) or the VO sync signal (synchronization) is received. Currently the synchronization mode is not supported.
- `HI_TDE2_CancelJob`: Cancels a task. When a task is created, the driver allocates some memory space. If a task fails, you need to cancel the task and release the memory by calling `HI_TDE2_CancelJob`.

## 9.4 Development Guide

### 9.4.1 Working Process

[Figure 9-1](#) shows the working process of the TDE.

**Figure 9-1** Working process of the TDE





## CAUTION

- You must call HI\_TDE2\_Open before calling other APIs. HI\_TDE2\_Open can be called repeatedly for multiple tasks.
- You must call HI\_TDE2\_Close after calling other APIs. Repeatedly calling HI\_TDE2\_Close has no effect.

### 9.4.2 Programming Guide

To use the TDE, perform the following steps:

**Step 1** Start a TDE device by calling HI\_TDE2\_Open.

**Step 2** Create a TDE task by calling HI\_TDE2\_BeginJob. If a TDE task is created successfully, its identifier TDE\_HANDLE is returned.

**Step 3** Add TDE operations to the TDE task. The supported TDE operations are as follows:

- HI\_TDE2\_QuickCopy: Copies and transfers a bitmap.
- HI\_TDE2\_QuickFill: Fills in a rectangle.
- HI\_TDE2\_QuickResize: Scales a raster bitmap.
- HI\_TDE2\_QuickDeflicker: Performs anti-flicker on a raster bitmap.
- HI\_TDE2\_Bitblit: Transfers two bitmaps that support additional operations.
- HI\_TDE2\_SolidDraw: Transfers the filled rectangles and bitmaps that support additional operations.
- HI\_TDE2\_MbBlit: Transfers a macroblock bitmap to a raster bitmap.
- HI\_TDE2\_BitmapMaskRop: Masks the ROP performed on an A1 bitmap.
- HI\_TDE2\_BitmapMaskBlend: Masks one or more blending operations performed on an A8 bitmap.

**Step 4** Submit the TDE task by calling HI\_TDE2\_EndJob. The TDE starts to perform the submitted operations in sequence.

**Step 5** Stop the TDE device by calling HI\_TDE2\_Close.

----End

### 9.4.3 Application Scenarios

The TDE module is used in the following scenarios:

- Transferring a raster bitmap
- Transferring a macroblock bitmap
- Setting three source bitmaps



### 9.4.3.1 Transferring a Raster Bitmap

#### Scenario

After the scaling and anti-flicker operations, the source bitmap is transferred to the address of the target bitmap. After the ROP exclusive OR operation is performed on the source bitmap and target bitmap, the operation result is rapidly copied to the display buffer.

#### Working Process

To transfer a raster bitmap, perform the following steps:

- Step 1** Start and initialize a TDE device by calling HI\_TDE2\_Open.
- Step 2** Create a TDE task by calling HI\_TDE2\_BeginJob.
- Step 3** Set the parameters and sizes of bitmaps and operating areas related to the ROP operation, and add the transfer operation to the task by calling HI\_TDE2\_BitBlt.
- Step 4** Copy the operation result to the display buffer by calling HI\_TDE2\_QuickCopy.
- Step 5** Submit the task in asynchronous and block mode by calling HI\_TDE2\_EndJob.
- Step 6** Stop the TDE device by calling HI\_TDE2\_Close.

----End

[Reference Code]

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "hi_tde_api.h"

#define TDE_BMP_WIDTH 128
#define TDE_BMP_HEIGHT 128

HI_S32 TDE_Sample1()
{
    HI_S32 s32Ret;

    /*The default value of the TDE2_OPT_S parameter is 0.*/
    TDE2_OPT_S stOpt = {0};

    /*The default values of the TDE2_SURFACE_S parameters are 0.*/
    TDE2_SURFACE_S stSrc = {0};
    TDE2_SURFACE_S stDst = {0};
    TDE2_SURFACE_S stDisp = {0};

    TDE2_RECT_S stSrcRect = {0};
    TDE2_RECT_S stDstRect = {0};
    TDE2_RECT_S stDispRect = {0};
```



```
TDE_HANDLE s32Handle;

if( access( "/dev/hi_tde" ,F_OK ) )
{
    printf("Device file doesn't exist.\n");
    return HI_FAILURE;
}

s32Ret = HI_TDE2_Open();
if(HI_SUCCESS != s32Ret)
{
    printf("Fail to open TDE device.\n");
    return HI_FAILURE;
}

/*Set the ROP parameters to bitwise exclusive OR.*/
stOpt.enAluCmd = TDE2_ALUCMD_ROP;
stOpt.enRopCode_Alpha = TDE2_ROP_MERGEOPEN;//The alpha value of source
2 is copied and then used.
stOpt.enRopCode_Color = TDE2_ROP_COPYOPEN;//The color values are ROP
ORed.
stOpt.bDeflicker = HI_TRUE;//Anti-flicker is enabled.
stOpt.bResize = HI_TRUE;//Scaling is enabled.

/*Information about the source bitmap to be filled*/
stSrc.enColorFmt = TDE2_COLOR_FMT_ARGB4444;
stSrc.u32PhyAddr = g_u32SrcBmpPhy;//There is picture data at this
physical address.
stSrc.ul6Height = TDE_BMP_HEIGHT;
stSrc.ul6Width = TDE_BMP_WIDTH;
stSrc.ul6Stride = TDE_BMP_WIDTH * 2;

stSrcRect.s32Xpos = 0;
stSrcRect.s32Ypos = 0;
/*The 28 pixels on the right of the input operating area are
ignored.*/
stSrcRect.u32Width = TDE_BMP_WIDTH - 28;
/*The 28 pixels on the bottom of the input operating area are
ignored.*/
stSrcRect.u32Height = TDE_BMP_HEIGHT - 28;

/*The width, height, and format of the target bitmap are the same as
those of the source bitmap.*/
memcpy(&stDst, &stSrc, sizeof(TDE2_SURFACE_S));
stDst.pu8PhyAddr = g_u32DstBmpPhy;//A memory space exists at this
```



physical address.

```
memcpy(&stDstRect, &stSrcRect, sizeof(TDE2_RECT_S));
/*Scale the size of a bitmap from 100 x 100 to 128 x 128.*/
stDstRect.u32Width = TDE_BMP_WIDTH;
stDstRect.u32Height = TDE_BMP_HEIGHT;
/*The parameter values of the displayed bitmap are the same as those
of the target bitmap.*/
memcpy(&stDisp, &stDst, sizeof(TDE2_SURFACE_S));
stDisp.u32PhyAddr = g_u32DispPhy;
memcpy(&stDispRect, &stDstRect, sizeof(TDE2_RECT_S));

/*Create a TDE task.*/
s32Handle = HI_TDE2_BeginJob();
if(HI_ERR_TDE_INVALID_HANDLE == s32Handle)
{
    printf("Fail to create Job.\n");
    HI_TDE2_Close();
    return HI_FAILURE;
}

/*Perform operations on the bitmaps stSrc and stDst, and overwrite the
result stored at the address of stDst with the operation result.*/
s32Ret = HI_TDE2_Bitblit(s32Handle, &stDst, &stDstRect,
                         &stSrc, &stSrcRect, &stDst, &stDstRect, &stOpt);
if(HI_SUCCESS != s32Ret)
{
    printf("Fail to add blit to Job.\n");
    HI_TDE2_CancelJob(s32Handle); //The task is canceled due to errors.
    HI_TDE2_Close();
    return HI_FAILURE;
}

/*Add the fast copy operation to the TDE task.*/
s32Ret = HI_TDE2_QuickCopy(s32Handle, &stDst, &stDstRect, &stDisp,
                           &stDispRect);
if(HI_SUCCESS != s32Ret)
{
    printf("Fail to add quick copy to Job.\n");
    HI_TDE2_CancelJob(s32Handle); //The task is canceled due to errors.
    HI_TDE2_Close();
    return HI_FAILURE;
}
```



```
/*Submit the TDE task in asynchronous and block mode.*/
s32Ret = HI_TDE2_EndJob(s32Handle, HI_FALSE, HI_TRUE, 100);
if(HI_SUCCESS != s32Ret)
{
    printf("Fail to End Job.\n");
    HI_TDE2_Close();
    return HI_FAILURE;
}

/*Ensure that the task is complete.*/
s32Ret = HI_TDE2_WaitForDone(s32Handle);
if (HI_ERR_TDE_QUERY_TIMEOUT == s32Ret)
{
    printf("Wait For Done Time Out.\n");
    HI_TDE2_Close();
    return HI_FAILURE;
}

/*The TDE completes all submitted tasks even a timeout occurs.*/
HI_TDE2_Close();
return HI_SUCCESS;
}
```

#### 9.4.3.2 Transferring a Macroblock Bitmap

##### Scenario

After the scaling and anti-flicker operations, a YCbCr420MB bitmap is output to the target bitmap as a YCbCr888 bitmap. Then, the operation result is transferred to the display buffer, and the format of the result is converted into ARGB4444.

##### Working Process

To transfer a macroblock bitmap, perform the following steps:

- Step 1** Start and initialize a TDE device by calling HI\_TDE2\_Open.
- Step 2** Create a TDE task by calling HI\_TDE2\_BeginJob.
- Step 3** Set the parameters related to the macroblock operations, and add the transfer operation to the TDE task by calling HI\_TDE2\_MbBlit.
- Step 4** Convert the format of the operation result by calling HI\_TDE2\_Bitblit, and transfer the result to the display buffer.
- Step 5** Calculate the height of the scaled bitmap, and perform vertical scaling by calling HI\_TDE\_VerticalScale.
- Step 6** Submit the task in asynchronous and non-block mode by calling HI\_TDE2\_EndJob.
- Step 7** Wait until the task is complete by calling HI\_TDE2\_WaitForDone.



**Step 8** Stop the TDE device by calling HI\_TDE2\_Close.

----End

[Reference Code]

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "hi_tde_api.h"

#define TDE_BMP_WIDTH 128
#define TDE_BMP_HEIGHT 128

HI_S32 TDE_Sample2()
{
    HI_S32 s32Ret;
    /*The default value of the TDE2_MBOPT_S parameter is 0.*/
    TDE2_MBOPT_S stMbOpt = {0};
    TDE2_OPT_S stOpt = {0};
    /*The default values of the TDE2_SURFACE_S parameters are 0.*/
    TDE2_MB_S stMbSrc = {0};
    TDE2_SURFACE_S stDst = {0};
    TDE2_SURFACE_S stDisp = {0};
    TDE2_RECT_S stMbSrcRect = {0};
    TDE2_RECT_S stDstRect = {0};
    TDE2_RECT_S stDispRect = {0};
    TDE_HANDLE s32Handle;

    if( access( "/dev/hi_tde" ,F_OK) )
    {
        printf("Device file doesn't exist.\n");
        return HI_FAILURE;
    }

    s32Ret = HI_TDE2_Open();
    if(HI_SUCCESS != s32Ret)
    {
        printf("Fail to open TDE device.\n");
        return HI_FAILURE;
    }
    /*Set the parameters related to the MbOpt operation, and use the
    default values of the parameters related to the ROP.*/
```



```
stMbOpt.bDeflicker = HI_TRUE;//Anti-flicker is enabled.  
stMbOpt.enResize = TDE2_MBRESIZE_QUALITY_MIDDLE;//Medium-quality  
scaling is enabled.  
  
/*Set the parameters of a macroblock bitmap.*/  
stMbSrc.enMbFmt = TDE2_MB_COLOR_FMT_JPG_YCbCr420MBP;  
stMbSrc.u32YPhyAddr = g_u32YPhy;  
stMbSrc.u32CbCrPhyAddr = g_u32CbCrPhy;  
stMbSrc.u32YWidth = TDE_BMP_WIDTH;  
stMbSrc.u32YHeight = TDE_BMP_HEIGHT;  
stMbSrc.u32YStride = 720;//Assume that the Y stride of the decoded  
picture is 720.  
stMbSrc.u32CbCrStride = 720;//Assume that the CbCr stride of the  
decoded picture is 720.  
  
stMbSrcRect.s32Xpos = 0;  
stMbSrcRect.s32Ypos = 0;  
stMbSrcRect.u32Width = TDE_BMP_WIDTH;  
stMbSrcRect.u32Height = TDE_BMP_HEIGHT;  
  
/*Information about the source bitmap to be filled*/  
stDst.enColorFmt = TDE2_COLOR_FMT_YCbCr888;  
stDst.u32PhyAddr = g_u32SrcBmpPhy;//A memory space exists at this  
physical address.  
stDst.ul6Height = TDE_BMP_HEIGHT;  
stDst.ul6Width = TDE_BMP_WIDTH;  
stDst.ul6Stride = TDE_BMP_WIDTH * 2;  
  
stDstRect.s32Xpos = 0;  
stDstRect.s32Ypos = 0;  
/*Scale the size of a bitmap from 128 x 128 to 64 x 64.*/  
stDstRect.u32Width = TDE_BMP_WIDTH / 2;  
stDstRect.u32Height = TDE_BMP_HEIGHT / 2;  
  
/*Set display parameters.*/  
memcpy(&stDisp, &stDst, sizeof(TDE2_SURFACE_S));  
stDisp.u32PhyAddr = g_u32DispPhy;  
stDisp.enColorFmt = TDE2_COLOR_FMT_ARGB4444;//Convert the format into  
ARGB4444.  
memcpy(&stDispRect, &stDstRect, sizeof(TDE2_RECT_S));  
  
/*Create a TDE task.*/  
s32Handle = HI_TDE2_BeginJob();  
if(HI_ERR_TDE_INVALID_HANDLE == s32Handle)
```



```
{  
    printf("Fail to create Job.\n");  
    HI_TDE2_Close();  
    return HI_FAILURE;  
}  
  
/*Obtain a YCbCr888 raster bitmap by calling HI_TDE2_MbBlit.*/  
s32Ret = HI_TDE2_MbBlit(s32Handle, &stMbSrc, &stMbSrcRect, &stDst,  
    &stDstRect, &stMbOpt);  
if(HI_SUCCESS != s32Ret)  
{  
    printf("Fail to add quick copy to Job.\n");  
    HI_TDE2_CancelJob(s32Handle);/The task is canceled due to errors.  
    HI_TDE2_Close();  
    return HI_FAILURE;  
}  
  
/*Add the Bitblit operation to the TDE task, and convert the format  
into ARGB4444.*/  
s32Ret = HI_TDE2_Bitblit(s32Handle, HI_NULL, HI_NULL, &stDst,  
    &stDstRect, &stDisp, &stDispRect, &stOpt);  
if(HI_SUCCESS != s32Ret)  
{  
    printf("Fail to add blit to Job.\n");  
    HI_TDE2_CancelJob(s32Handle);/The task is canceled due to errors.  
    HI_TDE2_Close();  
    return HI_FAILURE;  
}  
  
/*Submit the TDE task in synchronous and non-block mode.*/  
s32Ret = HI_TDE2_EndJob(s32Handle, HI_FALSE, HI_FALSE, 0);  
if(HI_SUCCESS != s32Ret)  
{  
    printf("Fail to End Job.\n");  
    HI_TDE2_Close();  
    return HI_FAILURE;  
}  
  
/*Wait until the task is complete.*/  
s32Ret = HI_TDE2_WaitForDone(s32Handle);  
if (HI_ERR_TDE_QUERY_TIMEOUT == s32Ret)  
{  
    printf("Wait For Done Time Out.\n");  
    HI_TDE2_Close();
```



```
        return HI_FAILURE;
    }

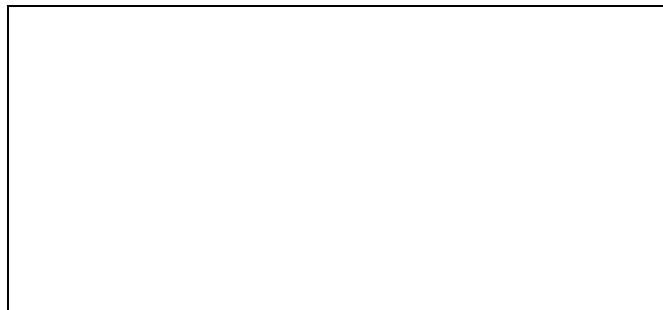
    /*The TDE completes all submitted tasks even a timeout occurs.*/
    HI_TDE2_Close();
    return HI_SUCCESS;
}
```

### 9.4.3.3 Setting Three Source Bitmaps

#### Scenario

During the bitmap mask ROP (performed by calling HI\_TDE2\_BitmapMaskRop) or bitmap mask blending operation (performed by calling HI\_TDE2\_BitmapMaskBlend), the information about the three source bitmaps, including the foreground bitmap, background bitmap, and mask bitmap, needs to be set. The type matrix can be processed during the mask ROP. [Figure 9-1](#) to [Figure 9-5](#) illustrate the effects.

**Figure 9-2** Background bitmap (corresponding to stSrc1 described in the example)



**Figure 9-3** Foreground bitmap (corresponding to stSrc2 described in the example)





**Figure 9-4** Mask bitmap (corresponding to stMask in the format of A1 described in the example)



**Figure 9-5** Output bitmap (corresponding to stDst described in the example)



## Working Process

To set three source bitmaps, perform the following steps:

- Step 1** Start and initialize a TDE device by calling HI\_TDE2\_Open.
- Step 2** Create a TDE task by calling HI\_TDE2\_BeginJob.
- Step 3** Set the sizes of the three bitmaps and operating areas, and add the bitmap mask ROP to the TDE task by calling HI\_TDE2\_BitmapMaskRop. The method of adding the bitmap mask blending operation is the same as the method of adding the bitmap mask ROP.
- Step 4** Submit the task in asynchronous and block mode by calling HI\_TDE2\_EndJob.
- Step 5** Stop the TDE device by calling HI\_TDE2\_Close.

----End

[Reference Code]

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "hi_tde_api.h"

#define TDE_BMP_WIDTH 128
#define TDE_BMP_HEIGHT 128
```



```
HI_S32 TDE_Sample3()
{
    HI_S32 s32Ret;
    /*The default values of the TDE2_SURFACE_S parameters are 0.*/
    TDE2_SURFACE_S stSrc1 = {0};
    TDE2_SURFACE_S stSrc2 = {0};
    TDE2_SURFACE_S stDst = {0};
    TDE2_SURFACE_S stMask = {0};
    TDE2_RECT_S stSrcRect1 = {0};
    TDE2_RECT_S stSrcRect2 = {0};
    TDE2_RECT_S stDstRect = {0};
    TDE2_RECT_S stMaskRect = {0};
    TDE_HANDLE s32Handle;

    if( access( "/dev/hi_tde" ,F_OK) )
    {
        printf("Device file doesn't exist.\n");
        return HI_FAILURE;
    }

    s32Ret = HI_TDE2_Open();
    if(HI_SUCCESS != s32Ret)
    {
        printf("Fail to open TDE device.\n");
        return HI_FAILURE;
    }

    /*Information about the source bitmap 1 to be filled*/
    stSrc1.enColorFmt = TDE2_COLOR_FMT_ARGB4444;
    stSrc1.u32PhyAddr = g_u32SrcBmpPhy1;//There is picture data at this
    physical address.

    stSrc1.ul6Height = TDE_BMP_HEIGHT;
    stSrc1.ul6Width = TDE_BMP_WIDTH;
    stSrc1.ul6Stride = TDE_BMP_WIDTH * 2;

    stSrcRect1.s32Xpos = 0;
    stSrcRect1.s32Ypos = 0;
    stSrcRect1.u32Width = TDE_BMP_WIDTH;
    stSrcRect1.u32Height = TDE_BMP_HEIGHT;

    /*The information about source bitmap 2 and source bitmap 1 is the
    same.*/
    memcpy(&stSrc2, &stSrc1, sizeof(TDE2_SURFACE_S));
    stSrc2.u32PhyAddr = g_u32SrcBmpPhy2;//There is picture data at this
```



```
physical address.

memcpy(&stSrcRect2, &stSrcRect1, sizeof(TDE2_RECT_S));

/*The width, height, and format of the target bitmap are the same as
those of the source bitmap.*/
memcpy(&stDst, &stSrc1, sizeof(TDE2_SURFACE_S));
stDst.pu8PhyAddr = g_u32DstBmpPhy;//A memory space exists at this
physical address.

memcpy(&stDstRect, &stSrcRect1, sizeof(TDE2_RECT_S));

/*Information about the mask bitmap to be filled.*/
stMask.enColorFmt = TDE2_COLOR_FMT_A1;
stMask.u32PhyAddr = g_u32MaskBmpPhy;//There is picture data at this
physical address.
stMask.ul6Height = TDE_BMP_HEIGHT;
stMask.ul6Width = TDE_BMP_WIDTH;
stMask.ul6Stride = TDE_BMP_WIDTH / 8;//Assume that the stride of the
A1 bitmap is TDE_BMP_WIDTH.

/ 8

/*The operating areas of the mask bitmap and source bitmap are the
same.*/
memcpy(&stMaskRect, &stSrcRect1, sizeof(TDE2_RECT_S));

/*Create a TDE task.*/
s32Handle = HI_TDE2_BeginJob();
if(HI_ERR_TDE_INVALID_HANDLE == s32Handle)
{
    printf("Fail to create Job.\n");
    HI_TDE2_Close();
    return HI_FAILURE;
}

/*Perform operations on the bitmaps stSrc and stDst, and overwrite the
result stored at the address of stDst with the operation result.*/
s32Ret = HI_TDE2_BitmapMaskRop(s32Handle, &stSrc1, &stSrcRect1, tSrc2,
                                &stSrcRect2, &stMask, &stMaskRect,
                                &stDst, &stDstRect, TDE2_ROP_COPYOPEN,
                                TDE2_ROP_COPYOPEN);
if(HI_SUCCESS != s32Ret)
{
    printf("Fail to add blit to Job.\n");
    HI_TDE2_CancelJob(s32Handle);//The task is canceled due to errors.
    HI_TDE2_Close();
```



```
        return HI_FAILURE;
    }

    /*Submit the TDE task in asynchronous and block mode.*/
    s32Ret = HI_TDE2_EndJob(s32Handle, HI_FALSE, HI_TRUE, 100);
    if(HI_SUCCESS != s32Ret)
    {
        printf("Fail to End Job.\n");
        HI_TDE2_Close();
        return HI_FAILURE;
    }

    /*The TDE completes all submitted tasks even a timeout occurs.*/
    /*Ensure that the task is complete.*/
    s32Ret = HI_TDE2_WaitForDone(s32Handle);
    if (HI_ERR_TDE_QUERY_TIMEOUT == s32Ret)
    {
        printf("Wait For Done Time Out.\n");
        HI_TDE2_Close();
        return HI_FAILURE;
    }

    HI_TDE2_Close();
    return HI_SUCCESS;
}
```



# Contents

---

<b>10 HiGo .....</b>	<b>1</b>
10.1 Overview .....	1
10.2 Important Concepts .....	2
10.2.1 Gdev Module .....	2
10.2.2 Surface Module .....	3
10.2.3 Blit Module .....	4
10.2.4 Text Module .....	6
10.2.5 WM Module.....	7
10.3 Features .....	7
10.4 Development Guide.....	8
10.4.1 Configuring and Starting Graphics Layers.....	8
10.4.2 Using the MemSurface.....	10
10.4.3 Filling Rectangles .....	11
10.4.4 Transferring Bit Blocks.....	13
10.4.5 Decoding and Displaying Pictures .....	14
10.4.6 Outputting Texts.....	16
10.4.7 Changing Windows and rRefreshing Desktops.....	18
10.4.8 Using the Cursor Module .....	19
10.4.9 Encoding .....	20



# Figures

<b>Figure 10-1</b> Typical application architecture of the HiGo .....	1
<b>Figure 10-2</b> Double-buffer mode.....	2
<b>Figure 10-3</b> Triple-buffer mode.....	2
<b>Figure 10-4</b> Colorkey operation performed on the source bitmap (operation 1) .....	5
<b>Figure 10-5</b> Colorkey operation performed on the target bitmap (operation 1) .....	6
<b>Figure 10-6</b> Colorkey operation performed on the source bitmap (operation 2) .....	6
<b>Figure 10-7</b> Colorkey operation performed on the target bitmap (operation 2) .....	6
<b>Figure 10-8</b> Process for configuring and starting a graphics layer .....	9
<b>Figure 10-9</b> Process for using a MemSurface.....	10
<b>Figure 10-10</b> Process for filling in a rectangle .....	12
<b>Figure 10-11</b> Process for transferring a bit block .....	13
<b>Figure 10-12</b> Process for decoding and displaying pictures .....	15
<b>Figure 10-13</b> Process for outputting texts.....	17
<b>Figure 10-14</b> Process for changing windows and refreshing desktops.....	18
<b>Figure 10-15</b> Process for using the cursor module .....	20
<b>Figure 10-16</b> Encoding process .....	21



## Tables

---

**Table 10-1** Buffers used in the four refresh modes..... 3



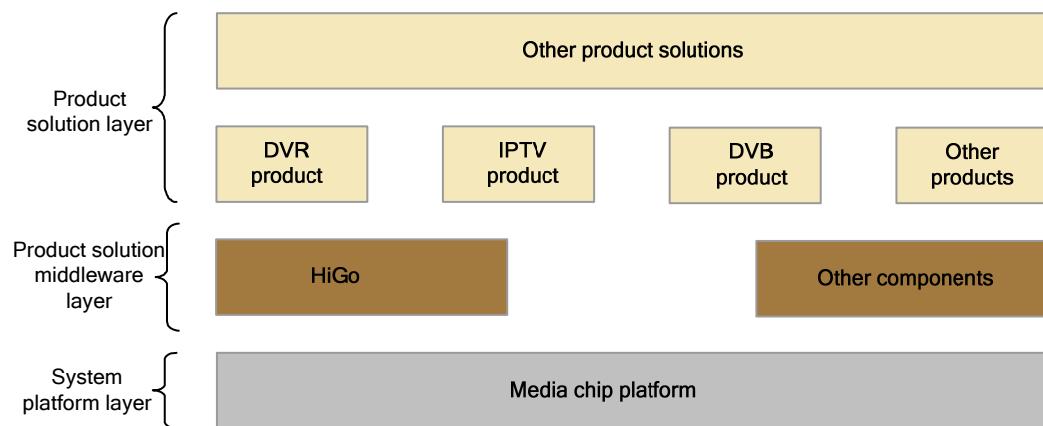
# 10 HiGo

## 10.1 Overview

The HiGo is a highly efficient graphics component provided by HiSilicon. It makes full use of the acceleration function of chips and provides you with the basic functions required for graphical user interface (GUI) development. At present, the HiGo applies to media chips of HiSilicon.

[Figure 10-1](#) shows the typical application architecture of the HiGo.

**Figure 10-1** Typical application architecture of the HiGo



The software architecture consists of the following three layers:

- System platform layer  
This layer is the bottom layer. It consists of the chip and media processing platform software. The media processing platform provides unified APIs for various chips.
- Product solution middleware layer  
This layer provides components based on product forms. Other products contain some other components.
- Product solution layer  
This layer combines GUI services by using the HiGo component to integrate product solutions, such as the DVR, IPTV, and DVB.



## 10.2 Important Concepts

### 10.2.1 Gdev Module

[Graphics layer refresh]

After drawing at a graphics layer, the drawing result is immediately displayed on the screen if the single-buffer refresh mode is used. Otherwise, the drawing result is not displayed on the screen until you refresh the graphics layer.

[Graphics layer refresh mode]

The refresh modes of the graphics layer are described as follows:

- Single-buffer mode

A buffer is used for both drawing and output. The drawn graphics can be rapidly displayed on the screen, but the drawing process is visible to users.

- Double-buffer mode

The drawn graphics are stored in a buffer. When the graphics layer is refreshed, the graphics is synchronized to the display buffer. In this way, the intermediate graphics are invisible to users, but cracks appear on some parts of the graphics.

For details, see [Figure 10-2](#).

**Figure 10-2** Double-buffer mode



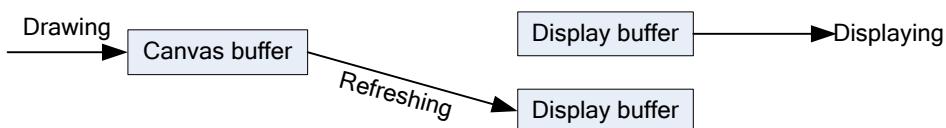
- Triple-buffer mode

The triple-buffer mode is an old flip mode. With high refresh rate, this mode eliminates cracks on graphics and therefore is ideal for animation users who have high requirements on the refresh rate. However, you must refresh the display buffer completely each time you refresh the graphics layer in this mode, which consumes more memory. For details, see [Figure 10-3](#).

- Over mode (triple buffers)

The over mode is similar to the triple-buffer mode. The only difference is that in over mode, the previous frame is overwritten by the next frame if the previous frame is not refreshed in time. For details, see [Figure 10-3](#).

**Figure 10-3** Triple-buffer mode



The difference among the four refresh modes lies in the number of canvas buffers and display buffers. The canvas buffer inside the HiGo is used by default. The size of the canvas buffer is specified by users when the graphics layer is created. To be specific, **CanvasWidth** and **CanvasHeight** in the HIGO\_LAYER\_INFO\_S structure indicate the width and height of the



canvas buffer. Users can call HI\_GO\_SetLayerSurface to specify a user-defined surface as the canvas buffer.

You can set the sizes of the canvas buffer and display buffer by calling the corresponding interfaces. In this way, memory can be allocated as required. In the double-buffer mode, triple-buffer mode, and over mode, two-level scaling is supported.

- Level 1 scaling: When the sizes of the canvas buffer and display buffer are inconsistent, the canvas buffer is scaled to fit into the display buffer by using the TDE.
- Level 2 scaling: When the sizes of the display buffer and screen (the screen size can be customized) are inconsistent, the display buffer is scaled to fit into the screen by using the display hardware. The level 2 scaling is supported only when the hardware configuration meets requirements.

Level 2 scaling (VO scaling) is recommended to ensure the consistency between the canvas and display buffers.

#### NOTE

You can use only one refresh mode at a time.

The digits in [Table 10-1](#) indicate the number of required buffers. The actual size of the required buffer is calculated as follows: displaywidth x displayheight x bpp x num, where **displaywidth** is the width of the graphics layer; **displayheight** is the height of the graphics layer; **bpp** is the number of bytes for a pixel; and **num** is the number of buffers that is obtained by querying [Table 10-1](#).

**Table 10-1** Buffers used in the four refresh modes

Mode	Number of Canvas Buffers	Number of Display Buffers
Single-buffer mode	1	0
Double-buffer mode	1	1
Triple-buffer mode	1	2
Over mode	1	2

[Multiple graphics layers]

Each display device supports multiple graphics layers. The number of supported graphics layers depends on the chip. The HiGo only manages the graphics layers.

#### NOTE

Hi3798C V100/Hi3796C V100/Hi3798C V200 supports four HD layers and one SD layer.

## 10.2.2 Surface Module

[Surface]

A surface is a consecutive area in the memory that stores the attributes of graphic data such as height, width, pixel format, colorkey, and alpha. The surface is divided into the following two types based on the type of memory it uses:

- Media memory zone (MMZ) surface



The MMZ surface is a physically consecutive area in the memory and can be identified by the hardware and all components of the HiGo. The MMZ is the basis for establishing associations between components.

- Operating system (OS) surface (system memory)

The OS surface is a logically consecutive memory area (it may be not physically continuous), such as the memory allocated by calling the malloc function. The OS surface can be identified by some components of the HiGo, but not the hardware. When the MMZ surface is insufficient, the OS surface can be used.

#### NOTE

Note the following when using the OS surface:

- The OS surface can be used during the .bmp, .gif, or .png software decoding when the picture is not scaled and the picture format is not converted. The usage is the same as that during hardware decoding.
- The RGB data stored in the surface can be encoded as a .bmp file.
- Texts can be output to the OS surface.
- The conversion between the MMZ surface and OS surface can be implemented by calling HI\_GO\_Blit. The HI\_GO\_Blit function allows you to set the **HIGO\_BLTOPT\_S** variable to 0 and perform the colorkey-related operations and blit operations on the RGB data stored on the OS surface and MMZ surface. Other operations are not supported.
- You can obtain the physical address (pData[0].pPhyData) and virtual address (pData[0].pData) of the OS surface by calling HI\_GO\_LockSurface(Surface, pData, HI\_TRUE).

The surface is divided into two types by source:

- Surfaces created by the HiGo. For example, during decoding, the Dec module places the decoded data on a surface for users. At present, the Dec module, Gdev module, and WM module provide surfaces.
- Memory surfaces created by users. Users can create a memory surface and temporarily store graphic data on the surface.

The surface can also be divided into two types by function:

- LayerSurface

The data stored on this surface can be directly output to the display device.

- MemSurface

This surface allows you to temporarily store graphic data.

### 10.2.3 Blit Module

#### [Rectangle filling]

Rectangle filling is the process of filling a rectangle on the surface with a color. Alpha blending is supported during rectangle filling.

#### [Bit block transfer]

Bit block transfer is the process of transferring bitmap data from one surface to another. During bit block transfer, color space conversion, pixel format conversion, alpha blending, colorkey, ROP attribute, scaling, rotation, and mirroring can be implemented.

#### [Color space conversion]

Color space conversion is the process of converting bitmap data from one color space to another. At present, the Blit module only supports the conversion from YUV to RGB.

#### [Pixel format conversion]



The pixel format includes the color depth (number of binary bits of each pixel), palette, or combination of RGBA components (red, green, blue, and alpha components) of pixels. Pixel format conversion is the process of converting the pixel format of the bitmap data into another pixel format. The Blit module supports the conversion of the following pixel formats:

- Pixel formats of source bitmaps: 8-bit palette, RGB444, ARGB4444, RGB555, RGB565, ARGB1555, RGB888, or ARGB8888
- Pixel format of the target bitmap: RGB444, ARGB4444, RGB555, RGB565, ARGB1555, RGB888, or ARGB8888.

#### [Alpha blending]

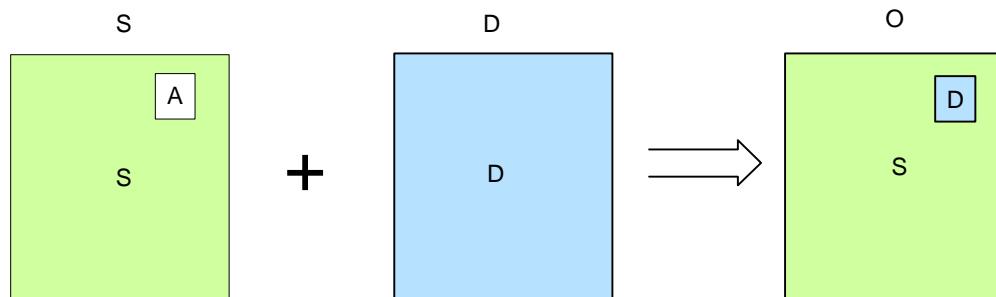
Alpha blending is the process of obtaining a bitmap with blended alpha value by calculating the weighted sum of the pixel values of multiple bitmaps based on alpha values. The alpha value is from the source bitmap, target bitmap, or global alpha value.

#### [Colorkey]

The colorkey operations can be divided into two types: operation 1 (the Blit module does not perform data operation) and operation 2 (the Blit module performs data operation), as shown in [Figure 10-4](#) to [Figure 10-7](#). The following describes the colorkey operations performed on the source bitmap and target bitmap:

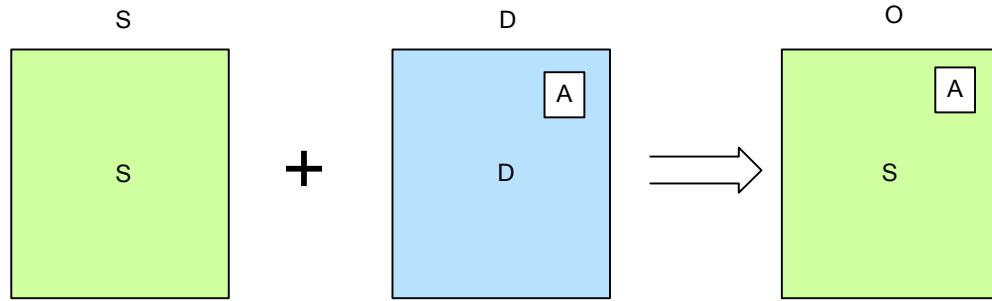
- When the colorkey operation is performed on the source bitmap, the bitmap is transferred from the source address to the output address. During this process, the pixel of the bitmap with colorkey is replaced with that of the target bitmap.
- When the colorkey operation is performed on the target bitmap, the bitmap is transferred from the target address to the output address. During this process, the pixel of the bitmap with the color that is different from the colorkey is replaced with that of the source bitmap.

**Figure 10-4** Colorkey operation performed on the source bitmap (operation 1)

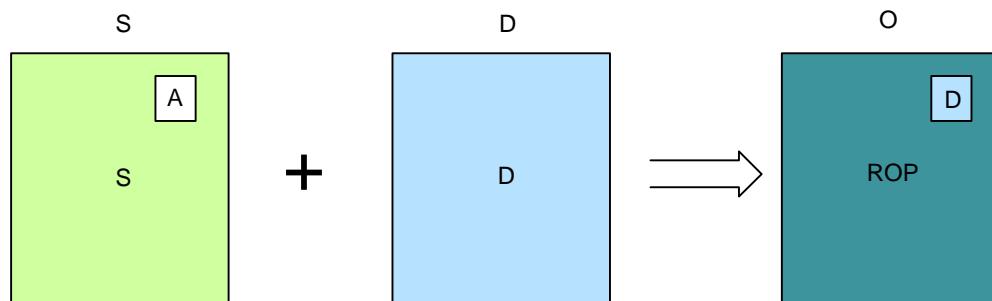




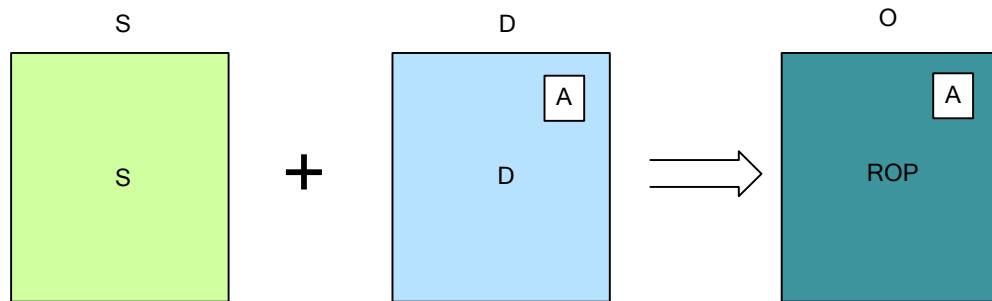
**Figure 10-5** Colorkey operation performed on the target bitmap (operation 1)



**Figure 10-6** Colorkey operation performed on the source bitmap (operation 2)



**Figure 10-7** Colorkey operation performed on the target bitmap (operation 2)



[ROP]

ROP is the process of calculating the color values of the source bitmap and target bitmap by using the Boolean operation.

## 10.2.4 Text Module

[Single-byte character set]

Single-byte character sets are those whose characters are encoded as single bytes, including the American Standard Code for Information Interchange (ASCII) character set.

[Multi-byte character set]



Multi-byte character sets are those whose characters are encoded as multiple bytes with variable or constant length, including the GB series and UTF-8.

## 10.2.5 WM Module

[Desktop]

A desktop is a display screen that is visible to users. The drawing results of all applications are displayed through a desktop. Desktop management indicates control over desktops, for example, establishing associations with windows and graphics layers.

[Window]

A window is a rectangle area on the screen and is a part of the desktop. Windows can overlap each other or be independent of each other. Window management indicates control over windows, for example, setting the window opacity, window location, and window size, and cutting window regions. Windows are dependent on desktops, that is, each window belongs to a desktop.

[Opacity]

Opacity indicates the visibility of the overlapping area of an upper-layer window to a lower-layer window when windows overlap.

[Z-order]

Z-order indicates the sequence of overlapping windows. Z-axis is perpendicular to the screen and points from the inside to the outside of the screen. The Z-order value increases from 0.



### NOTE

- Desktops and graphics layers are one-to-one mapped. The HiGo supports both multiple graphics layers and multiple desktop systems.
- Windows are the managed objects on which graphics are drawn, and windows are drawn by operating the surfaces of multiple windows. Therefore, a surface is required for drawing a window.

## 10.3 Features

As the graphics interface of HiSilicon chips, the HiGo applies to all media chips of HiSilicon. In addition, the HiGo makes full use of the acceleration function of the hardware to achieve high efficiency. The HiGo is divided into the following nine modules by function:

- Gdev module

The Gdev module manages multiple graphics layers of a graphic device and supports the display of the drawing results for users. In addition, the Gdev module supports operations between graphics layers, including advanced special effects such as alpha and colorkey.

- Surface module

The surface module manages the surface, for example, creating a surface, and setting and querying the surface attributes.

- Blit module

The Blit module transfers bit blocks. During block transfer, the operations, such as the alpha operation, colorkey operation, ROP, and CSC, are supported. The Blit module also supports special effect operations such as mirroring and rotation.

- Decoder (Dec) module



The Dec module decodes various pictures. The supported image formats include BMP, JPEG, GIF, and PNG.

- Text module

The text module allows you to enter strings on the GUI. At present, matrix and vector Truetype fonts are supported.

- WM module

The WM module supports multi-window overlapping and inter-window operations such as Alpha and Colorkey.

- Cursor module

The cursor module allows you to show or hide the cursor, set the position of the cursor, and set the cursor picture.

- Encoder (Enc) module

The Enc module allows you to encode the BMP or JPEG pictures on the surface.

- Proc log module

For details about the APIs for each module, see the *HMS API Development Reference*.

## 10.4 Development Guide

The HiGo module is used in the following scenarios:

- Configuring and starting graphics layers
- Using the MemSurface
- Filling rectangles
- Transferring bit blocks
- Decoding and displaying pictures
- Outputting texts
- Changing windows and refreshing desktops
- Using the cursor module
- Encoding

### 10.4.1 Configuring and Starting Graphics Layers

#### Scenario

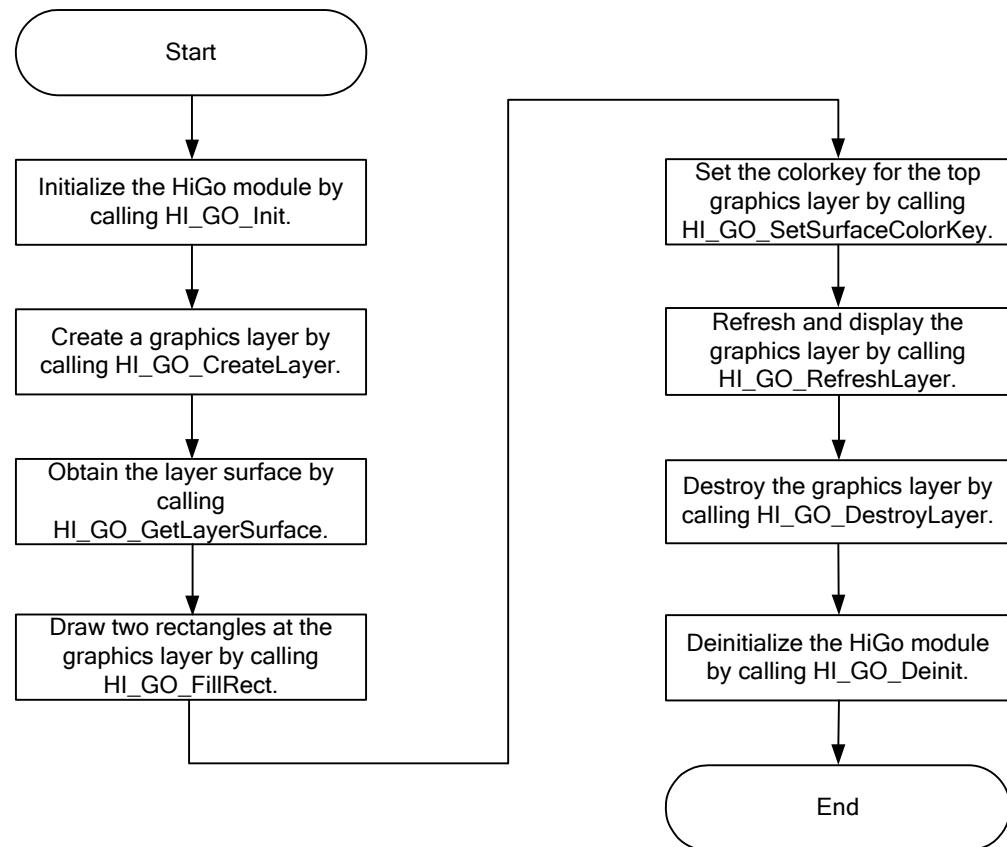
The Gdev module manages multiple graphics layers of the same graphic device and supports displaying the drawing results on display devices for users. In addition, the Gdev module supports operations between graphics layers, including advanced special effects such as alpha and colorkey.

#### Working Process

Figure 10-8 shows the process for configuring and starting a graphics layer.



**Figure 10-8** Process for configuring and starting a graphics layer



## Programming Guide

To configure and start a graphics layer, perform the following steps:

- Step 1** Initialize the HiGo and prepare the resources required by the system.
- Step 2** Create a graphics layer.
- Step 3** Obtain the surface of the graphics layer.
- Step 4** Draw two rectangles at the graphics layer with the colorkey value as **0xFF0000** and **0x00FF00** respectively.
- Step 5** Set the colorkey of the top graphics layer to **0xFF0000**.
- Step 6** Refresh the graphics layer. Only one rectangle is displayed.
- Step 7** Destroy the graphics layer.
- Step 8** Deinitialize the HiGo.

----End



## Notes

At present, if there are multiple layers, the HiGo can manage only the graphics layers, but not the video layer, cursor layer, and background layer. The difference between a layer and a surface (excluding the layer surface) is that a surface is just a temporary drawing area and the drawn result must be transferred to a layer (layer surface) for display.

## Sample

See `sample_gdev.c`.

### 10.4.2 Using the MemSurface

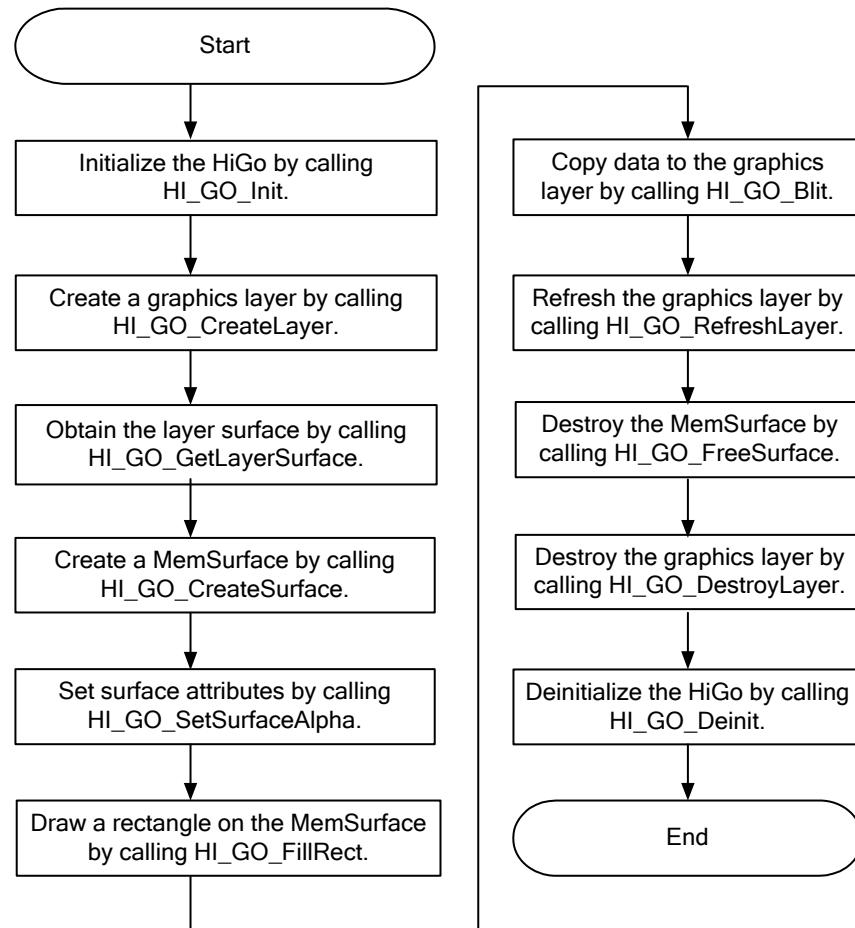
#### Scenario

The surface module provides you with an interface to perform surface operations, for example, creating a surface, and setting and querying the surface attributes.

#### Working Process

[Figure 10-9](#) shows the process for using the MemSurface.

**Figure 10-9** Process for using a MemSurface





## Programming Guide

To use a MemSurface, perform the following steps:

- Step 1** Initialize the HiGo.
- Step 2** Create a graphics layer.
- Step 3** Obtain the surface of the graphics layer.
- Step 4** Create a MemSurface and save it in the memory.
- Step 5** Set the alpha attribute of the MemSurface. This enables the translucency effect during the Blit operations.
- Step 6** Draw a rectangle on the MemSurface.
- Step 7** Transfer the data on the MemSurface to the surface of the graphics layer.
- Step 8** Refresh the graphics layer to display the rectangle on a display terminal.
- Step 9** Destroy the MemSurface.
- Step 10** Destroy the graphics layer.
- Step 11** Deinitialize the HiGo.

----End

## Notes

None

## Sample

See `sample_surface.c`.

### 10.4.3 Filling Rectangles

#### Scenario

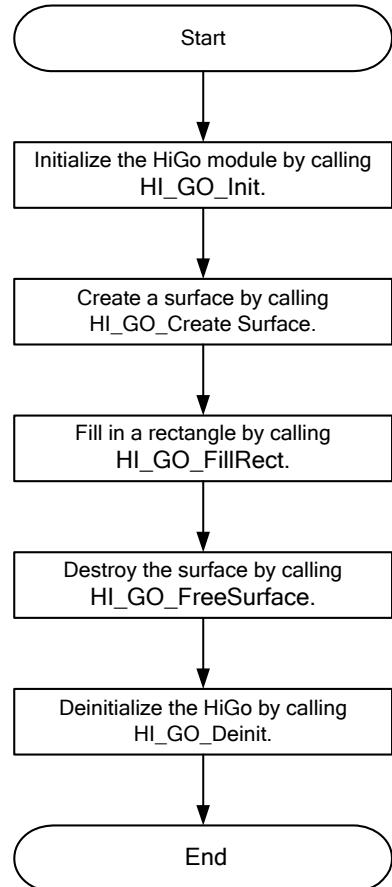
The Blit module is used to fill in rectangles and transfer bit blocks. During rectangle filling, alpha blending is supported. During bit block transfer, CSC, pixel format conversion, alpha blending, global alpha operation, porter duff operation, colorkey operation, ROP attribute, scaling, rotation, and mirroring are supported.

#### Working Process

[Figure 10-10](#) shows the process for filling a rectangle.



**Figure 10-10** Process for filling in a rectangle



## Programming Guide

To fill in a rectangle, perform the following steps:

- Step 1** Initialize the HiGo.
- Step 2** Create a MemSurface and obtain its surface handle.
- Step 3** Fill in a rectangle on the surface with a specified color. You need to specify the surface, rectangle area, filling color, and alpha blending mode.
- Step 4** Destroy the surface.
- Step 5** Deinitialize the HiGo.

**----End**

## Notes

The surface on which a rectangle is filled in can be a MemSurface or a LayerSurface. If the surface is a LayerSurface, you need to obtain the surface handle of the graphics layer after creating it.



## Sample

See `sample_fillrect.c`.

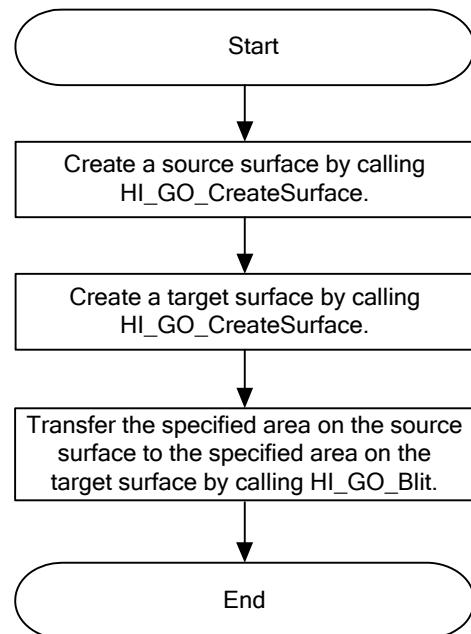
### 10.4.4 Transferring Bit Blocks

#### Scenario

#### Working Process

Figure 10-11 shows the process for transferring a bit block.

**Figure 10-11** Process for transferring a bit block



#### Programming Guide

To transfer a bit block, perform the following steps:

- Step 1** Create a source surface. Then you can obtain its handle.
- Step 2** Create a target surface. Then you can obtain its handle.
- Step 3** Transfer a bit block. You need to specify the transfer areas on the source surface, target area, and transfer attributes before transferring a bit block.

----End

#### Notes

Note the following when transferring bit blocks:

- Only the CSC from YUV to RGB is supported.



- The pixel format of the target bitmap can be RGB444, ARGB4444, RGB555, RGB565, ARGB1555, RGB888, or ARGB8888.
- Alpha blending and ROP cannot be performed at the same time.
- Scaling, rotation, and mirroring cannot be performed at the same time.
- Scaling, rotation, and mirroring cannot be performed during alpha blending.
- Scaling, rotation, and mirroring cannot be performed during the ROP.
- Scaling, rotation, and mirroring cannot be performed during the colorkey operation.
- Scaling, rotation, and mirroring are applicable only when both the source bitmap and target bitmap are in 16-bit or 32-bit format.

## Sample

See `sample_fillrect.c`.

## 10.4.5 Decoding and Displaying Pictures

### Scenario

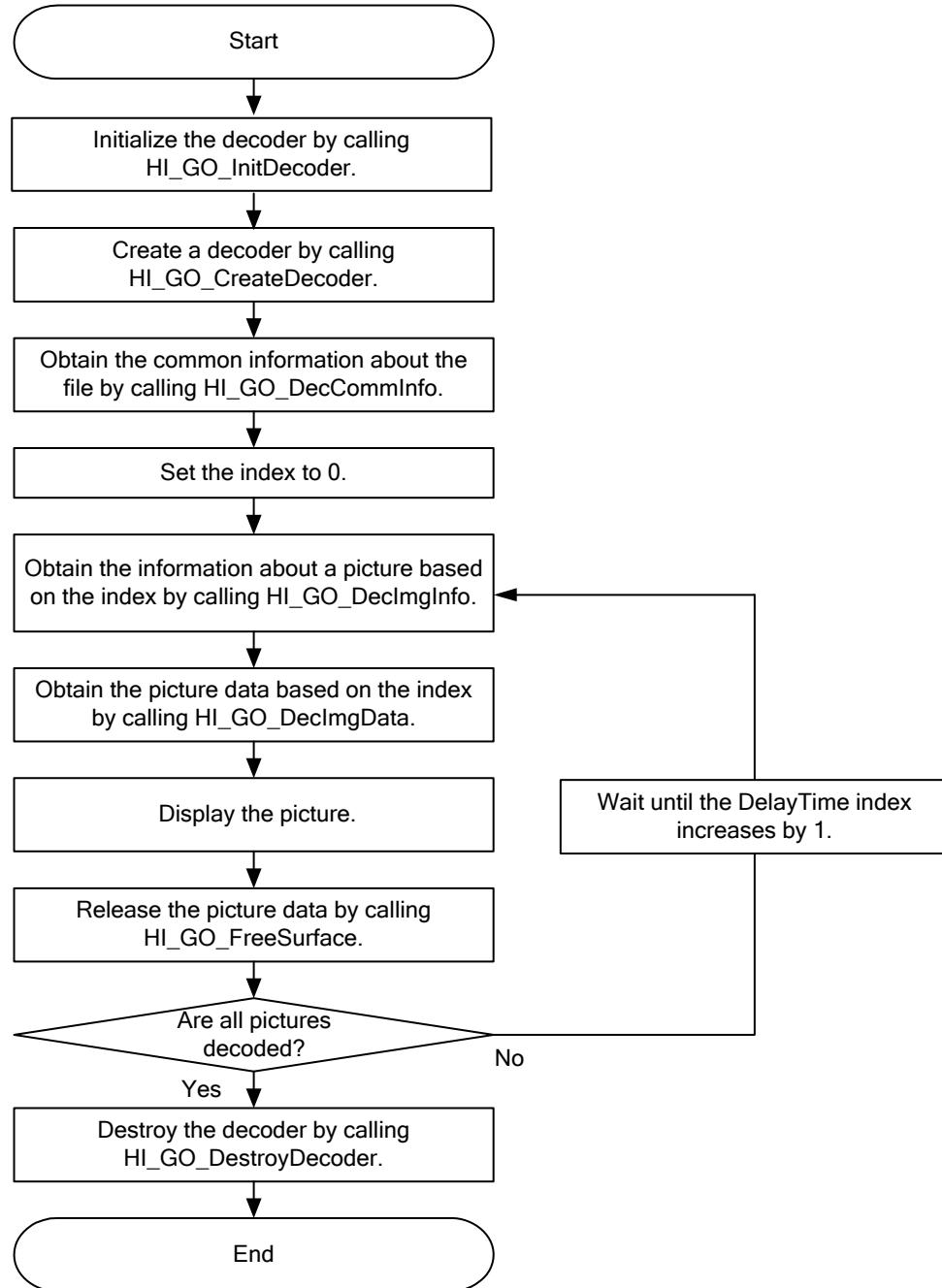
The Dec module decodes pictures and outputs the decoded pictures to the MemSurface. This module can decode the pictures in four formats: JPEG (baseline/progressive), GIF (89a/87a), BMP (V3), and PNG. In addition, this module supports three input modes: file, memory, and stream.

### Working Process

[Figure 10-12](#) shows the process for decoding and displaying pictures.



**Figure 10-12** Process for decoding and displaying pictures



## Programming Guide

To decode and display pictures, perform the following steps:

**Step 1** Initialize the decoder.

**Step 2** Create a decoder. You need to specify the data source of the decoder. If the data source is memory, specify the memory address; if the data source is a file, specify the file path. You also need to obtain the handle of the created decoder.



- Step 3** Obtain the common data information about the file and perform steps 4 to 7 on each picture in the file. The information includes the number of pictures in the file, picture file type, and width, height, and background color of the source pictures.
- Step 4** Obtain the information about a picture. The information includes the X offset and Y offset in relationship to the screen, width and height of the picture, channel alpha, colorkey, pixel format, and interval between the picture and the next picture.
- Step 5** Obtain the data of the picture. You can obtain the handle of the surface that stores the picture data.
- Step 6** Display the picture.
- Step 7** Release the surface that stores the picture data.
- Step 8** Continue to perform decoding until all pictures are decoded.
- Step 9** Destroy the decoder.
- End

## Notes

None

## Sample

See `sample_dec2.c`.

## 10.4.6 Outputting Texts

### Scenario

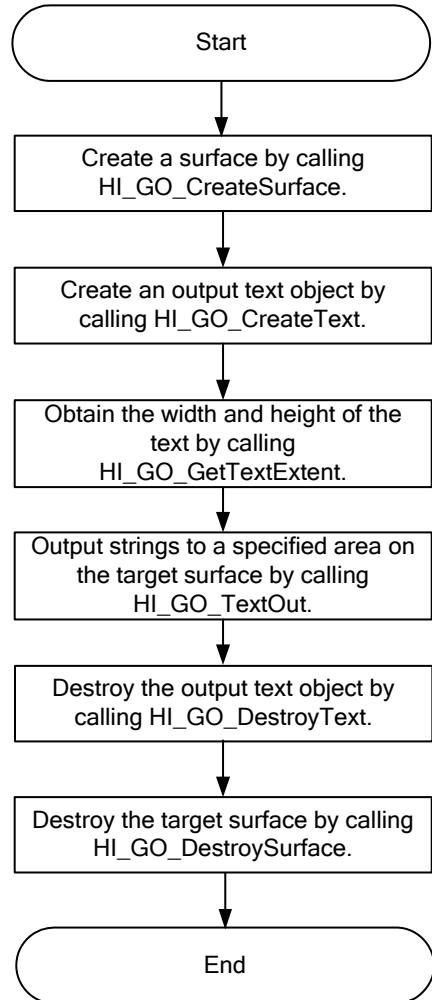
The text module enables the user to enter texts (typesetting is not included) in the specified rectangle on the specified surface. This module only supports the input of the UTF-8 and GB2312 character sets. Mixed output of multiple languages is supported, and the mixed output of English and another language is recommended.

### Working Process

[Figure 10-13](#) shows the process for outputting texts.



**Figure 10-13** Process for outputting texts



## Programming Guide

To output texts, perform the following steps:

- Step 1** Create a target surface.
- Step 2** Create an output text object. You need to specify the font file corresponding to the character set, create an output text object, and output the object handle. An object supports a single-byte character set and a multi-byte character set at the same time.
- Step 3** Obtain the width and height of the text.
- Step 4** Output strings to a specified area on the target surface. To be specific, you need to specify an output area on the target surface based on the width and height of the text, and then output the text to the specified area.
- Step 5** Destroy the output text object.
- Step 6** Destroy the target surface.

----End



## Notes

None

## Sample

See `sample_text.c`.

## 10.4.7 Changing Windows and Refreshing Desktops

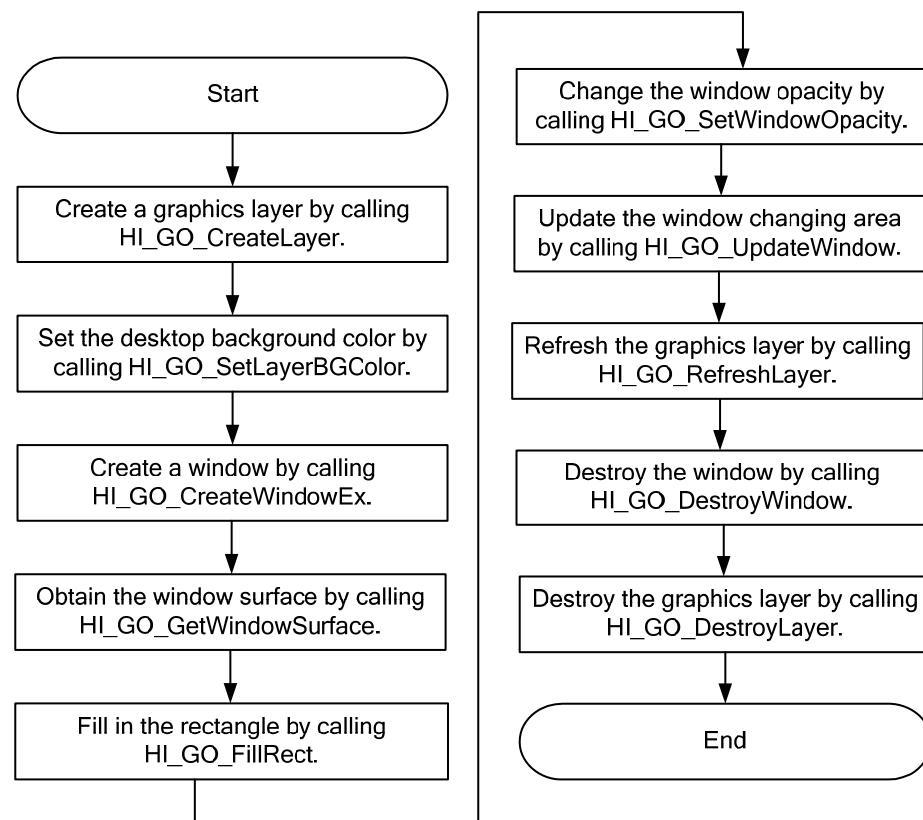
### Scenario

The WM module is a window manager that provides a convenient drawing management mechanism. The WM module allows you to create a desktop and multiple windows on the desktop. It facilitates management of multi-window overlapping. In addition, the WM module can be used to create a desktop at a graphics layer, create a window on the desktop, and set and obtain window attributes including the content, location, and opacity of the window.

### Working Process

[Figure 10-14](#) shows the process for changing windows and refreshing desktops.

**Figure 10-14** Process for changing windows and refreshing desktops





## Programming Guide

To change windows and refresh desktops, perform the following steps:

- Step 1** Create a graphics layer. Create a graphics layer for displaying the desktop and obtain the handle of the graphics layer.
- Step 2** Set the desktop background color. Specify the layer background color for a specified graphics layer.
- Step 3** Create a window. To be specific, create a window in a specified region on a specified desktop and obtain the window handle. This window belongs to the specified desktop.
- Step 4** Obtain the window surface, and obtain the surface handle based on the window handle.
- Step 5** Draw graphics (such as filling in a rectangle) on the window surface.
- Step 6** Change the window attributes, including the region, Z-order, and opacity of the window.
- Step 7** Update the window to notify the desktop of the modification.
- Step 8** Refresh the desktop. Contents on the desktop have changed and the desktop needs to be refreshed to display the updated contents.
- Step 9** Destroy the window. The memory consumed by the window is also released.
- Step 10** Destroy the graphics layer.

----End

## Notes

None

## Sample

See `sample_wm.c`.

## 10.4.8 Using the Cursor Module

### Scenario

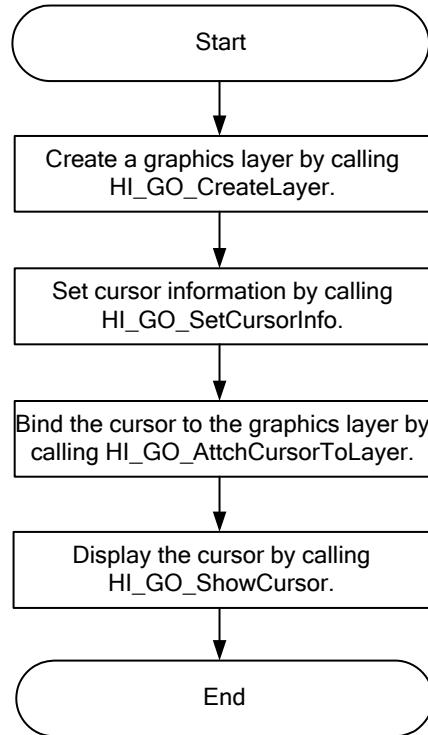
The cursor module allows you to show or hide the cursor, set the cursor position and cursor picture, and bind the cursor to a graphics layer.

### Working Process

[Figure 10-15](#) shows the process for using the cursor module.



**Figure 10-15** Process for using the cursor module



## Programming Guide

To use the cursor module, perform the following steps:

- Step 1** Create a graphics layer.
- Step 2** Set the cursor information, including the surface of the cursor picture and cursor hotspot.
- Step 3** Bind the cursor to the graphics layer. The cursor can be displayed at the graphics layer only after being bound to the graphics layer.
- Step 4** Display the cursor.

----End

## Notes

None

## Sample

See `sample_cursor.c`.

## 10.4.9 Encoding

### Scenario

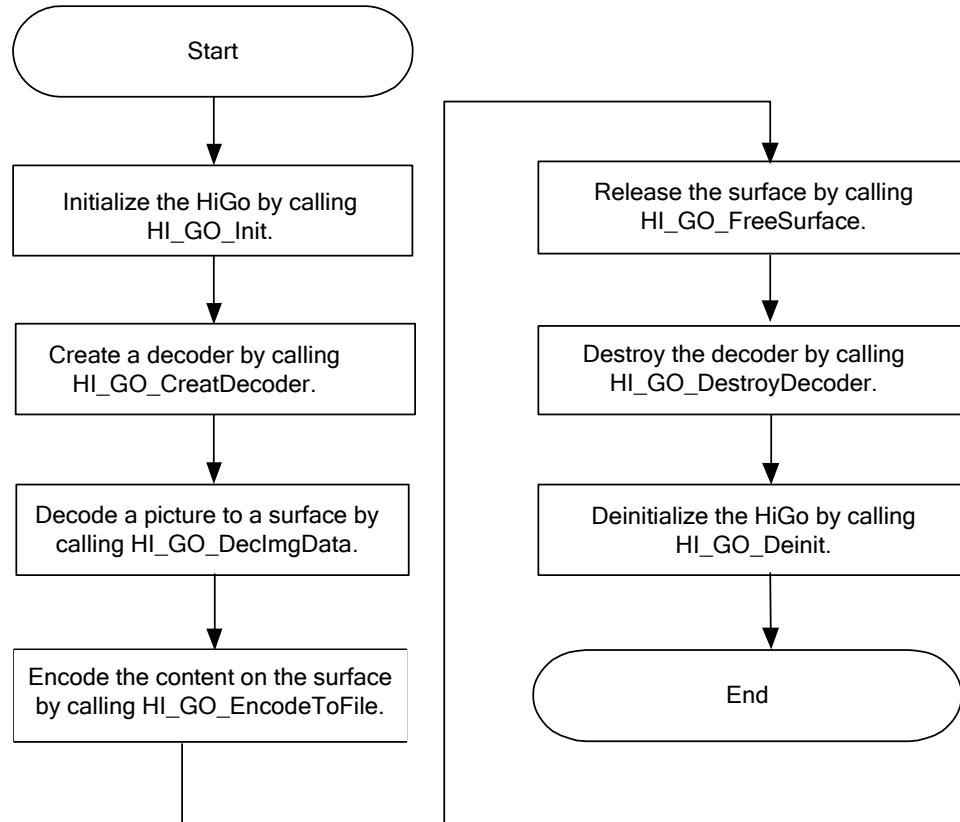
The Enc module supports BMP and JPEG encoding on a surface.



## Working Process

Figure 10-16 shows an encoding process.

**Figure 10-16** Encoding process



## Programming Guide

To perform encoding, perform the following steps:

- Step 1** Initialize the HiGo and prepare the resources required by the system.
- Step 2** Create a decoder.
- Step 3** Decode a picture to a surface.
- Step 4** Encode the content on the surface to a specified file.
- Step 5** Release the surface.
- Step 6** Destroy the decoder.
- Step 7** Deinitialize the HiGo.

----End



## Notes

None

## Sample

See `sample_enc.c`.



# Contents

---

<b>11 Tuner .....</b>	<b>4</b>
11.1 Overview .....	4
11.2 Important Concepts .....	5
11.3 Features .....	5
11.4 Development Guide.....	6
11.4.1 Working Process.....	6
11.4.2 Initializing and Deinitializing the Tuner Module .....	7
11.4.3 Configuring Front-End Devices .....	9
11.4.4 Locking Frequencies and Obtaining Signal Information .....	18
11.4.5 Blind Scanning .....	20
11.4.6 Sample.....	21
11.5 FAQs.....	22
11.5.1 Debugging information .....	22
11.5.2 Common Issues .....	23



# Figures

<b>Figure 11-1</b> Application architecture of the tuner module .....	4
<b>Figure 11-2</b> Overall working process .....	6
<b>Figure 11-3</b> Initialization and deinitialization process.....	7
<b>Figure 11-4</b> Process for configuring the LNB .....	10
<b>Figure 11-5</b> Process for controlling switches .....	12
<b>Figure 11-6</b> Typical switch cascading scenario .....	14
<b>Figure 11-7</b> Process for controlling antennas .....	17
<b>Figure 11-8</b> Process for locking frequencies and performing subsequent operations.....	19



## Tables

<b>Table 11-1</b> Tuner debugging information .....	22
---	----



# 11 Tuner

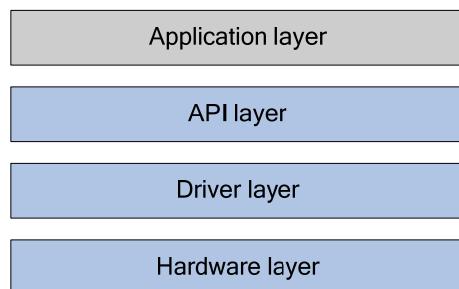
## 11.1 Overview

In the digital TV STB, the tuner tunes the digital TV signals, and the Demod demodulates the signals. The tuner and the Demod work with the DEMUX and the decoder to receive and decode the digital TV signals. The tuner module described in this document is a generalized concept, including the drivers of the tuner and Demod. For the digital video broadcasting satellite (DVB-S)/S2 application, the low noise block (LNB) control driver, various switches, digital satellite equipment control (DiSEqC) are also supported. The tuner module provides a set of APIs for upper-layer software.

Currently the tuner module supports the J.83-A/B/C, DVB-S/S2, DVB-T/T2, and ISDB-T standards.

[Figure 11-1](#) shows the architecture of the tuner module.

**Figure 11-1** Application architecture of the tuner module



The preceding layers are described as follows:

- Application layer
  - The application layer calls the APIs to implement the DVB and other services.
- API layer
  - The API layer encapsulates the drivers and masks the bottom layers. All tuner API names start with HI\_UNF\_TUNER.
- Driver layer
  - The driver layer provides drivers for the tuner, the Demod, and the LNB control chip.
- Hardware layer



The hardware layer includes the tuner, the Demod, the LNB control chip, and the related circuits.

## 11.2 Important Concepts

### [Demod]

Demod is the abbreviation of demodulator. The modulation mode for each standard differs, and therefore the used Demod varies. For example, QAM is used for the DVB-C, and the Demod that supports the demodulation of QAM signals is required for the STB that supports DVB-C.

### [LNB]

Installed in the dish, the LNB amplifies the satellite signals received by the dish antenna and down-converts the signals to the L waveband.

### [UniCable]

Common front-end devices that support the UniCable technology include the UniCable LNB and UniCable multi-switch. By using the UniCable technology, you can connect one dish antenna to multiple STBs or multiple dish antennas to multiple STBs by using one coaxial cable, simplifying the installation.

## 11.3 Features

The tuner module has the following features:

- Integrates some tuner drivers. For details, see the definition of [HI\\_UNF\\_TUNER\\_DEV\\_TYPE\\_E](#).
- Integrates some Demod drivers. For details, see the definition of [HI\\_UNF\\_DEMOD\\_DEV\\_TYPE\\_E](#).
- Integrates some drivers of the LNB control chip. For details, see the definition of [HI\\_UNF\\_LNBCTRL\\_DEV\\_TYPE\\_E](#).
- Supports the J.83A/B/C, DVB-S/S2/T/T2, and ISDB-T signal demodulation.
- Supports the QAM, QPSK, and 8PSK modulation modes and single-carrier or multi-carrier demodulation.
- Supports frequency locking.
- Supports the operations such as obtaining the strength, quality, signal-to-noise ratio (SNR), and bit error rate (BER) of the signals.
- Supports DVB-S/S2 blind scanning.
- Supports simple and complex DVB-T2 modes.
- Controls various DVB-S/S2 front-end devices and supports various common switches and DiSEqC1.0/1.1/1.2/universal satellites automatic location system (USALS)/UniCable.
- Allows a maximum of five Demods to work at the same time.



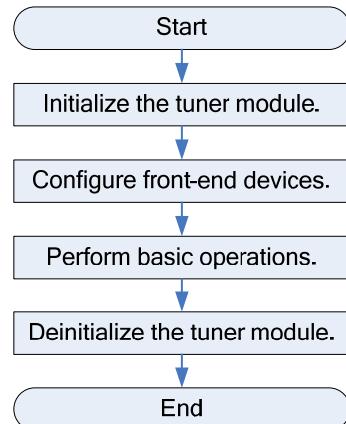
## 11.4 Development Guide

This section describes the overall operating process of the tuner module, details the sub processes and related APIs, and provides an instance for reference.

### 11.4.1 Working Process

[Figure 11-2](#) shows the overall working process.

**Figure 11-2** Overall working process



The overall working process is as follows:

**Step 1** Initialize the tuner module.

The initialization includes initializing the tuner module, starting the tuner, and setting the attributes. This operation is performed only once during a running cycle.

**Step 2** Configure the front-end devices.

The front-end devices include the LNB, various switches, and motors. These devices need to be configured after initialization when the front-end environment changes:

- The LNB needs to be configured if you want to change the LNB which is used as the signal source when there are multiple inputs.
- The switches need to be set if they are used.
- The motor needs to be set if it is to be controlled.

This step is only for the DVB-S/S2 application. For the DVB-C or DVB-T application, go to step 3.

**Step 3** Perform basic operations.

Basic operations include locking the frequency, obtaining signal information, performing blind scanning for DVB-S/S2, and querying the number of channels or selecting channels at the physical layer in DVB-T2 complex mode.

**Step 4** Deinitialize the tuner module.

After the previous operations, stop and deinitialize the tuner if it is no longer required.

**----End**



## 11.4.2 Initializing and Deinitializing the Tuner Module

### Programming Guide

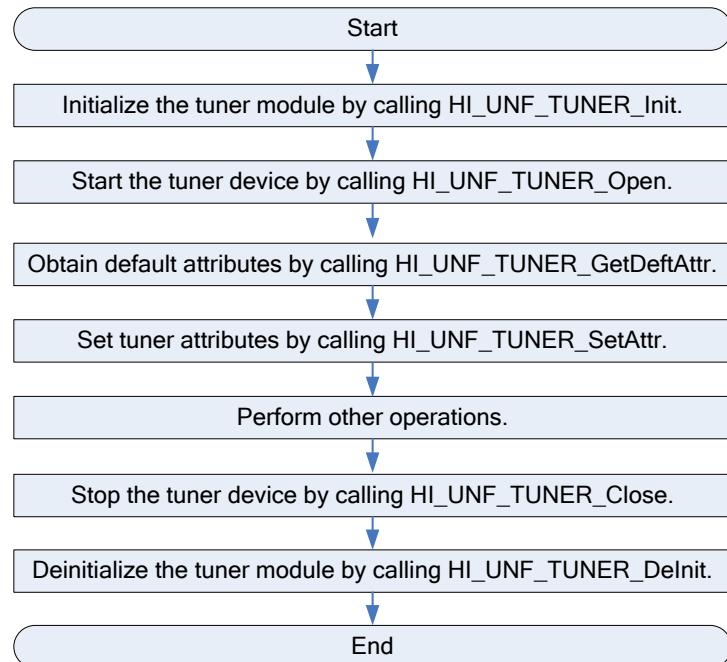
Before calling tuner APIs, initialize the tuner module. The initialization includes initializing the tuner module, starting tuner devices, and setting tuner attributes.

When tuner APIs and functions are no longer used, deinitialize the tuner module. The resources occupied by the tuner module are released after the module is deinitialized.

### Working Process

Figure 11-3 shows the working process.

**Figure 11-3** Initialization and deinitialization process



### API Description

The APIs are described as follows:

- `HI_S32 HI_UNF_TUNER_Init(HI_VOID)`: Initializes the tuner module.
- `HI_S32 HI_UNF_TUNER_Open(HI_U32 u32tunerId)`: Opens the tuner device.
- `HI_S32 HI_UNF_TUNER_GetDeftAttr(HI_U32 u32tunerId, HI_UNF_TUNER_ATTR_S *pstTunerAttr)`: Obtains the default attributes.
- `HI_S32 HI_UNF_TUNER_SetAttr(HI_U32 u32tunerId, HI_UNF_TUNER_ATTR_S *pstTunerAttr)`: Sets tuner attributes.

Note that the parameter settings vary according to the hardware configuration. Before setting these parameters, check the hardware configuration.

```
typedef struct hiTUNER_ATTR_S
```



```
{  
    HI_UNF_TUNER_SIG_TYPE_E      enSigType ;  
    HI_UNF_TUNER_DEV_TYPE_E    enTunerDevType;  
    HI_U32   u32TunerAddr;  
    HI_UNF_DEMOD_DEV_TYPE_E   enDemodDevType;  
    HI_U32   u32DemodAddr;  
    HI_UNF_TUNER_OUPUT_MODE_E  enOutputMode ;  
    HI_U8    enI2cChannel;  
    HI_U32                u32ResetGpioNo;  
} HI_UNF_TUNER_ATTR_S ;  
  
- enSigType  
    Signal type enumeration  
- enTunerDevType  
    Check the type of the tuner device and set the corresponding enumerations.  
- u32TunerAddr  
    I2C device address of the tuner  
- enDemodDevType  
    Check the type of the Demod and set the corresponding enumerations.  
- u32DemodAddr  
    I2C device address of the Demod  
- enOutputMode  
    TS output mode of the Demod. Check the modes supported by the hardware and set the corresponding enumerations.  
- enI2cChannel  
    ID of the I2C channel used by the Demod and tuner. Check the hardware and set the corresponding enumerations.  
- u32ResetGpioNo  
    Reset pin of the external Demod for controlling chip reset. For terrestrial and cable STBs, you need to only set the preceding attributes. However, for satellite STBs, you have to set more attributes. See the following.
```

- `HI_S32 HI_UNF_TUNER_SetSatAttr(HI_U32 u32tunerId, const HI_UNF_TUNER_SAT_ATTR_S *pstSatTunerAttr);`: Sets satellite STB attributes.

For the DVB-S/S2, the following parameters need to be configured:

```
typedef struct  hiUNF_TUNER_SAT_ATTR_S  
{  
    HI_U32           u32DemodClk;  
    HI_U16           u16TunerMaxLPF;  
    HI_U16           u16TunerI2CCLK;  
    HI_UNF_TUNER_RFAGC_MODE_E   enRFAGC;  
    HI_UNF_TUNER_IQSPECTRUM_MODE_E enIQSpectrum;  
    HI_UNF_TUNER_TSCLK_POLAR_E   enTSClkPolar;  
    HI_UNF_TUNER_TS_FORMAT_E    enTSFormat;  
    HI_UNF_TUNER_TS_SERIAL_PIN_E enTSSerialPIN;  
    HI_UNF_TUNER_DISEQCWAVE_MODE_E enDiSEQCWave;
```



```
    HI_UNF_LNBCTRL_DEV_TYPE_E      enLNBCtrlDev;
    HI_U16                         u16LNBDevAddress;
} HI_UNF_TUNER_SAT_ATTR_S;
```

Note the following:

- enDiSEqCWave

This parameter is used to set the object that controls (transmits and receives) the 22 kHz signals of the DiSEqC. Typically, both the Demod and the LNB can control the 22 kHz signals. You can set **enDiSEqCWave** to **HI\_UNF\_TUNER\_DISEQCWAVE\_NORMAL** to allow the Demod to control or **HI\_UNF\_TUNER\_DISEQCWAVE\_ENVELOPE** to allow the LNB to control.

- enLNBCtrlDev

Check the type of the LNB control chip and set the corresponding enumerations.

The parameter settings vary according to the hardware types and design. If you use the standard reference board, see the configuration in the sample. If you use different hardware or design, set the parameter based on the actual situation.

- **HI\_S32 HI\_UNF\_TUNER\_SetTSOUT(HI\_U32 u32TunerId, HI\_UNF\_TUNER\_TSOUT\_SET\_S \*pstTSOUT):** Sets the TS output pin.  
Currently only the TS output pin of Hi3130 V200, Hi3136 V100, and Hi3137 V100 can be flexibly configured. This API applies only to Hi3130C V200, Hi3136 V100, and Hi3137 V100.
- **HI\_S32 HI\_UNF\_TUNER\_GetAttr(HI\_U32 u32TunerId, HI\_UNF\_TUNER\_ATTR\_S \*pstTunerAttr):** Obtains the attributes.
- **HI\_S32 HI\_UNF\_TUNER\_Close(HI\_U32 u32TunerId):** Stops the tuner device.
- **HI\_S32 HI\_UNF\_TUNER\_DeInit(HI\_VOID):** Deinitializes the tuner module.

For details, see the data structure and API definition described in **hi\_unf\_frontend.h**.

### 11.4.3 Configuring Front-End Devices

This section applies only to the DVB-S/S2 application. You can ignore this section for other applications.

#### 11.4.3.1 LNB Configuration

##### Programming Guide

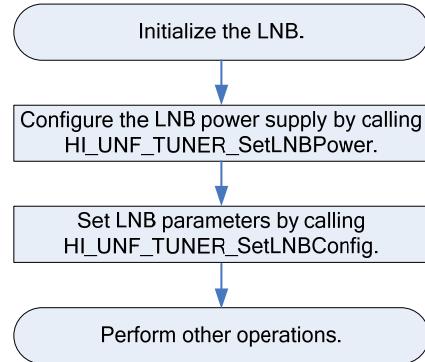
The LNB configuration includes the LNB power supply configuration and LNB parameter configuration. For details, see the following sections.

##### Working Process

Figure 11-4 shows the process for configuring the LNB.



**Figure 11-4** Process for configuring the LNB



## LNB Power Supply

### [Programming Guide]

Typically, the tuner supplies 13 V or 18 V power to support the LNB or other active devices. In some application scenarios, the receivers need to stop supplying power to the LNB. For example, when multiple receivers supply power to one LNB and the power voltages are different, the LNB may be damaged. In this case, you need to stop the receivers and allow only one receiver to supply power to the LNB. In some scenarios, the 13 V or 18 V voltage may be insufficient. For example, when the cable is very long, you can set a higher voltage, about 14 V or 19 V, if the receiver hardware allows.

### [API Description]

```
HI_S32 HI_UNF_TUNER_SetLNBPower(HI_U32 u32TunerId,  
HI_UNF_TUNER_FE_LNB_POWER_E enLNBPower);
```

You need set the power supply mode by calling the preceding APIs. The following power supply modes are supported:

- **HI\_UNF\_TUNER\_FE\_LNB\_POWER\_OFF**  
The tuner does not supply power.
- **HI\_UNF\_TUNER\_FE\_LNB\_POWER\_ON**  
The 13 V or 18 V power is supplied, which is controlled by the driver based on the signal polarity mode of the locked frequency.
- **HI\_UNF\_TUNER\_FE\_LNB\_POWER\_ENHANCED**  
A higher voltage is supplied. Note that this mode requires support of the hardware. If the hardware allows, 14 V or 19 V power is supplied; otherwise, the 13 V or 18 V power is supplied.

For details, see the data structure and API definition described in **hi\_unf\_frontend.h**.

### NOTE

This API needs to be called after initialization and before the switch and DiSEqC APIs are used. To reduce power consumption, the tuner does not supply power when it is initialized. If this API is not called, the switch and DiSEqC configurations are invalid. You can set the LNB power supply mode at any time by calling this API.



## LNB Parameters

### [Programming Guide]

In the DVB-S/S2 application, you need to convert the downlink frequency and intermediate frequency (IF) based on the LNB parameters and working status when you lock the frequency. You need to configure the LNB parameters before locking the frequency for the first time or after replacing the LNB.

### [API Description]

```
HI_S32 HI_UNF_TUNER_SetLNBCConfig(HI_U32 u32TunerId,  
HI_UNF_TUNER_FE_LNB_CONFIG_S *pstLNB);
```

This API is used to set the LNB local oscillator type (mono or dual, unicable), high/low local oscillator frequency, and working wave band (C or Ku wave band). For the unicable LNB, the ID and IF frequency of the satellite channel router (SCR) and connected port must also be configured.

For details, see the data structure and API definition described in **hi\_unf\_frontend.h**.

### 11.4.3.2 Switches and DiSEqC Devices

## Compilation Option

The DiSEqC provided in the SDK is an optional module. You can determine whether to compile it in the **make menuconfig** interface.

To implement or adapt the DiSEqC module, call

HI\_UNF\_TUNER\_DISEQC\_SendRecvMessage. You can use this API to adapt your own DiSEqC protocol stack and bypass the DiSEqC protocol stack provided by HiSilicon in the driver. For details, see the description of related data structure and API definition.

```
HI_S32 HI_UNF_TUNER_DISEQC_SendRecvMessage(HI_U32 u32TunerId,  
                                              const HI_UNF_TUNER_DISEQC_SENDMSG_S * pstSendMsg,  
                                              HI_UNF_TUNER_DISEQC_RECVMSG_S * pstRecvMsg);
```

## Features

The DiSEqC in the SDK has the following features:

- DiSEqC 1.0
- DiSEqC 1.1
- DiSEqC 1.2
- USALS
- Unicable
- Switch cascading. The cascaded switches are set in any sequence.

The commands for later versions of DiSEqC 2.0 are not supported. If you need to set the level of supported commands for the device, set it to **HI\_UNF\_TUNER\_DISEQC\_LEVEL\_1\_X**, which indicates that the DiSEqC 1.0/1.1/1.2 unidirectional commands are supported.

## Switch Control

### [Programming Guide]

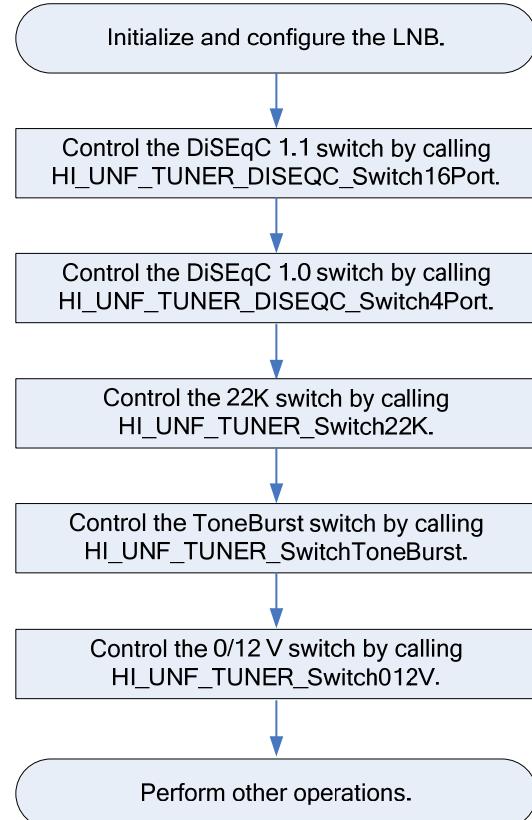


Various switches are used in the DVB-S/S2 STB application, such as the DiSEqC 1.0 switch, DiSEqC 1.1 switch, 22 kHz switch, Tone Burst switch, 0/12 V switch, and 13/18 V switch. The SDK provides corresponding APIs to control these switches. For details, see the API description in this section.

### [Working Process]

Figure 11-5 shows the typical process for controlling switches.

**Figure 11-5** Process for controlling switches



When multiple switches are used, the switch control APIs do not need to be called based on the sequence shown in Figure 11-1. In real-time configuration, you just need to call the API corresponding to the switch to be changed.



### CAUTION

If you set the power supply mode to **HI\_UNF\_TUNER\_FE\_LNB\_POWER\_OFF**, 22 kHz signals cannot be sent, and therefore the switch control cannot take effect. That is, the switches and other DiSEqC devices cannot be used when the tuner does not output power.

### [API Description]

The APIs are described as follows:



- DiSEqC 1.0 switch

```
HI_S32 HI_UNF_TUNER_DISEQC_Switch4Port(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_SWITCH4PORT_S* pstPara);
```

DiSEqC 1.0 switches are mainly 4in1 switches. You can also call this API for some special switches that can also be controlled by the DiSEqC 1.0 commands.

**NOTE**

The LNB polarity mode and 22 kHz control parameter are included in the parameters, because the DiSEqC devices (for example, LNB) need to configure these two parameters based on actual situation. If you use these devices, you need to set these two parameters. If you do not use these devices, you can set these two parameters to any valid values.

- DiSEqC 1.1 switch

```
HI_S32 HI_UNF_TUNER_DISEQC_Switch16Port(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_SWITCH16PORT_S* pstPara);
```

The switches (8in1 and 16in1) complying with DiSEqC 1.1 are controlled by this interface.

- 22 kHz switch

```
HI_S32 HI_UNF_TUNER_Switch22K(HI_U32 u32TunerId,  
HI_UNF_TUNER_SWITCH_22K_E enPort);
```

- Tone Burst switch

```
HI_S32 HI_UNF_TUNER_SwitchToneBurst(HI_U32 u32TunerId,  
HI_UNF_TUNER_SWITCH_TONEBURST_E enStatus);
```

- 0/12 V switch

```
HI_S32 HI_UNF_TUNER_Switch012V(HI_U32 u32TunerId,  
HI_UNF_TUNER_SWITCH_0_12V_E enPort);
```

**NOTE**

Currently, 0/12 V switches are not supported, but the interface is reserved. 0/12 V switches are obsolete.

- 13/18 V switch

These switches are turned on or off based on the cable voltage and are not controlled by APIs.

For details, see the data structure and API definition described in **hi\_unf\_frontend.h**.

## Switch Cascading

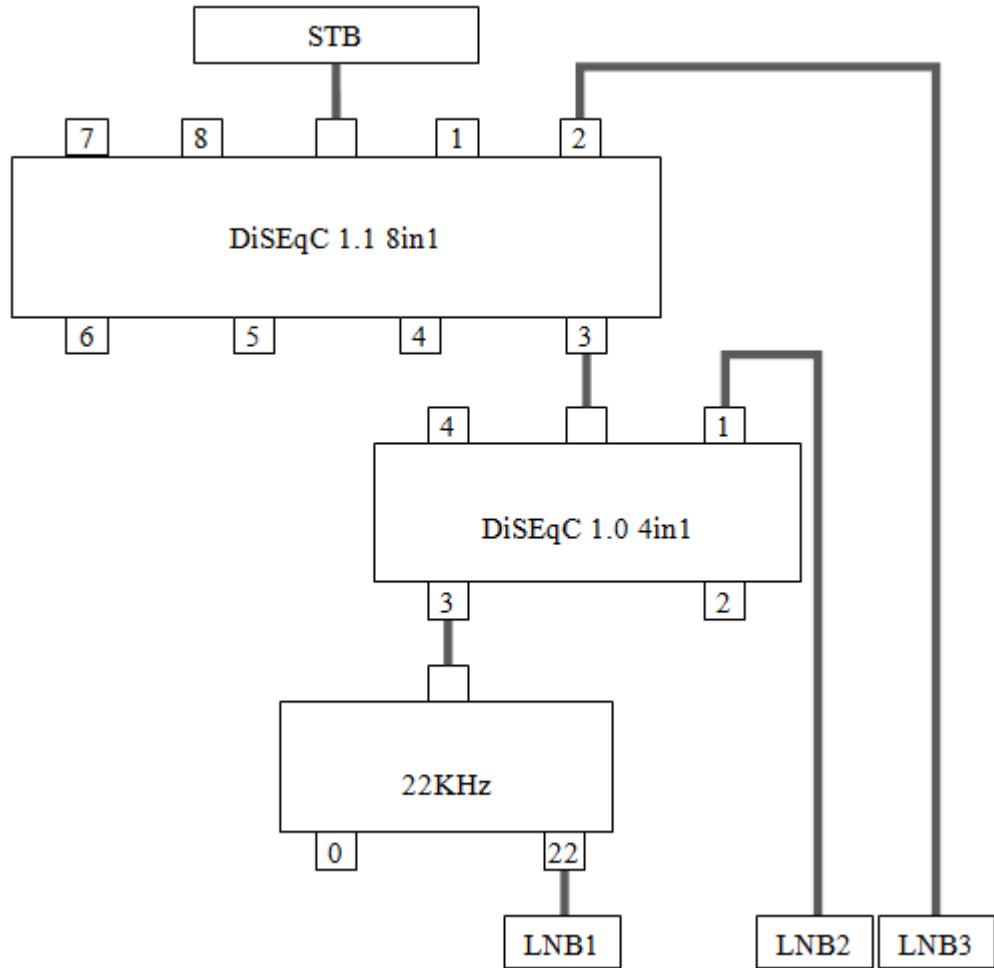
### [Programming Guide]

The DiSEqC module supports the cascading of various switches, such as 4in1-22 kHz (the 4in1 switch is cascaded with the 22 kHz switch), 8in1-22 kHz (the 8in1 switch is cascaded with the 22 kHz switch), 8in1-4in1 (the 8in1 switch is cascaded with the 4in1 switch), 8in1-4in1-22 kHz (the 8in1 switch is cascaded with the 4in1 switch, and then cascaded with the 22 kHz switch), and 4in1-8in1-22 kHz (the 4in1 switch is cascaded with the 8in1 switch, and then cascaded with the 22 kHz switch). The switch in the front is close to the receiver.

Figure 11-6 shows the typical cascading scenario.



**Figure 11-6** Typical switch cascading scenario



#### NOTE

The LNBs connect to other switch interfaces are not drawn out in the figure.

In the scenario shown in [Figure 11-1](#):

- The following interfaces need to be called when LNB1 is used:  
`Switch16Port(id, Port3);  
Switch4Port(id, Port3);  
Switch22K(id, 22);`
- The following interfaces need to be called when LNB1 is switched to LNB2:  
`Switch4Port(id, Port1);  
Switch22K(id, NONE);`
- The following interfaces need to be called when LNB2 is switched to LNB3:  
`Switch16Port(id, Port2);  
Switch4Port(id, NONE);`



Switch22K(id, NONE) must be called when LNB1 is switched to LNB2. Otherwise, 22 kHz signals will be sent consecutively, which affects blind scanning. You are advised to call Switch4Port (id, NONE) (but not mandatory) when LNB2 is switched to LNB3. Otherwise, the system considers that DiSEqC switches still exist. As a result, the system sends the corresponding commands, which takes additional tens of milliseconds. Therefore, you are advised to set all involved devices. If the switch is not used, set the corresponding parameter to **NONE**.

### [Reference Code]

The reference code is as follows:

```
HI_S32 s32TunerPort = 0;
HI_S32 s32Ret = 0;

HI_UNF_TUNER_FE_LNB_POWER_E enPower = HI_UNF_TUNER_FE_LNB_POWER_ON;
HI_UNF_TUNER_DISEQC_SWITCH16PORT_S st16Port;
HI_UNF_TUNER_DISEQC_SWITCH4PORT_S st4Port;

/* Set LNB power */
s32Ret = HI_UNF_TUNER_SetLNBPowers(s32TunerPort, enPower);
if (HI_SUCCESS != s32Ret)
{
    /* Error */
}

/* If LNB power off, 22K signal can't be sent, switch control will be not
effective */
if (HI_UNF_TUNER_FE_LNB_POWER_OFF != enPower)
{

    /* Wait DiSEqC switch to start */
    usleep(50000);

    /* Use tone burst switch */
    s32Ret = HI_UNF_TUNER_SwitchToneBurst(s32TunerPort,
HI_UNF_TUNER_SWITCH_TONEBURST_0);

    /* Use DiSEqC 1.1 Switch */
    st16Port.enLevel = HI_UNF_TUNER_DISEQC_LEVEL_1_X;
    st16Port.enPort = HI_UNF_TUNER_DISEQC_SWITCH_PORT_3;
    s32Ret |= HI_UNF_TUNER_DISEQC_Switch16Port(s32TunerPort, &st16Port);

    /* Use DiSEqC 1.0 Switch */
    st4Port.enLevel = HI_UNF_TUNER_DISEQC_LEVEL_1_X;
    st4Port.enPort = HI_UNF_TUNER_DISEQC_SWITCH_PORT_3;
    st4Port.enPolar = HI_UNF_TUNER_FE_POLARIZATION_V;
    st4Port.enLNB22K = HI_UNF_TUNER_FE_LNB_22K_ON;
```



```
s32Ret |= HI_UNF_TUNER_DISEQC_Switch4Port(s32TunerPort, &st4Port);  
  
/* Use 22K switch */  
s32Ret |= HI_UNF_TUNER_Switch22K(s32TunerPort,  
HI_UNF_TUNER_SWITCH_22K_22);  
  
if (HI_SUCCESS != s32Ret)  
{  
    /* Error */  
}  
}
```



## CAUTION

The DiSEqC devices need some time for initialization when they are powered on. After the DiSEqC devices are powered on for the first time (or they are powered on after they are powered off), you need to wait until the devices are initialized. The standard time for initialization is 100 ms. Some switches can be initialized quite fast and only need tens of ms; but some motors are initialized quite slowly, and the subsequent operations can be performed only after the antennas are in position. The wait time varies according to devices. You do not need to wait in the subsequent operations.

## Antenna Control

### [Programming Guide]

The antennas of the DiSEqC 1.2 motors can be controlled by the specific DiSEqC commands.

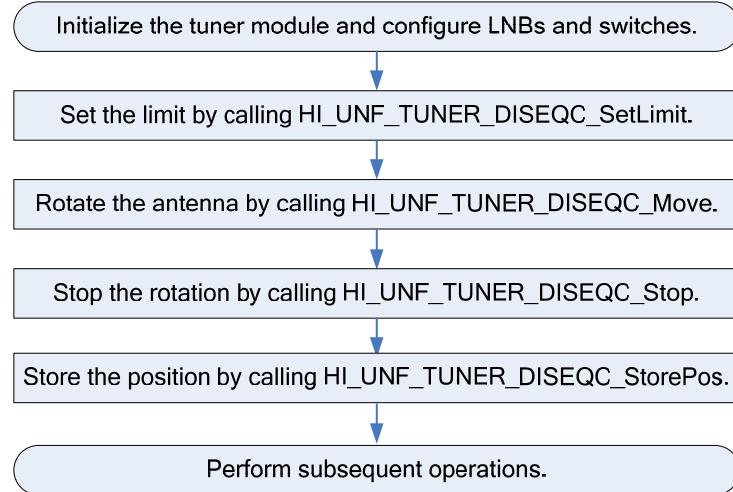
The DiSEqC module supports various functions of the DiSEqC 1.2 antennas, such as the limit control, drive, halt, store, Goto, recalculate, and Goto X. It also supports the USALS. The devices can be controlled by calling the corresponding APIs.

### [Working Process]

[Figure 11-7](#) shows the typical process for controlling antennas.



**Figure 11-7** Process for controlling antennas



### [API Description]

The APIs are described as follows:

- Limit

```
HI_S32 HI_UNF_TUNER_DISEQC_SetLimit(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_LIMIT_S* pstPara);
```

Sets the operation such as west limit, east limit, or limit off.

- Drive

```
HI_S32 HI_UNF_TUNER_DISEQC_Move(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_MOVE_S* pstPara);
```

Rotates the antenna to the east or west in steps slowly (one step each time), quickly (five steps each time), or consecutively. The timeout mode is not supported.

- Halt

```
HI_S32 HI_UNF_TUNER_DISEQC_Stop(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_LEVEL_E enLevel);
```

Stops rotating the antenna.

- Store/Goto

```
HI_S32 HI_UNF_TUNER_DISEQC_StorePos(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_POSITION_S *pstPara);  
HI_S32 HI_UNF_TUNER_DISEQC_GotoPos(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_POSITION_S *pstPara);
```

Stores the current position or goes to a stored position. Goto 0 indicates that the antenna is rotated to the 0° position.

- Recalculate

```
HI_S32 HI_UNF_TUNER_DISEQC_Recalculate(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_RECALCULATE_S* pstPara);
```

Recalculates the angle of the stored satellite.



#### NOTE

The parameters vary according to devices. For example, the typical parameter satellite *n* indicates that the angles of the other stored satellites are recalculated by using the current angle as the angle of stored satellite.

- Go to X and USALS

```
HI_S32 HI_UNF_TUNER_DISEQC_CalcAngular(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_USALS_PARA_S* pstPara);  
  
HI_S32 HI_UNF_TUNER_DISEQC_GotoAngular(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_USALS_ANGULAR_S* pstPara);
```

The USALS calculates the azimuth based on the latitude and longitude of the current position and the longitude of the target satellite, and then rotates the antenna to the specified angle by using the Goto X function.

#### NOTE

Pay attention to the parameters of west longitude and south latitude when using these two APIs. For details, see the definition of HI\_UNF\_TUNER\_DISEQC\_USALS\_PARA\_S in **hi\_unf\_frontend.h**.

For details, see the data structure and API definition described in **hi\_unf\_frontend.h**.

## Other Control

### [Programming Guide]

The DiSEqC module can send commands such as reset, standby, and power-on to the DiSEqC devices.

### [API Description]

The APIs are described as follows:

- Reset

```
HI_S32 HI_UNF_TUNER_DISEQC_Reset(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_LEVEL_E enLevel);
```

- Enter the standby mode

```
HI_S32 HI_UNF_TUNER_DISEQC_Standyby(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_LEVEL_E enLevel);
```

- Power on (resume from the standby mode)

```
HI_S32 HI_UNF_TUNER_DISEQC_WakeUp(HI_U32 u32TunerId,  
HI_UNF_TUNER_DISEQC_LEVEL_E enLevel);
```

For details, see the data structure and API definition described in **hi\_unf\_frontend.h**.

Only a few devices support the previous commands.

## 11.4.4 Locking Frequencies and Obtaining Signal Information

### [Programming Guide]

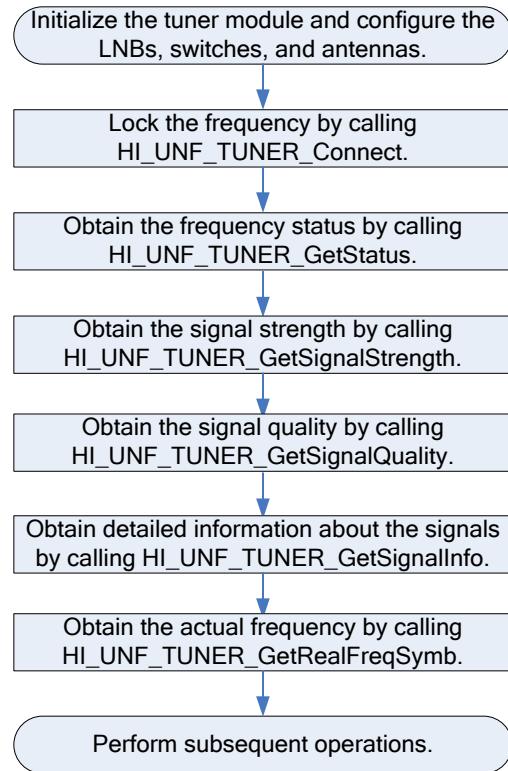
For the DVB-C and DVB-T applications, you can lock frequencies after initialization. For the DVB-S/S2 application, you need to configure the LNB and various switches or DiSEqC devices based on the environment before locking the frequencies.

### [Working Process]



Figure 11-8 shows the process for locking frequencies and performing subsequent operations.

**Figure 11-8** Process for locking frequencies and performing subsequent operations



The operations after locking the frequency can be performed in any sequence as required.

#### [API Description]

The description of the interfaces is as follows:

- Lock the frequency.

```
HI_S32 HI_UNF_TUNER_Connect(HI_U32 u32tunerId ,  
HI_UNF_TUNER_CONNECT_PARA_S *pstConnectPara, HI_U32 u32TimeOut);
```

- Obtain the status.

```
HI_S32 HI_UNF_TUNER_GetStatus(HI_U32 u32tunerId ,  
HI_UNF_TUNER_STATUS_S *pstTunerStatus);
```

- Obtain the signal strength.

```
HI_S32 HI_UNF_TUNER_GetSignalStrength(HI_U32 u32tunerId , HI_U32  
*pu32SignalStrength );
```

- Obtain the signal quality.

```
HI_S32 HI_UNF_TUNER_GetSignalQuality(HI_U32 u32TunerId, HI_U32  
*pu32SignalQuality);
```

Only DVB-S/S2 supports this API. The signal quality percentage is returned based on the modulation mode. For the DVB-C signals, you can obtain the SNR as the signal quality.

- Obtain the signal information.



```
HI_S32 HI_UNF_TUNER_GetSignalInfo(HI_U32 u32TunerId,  
HI_UNF_TUNER_SIGNALINFO_S *pstSignalInfo);
```

Only DVB-S/S2/T/T2 support this API. The signal modulation mode, FEC parameters are detected by the Demod and can be obtained by calling this API.

- Obtain the actual frequency.

```
HI_S32 HI_UNF_TUNER_GetRealFreqSymb( HI_U32 u32TunerId, HI_U32  
*pu32Freq, HI_U32 *pu32Symb );
```

This API can be used to obtain the frequency offset and symbol rate deviation.

- Obtain the SNR.

```
HI_S32 HI_UNF_TUNER_GetSNR(HI_U32 u32tunerId , HI_U32 *pu32SNR );
```

- Obtain the BER.

```
HI_S32 HI_UNF_TUNER_GetBER(HI_U32 u32tunerId , HI_U32 *pu32BER);
```

For details, see the data structure and API definition described in **hi\_unf\_frontend.h**.

## 11.4.5 Blind Scanning

Currently only the DVB-S/S2 applications support bind scanning.

### [Programming Guide]

The blind scanning function provided by the tuner module includes automatic blind scanning and manual blind scanning. For details about the difference between these two modes, see API description.

- The frequency information and blind scanning status and progress are notified by using callback functions.
- Blind scanning can be stopped.

### [API Description]

The APIs are described as follows:

- Start blind scanning.

```
HI_S32 HI_UNF_TUNER_BlindScanStart(HI_U32 u32TunerId,  
HI_UNF_TUNER_BLINDSCAN_PARA_S *pstPara);
```

You can start blind scanning by calling this API. You need to set the following parameters:

```
typedef struct hiUNF_TUNER_BLINDSCAN_PARA_S  
{  
    HI_UNF_TUNER_BLINDSCAN_MODE_E enMode;  
    union  
    {  
        HI_UNF_TUNER_SAT_BLINDSCAN_PARA_S stSat;  
    } unScanPara;  
} HI_UNF_TUNER_BLINDSCAN_PARA_S;  
  
typedef struct hiUNF_TUNER_SAT_BLINDSCAN_PARA_S  
{
```



```
HI_UNF_TUNER_FE_POLARIZATION_E enPolar;
HI_UNF_TUNER_FE_LNB_22K_E enLNB22K;
HI_U32 u32StartFreq;
HI_U32 u32StopFreq;

HI_VOID (*pfnDISEQCSet)(HI_U32 u32TunerId,
HI_UNF_TUNER_FE_POLARIZATION_E enPolar,
HI_UNF_TUNER_FE_LNB_22K_E enLNB22K);
HI_VOID (*pfnEVTNotify)(HI_U32 u32TunerId,
HI_UNF_TUNER_BLINDSCAN_EVT_E enEVT,
HI_UNF_TUNER_BLINDSCAN_NOTIFY_U * punNotify);
} HI_UNF_TUNER_SAT_BLINDSCAN_PARA_S;
```

Blind scanning includes automatic blind scanning and manual blind scanning.

- HI\_UNF\_TUNER\_BLINDSCAN\_MODE\_AUTO

In automatic blind scanning mode, the driver scans all frequency bands based on the current LNB configuration. The driver automatically manages the polarity mode and LNB 22 kHz configuration. The upper software does not need to specify the start or end frequency for scanning. The parameters do not need to be configured. Even if you configure them, the configurations are invalid.

- HI\_UNF\_TUNER\_BLINDSCAN\_MODE\_MANUAL

If you need to scan only a specified frequency band, you can manually scan it. In manual scanning mode, you need to set the polarity mode, LNB 22 kHz, and start and end frequencies for scanning. Note that the frequencies are IFs ranging from 950,000 kHz to 2,150,000 kHz.

In addition, you need to register the following two callback functions:

- pfnDISEQCSet for setting DiSEqC devices

The LNB polarity mode and 22 kHz status need to be configured for some DiSEqC devices, because the polarity mode and 22 kHz status may change during blind scanning. In this case, you need to register pfnDISEQCSet. If you do not use those DiSEqC devices or do not need to configure the driver, you can set pfnDISEQCSet to **NULL**. If you register this function, you can call it when the LNB polarity mode and the 22 kHz status change.

- pfnEVTNotify for notifying the user when the blind scanning status changes, the blind scanning progress changes, or a new frequency is scanned

This function must be registered. The notification function is called if any of the preceding three events occur. Blind scanning status changes when blind scanning starts or ends (including normal completion, abnormal completion, and user termination). The progress change granularity is 1%. The pfnEVTNotify function is called when a new frequency is scanned. For the DVB-S/S2, the scanned frequency information includes frequency, symbol rate, polarity mode, and TP reliability.

- Stop scanning

```
HI_S32 HI_UNF_TUNER_BblindScanStop(HI_U32 u32TunerId);
```

For details, see the data structure and API definition described in **hi\_unf\_frontend.h**.

## 11.4.6 Sample

See **sample/tuner/tuner\_demo.c** in the SDK.



## 11.5 FAQs

### 11.5.1 Debugging information

#### 11.5.1.1 Does the Tuner Module Have /proc Debugging Information?

##### Analysis

The tuner module provides a /proc debugging interface.

##### Solution

Run the following command in the serial port console:

```
cat /proc/msp/tuner
```

The tuner configuration information, locked frequency status, and signal information are displayed as follows:

```
-----Hisilicon TUNER Info-----
Port:LockStat I2CChannel Frequency SymbRate QamMode DemodType TunerType
      0:    locked      3          1310000     27500000   QPSK       avl6211   SHARP7903
BER:4299316*(E-7),  SNR:965,  SignalStrength:26398
FEC type:DVBS,  FEC rate:3/4
```

Table 11-1 describes the preceding information.

**Table 11-1** Tuner debugging information

Item	Value	Description
Port	0	Device 0 is used.
LockStat	locked	Locked
I2CChannel	3	I <sup>2</sup> C 3 is used.
Frequency	1310000	The locked signal frequency is 1,310,000 kHz. For the DVB-S/S2, it is IF.
SymbRate	27500000	The locked signal symbol rate is 27,500,000 symbol/s.
QamMode	QPSK	The locked signal uses the quadrature phase shift keying (QPSK) modulation mode.
DemodType	avl6211	The Demod type is AVL6211.
TunerType	SHARP7903	The tuner type is SHARP7903.
BER	4299316*(E-7)	BER, SNR, signal strength levels
SNR	965	Note: The displayed values are the values directly read in the driver and are only for reference. You can obtain the related value by calling the corresponding API.
SignalStrength	26398	



Item	Value	Description
FEC type	DVBS	DVBS and 3/4 forward error correction (FEC) modes are used.
FEC rate	3/4	These two items only exist in DVB-S/S2.

## 11.5.2 Common Issues

### 11.5.2.1 What Do I Do If I<sup>2</sup>C Information Fails to Be Displayed During Initialization?

#### Symptom

The following information is displayed during initialization:

```
[942806 ERROR-ecs]:I2C_DRV_WaitWriteEnd[102]:wait write data timeout!  
[942812 ERROR-ecs]:I2C_DRV_Write[230]:wait write data timeout!  
[942819 ERROR-ecs]:I2C_DRV_Write[201]:wait write data timeout!
```

#### Analysis

If the information keeps being displayed and a message indicating initialization failure is returned, the I<sup>2</sup>C channel or the slave addresses of related devices may be incorrectly configured.

#### Solution

Check whether the I<sup>2</sup>C channel ID, tuner type and its slave address, and Demod type and its slave address are correctly configured based on hardware. For the DVB-S/S2, you also need to check the LNB control chip type and its slave address.

### 11.5.2.2 What Do I Do If the Frequency Cannot Be Locked?

#### Symptom

The frequency is set but cannot be locked.

#### Analysis

This issue may be caused for the following reasons:

- An error occurs in the signal source.
- The environment is incorrectly configured.
- The parameters are incorrect.

#### Solution

Do as follows based on the causes:

- An error occurs in the signal source.



Check whether the problem persists by replacing the used device with other receiving or test devices. Ensure that the receiving device, transfer device, and the cable work properly.

- The environment is configured incorrectly.

In the DVB-S/S2 environment, check whether the switches are used. If the switches are used, ensure that the switches are properly configured by calling the corresponding APIs. The power splitter is not affected.

- The parameters are incorrect.

Check whether the parameters are correctly configured. Pay attention to the unit. Ensure that the frequency locked by the DVB-S/S2 is the downlink frequency but not the IF.

### 11.5.2.3 What Do I Do If the Frequency Is Locked but the Program Cannot Be Played?

#### Symptom

The frequency is successfully locked but the program cannot be played.

#### Analysis

You can check whether the frequency is successfully locked based on the debugging information. For details, see section [11.5.1 "Debugging information."](#)

If the frequency is successfully locked but the program cannot be played, check whether the subsequent operations are correct.

#### Solution

If you use an external Demod:

- Check whether the port bound to the DEMUX is correct.  
Check it based on the hardware.
- Check whether data is received from the correct DEMUX.

The subsequent operations are not described in this document. For details, see the related documents.

### 11.5.2.4 What Do I Do If the Switches or DiSEqC Devices Cannot Be Controlled?

#### Symptom

The corresponding APIs are called, but the devices do not respond properly.

#### Analysis

This issue may be caused for the following reasons:

- The LNB power supply is disabled.
- The connection cable is abnormal.
- The device is abnormal.
- The device does not support the command.



- The 22 kHz signals (signal frequency, amplitude, or edge) do not comply with the standard.

## Solution

Check the following items in sequence:

- The LNB power supply is disabled.  
Set the LNB power supply to ON or ENHANCED.
- The connection cable is abnormal.

If the connection cable is loose, a command transmission error may occur. Therefore, screw the connectors tightly. If the F-type connector is not matched, a conversion adapter is recommended. A connection cable of low quality may also result in errors. The connection cable of good quality is recommended.

- The device is abnormal.  
Check that the device works properly.
- The device does not support the command.  
Check whether the sent commands are supported by the device.

- The signals from the STB do not comply with the standard.  
Check whether the sent commands are correct. Some devices support only a few commands. Therefore, check whether the sent commands are supported by the device.



# Contents

---

<b>12 KEYLED .....</b>	<b>1</b>
12.1 Overview .....	1
12.2 Important Concepts .....	1
12.3 Features .....	2
12.4 Development Guide.....	2
12.4.1 Keyboard Scanning and LED Display .....	3
12.4.2 Porting New KEYLED Drivers .....	5



## Figures

<b>Figure 12-1</b> KEYLED module in the system .....	1
<b>Figure 12-2</b> Working process for keyboard scanning and LED display .....	4

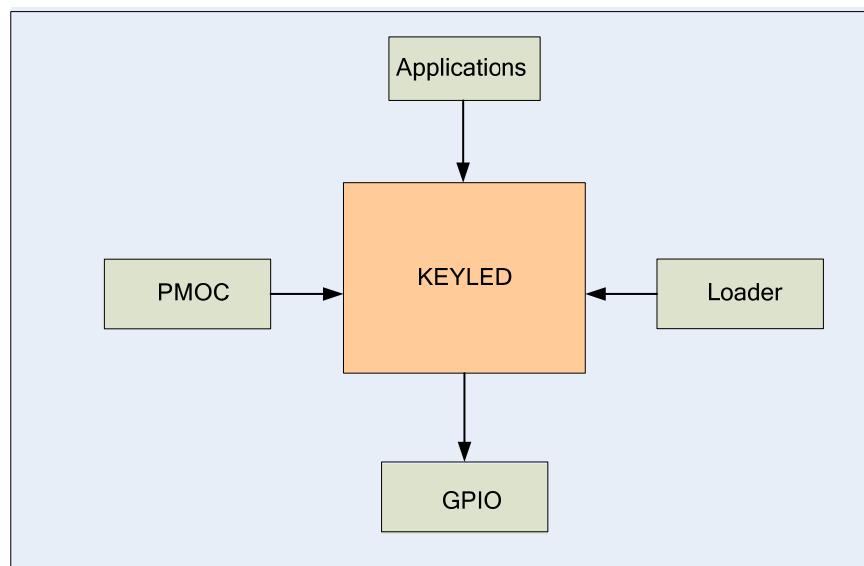


# 12 KEYLED

## 12.1 Overview

The KEYLED module implements keyboard scanning and light-emitting diode (LED) display functions. [Figure 12-1](#) shows the KEYLED module in the system.

**Figure 12-1** KEYLED module in the system



The KEYLED module includes various chip drivers, and some of them depend on the GPIO module. The KEYLED module provides keyboard scanning and LED display functions for the PMOC and loader modules and upper-layer applications.

## 12.2 Important Concepts

[LED]

The LED is a solid semiconductor component that can converts the electric power into visible light.



### [Nixie tube]

A Nixie tube consists of eight segments, that is, seven LEDs and a decimal point, in the shape of 8. The segments to which power is supplied are lit and form the symbol we can see.

### [Driver chip]

The driver chip indicates the chip integrated on the front panel for keyboard and LED control. The front panel type described in the SDK indicates the driver chip type on the front panel.

## 12.3 Features

The KEYLED module provides the following functions:

- KEYLED module management. The following APIs are provided:
  - HI\_UNF\_KEYLED\_Init: Initializes the KEYLED module.
  - HI\_UNF\_KEYLED\_DeInit: Deinitializes the KEYLED module.
  - HI\_UNF\_KEYLED\_SelectType: Selects the KEYLED driver type.
- Keyboard scanning, including obtaining the key values and status of the keyboard on the front panel. The following APIs are provided:
  - HI\_UNF\_KEY\_Open: Enables the key function.
  - HI\_UNF\_KEY\_Close: Disables the key function.
  - HI\_UNF\_KEY\_Reset: Clears the key values that are not received.
  - HI\_UNF\_KEY\_GetValue: Obtains a key value.
  - HI\_UNF\_KEY\_SetBlockTime: Sets the timeout period of reading keys.
  - HI\_UNF\_KEY\_ReportKeyTimeoutVal: Sets the report time interval between which keys can be pressed repeatedly.
  - HI\_UNF\_KEY\_IsRepKey: Enables or disables the function of reporting the same key.
  - HI\_UNF\_KEY\_IsKeyUp: Enables or disables the function of reporting the released status of a key.
- LED display on the front panel. The following APIs are provided:
  - HI\_UNF\_LED\_Open: Enables the display function of LEDs.
  - HI\_UNF\_LED\_Close: Disables the display function of LEDs.
  - HI\_UNF\_LED\_Display: Displays characters based on the type of the input display code.
  - HI\_UNF\_LED\_DisplayTime: Displays the time on LEDs.
  - HI\_UNF\_LED\_SetFlashPin: Sets the ID of the LED that is to blink.
  - HI\_UNF\_LED\_SetFlashFreq: Sets the blink frequency level of an LED.

## 12.4 Development Guide

The KEYLED module is used in the following scenarios:

- Keyboard scanning and LED display on the front panel
- Porting new KEYLED drivers



## 12.4.1 Keyboard Scanning and LED Display

### Scenario

The key values and status of the keyboard on the front panel is obtained, and numbers or time is displayed on the LED as required. The LED is typically used during STB standby mode or normal playback.

### Working Process

The process for keyboard scanning and LED display is as follows:

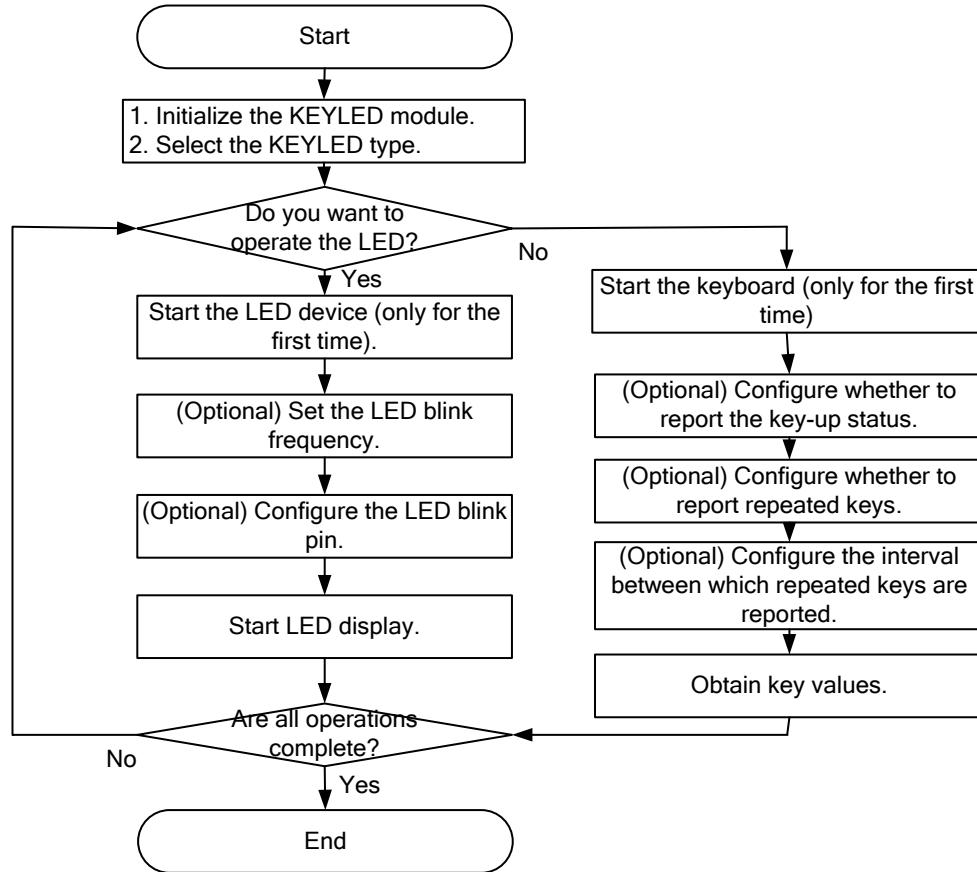
- Step 1** Initialize the KEYLED module by calling HI\_UNF\_KEYLED\_Init.
- Step 2** Select the KEYLED type based on the hardware by calling HI\_UNF\_KEYLED\_SelectType.
- Step 3** Start the LED devices on the front panel by calling HI\_UNF\_LED\_Open.
- Step 4** (Optional) Set the ID of the LED that is to blink by calling HI\_UNF\_LED\_SetFlashPin.
- Step 5** (Optional) Set the LED blink frequency by calling HI\_UNF\_LED\_SetFlashFreq.
- Step 6** Update the LED display by calling HI\_UNF\_LED\_Display or HI\_UNF\_LED\_DisplayTime.
- Step 7** Start the keyboard on the front panel by calling HI\_UNF\_KEY\_Open.
- Step 8** (Optional) Set whether to report the release status of a key by calling HI\_UNF\_KEY\_IsKeyUp.
- Step 9** (Optional) Set whether to report repeated keys by calling HI\_UNF\_KEY\_IsRepKey.
- Step 10** (Optional) Set the time interval between which keys can be pressed repeatedly by calling HI\_UNF\_KEY\_RepKeyTimeoutVal.
- Step 11** Obtain the key value in block mode by calling HI\_UNF\_KEY\_GetValue.
- Step 12** Stop the LED device on the front panel by calling HI\_UNF\_LED\_Close.
- Step 13** Stop the key device on the front panel by calling HI\_UNF\_KEY\_Close.
- Step 14** Deinitialize the KEYLED module by calling HI\_UNF\_KEYLED\_DeInit.

----End

Figure 12-2 shows the working process for keyboard scanning and LED display.



**Figure 12-2** Working process for keyboard scanning and LED display



## Notes

The current SDK allows you to choose whether to compile the KEYLED drivers and which KEYLED drivers are to be compiled into the code based on the configuration file to meet the clipping requirements. By default, not all KEYLED drivers are compiled. If a front panel is not compiled into the driver code, a code indicating failure is returned when you choose the front panel type by calling HI\_UNF\_KEYLED\_SelectType.

Data is updated to the LED in the next cycle after HI\_UNF\_LED\_Display is called. If HI\_UNF\_KEYLED\_GetValue is called when some keys are pressed, valid key values are returned; if HI\_UNF\_KEYLED\_GetValue is called but no key is pressed, an error code indicating operation failure is returned.

During LED display, display codes are formed based on the arrangement order and features (co-cathode or co-anode) of LEDs. Each bit of the display code controls a segment of scanned LEDs. In co-anode mode, value 0 indicates that the LED is on; in co-cathode mode, value 1 indicates that the LED is on. For example, the display code of value 8 is 0xFF in co-cathode mode.

## Sample

See `sample_keyled.c`.



## 12.4.2 Porting New KEYLED Drivers

### Scenario

The driver of the HiSilicon KEYLED module supports the following devices:

- KEYLED of the PT6961 chip driver
- KEYLED of the CT1642 chip driver
- KEYLED of the PT6964 chip driver
- KEYLED of the FD650 chip driver



### CAUTION

The KEYLED device can work properly only with the support of pins of the master chip and hardware board in addition to software drivers.

If you want to use other KEYLED devices, the KEYLED driver must be ported.

### Working Process

To port new KEYLED drivers, perform the following steps:

**Step 1** Add a KEYLED enumeration in **source/msp/include/hi\_unf\_keyled.h**.

**Step 2** Create a folder named a KEYLED device in **source/msp/drv/keyled**.

**Step 3** Implement the following APIs at the KEYLED driver layer. For details about the functions of these APIs, see the related development reference.

```
HI_S32 KEYLED_KEY_Open(HI_VOID);  
HI_S32 KEYLED_KEY_Close(HI_VOID);  
HI_S32 KEYLED_KEY_Reset(HI_VOID);  
HI_S32 KEYLED_KEY_GetValue(HI_U32 *pu32PressStatus, HI_U32 *pu32KeyId);  
HI_S32 KEYLED_KEY_SetBlockTime(HI_U32 u32BlockTimeMs);  
HI_S32 KEYLED_KEY_SetRepTime(HI_U32 u32RepTimeMs);  
HI_S32 KEYLED_KEY_IsRepKey(HI_U32 u32IsRepKey);  
HI_S32 KEYLED_KEY_IsKeyUp(HI_U32 u32IsKeyUp);  
HI_S32 KEYLED_LED_Open(HI_VOID);  
HI_S32 KEYLED_LED_Close(HI_VOID);  
HI_S32 KEYLED_LED_Display(HI_U32 u32CodeValue);  
HI_S32 KEYLED_LED_DisplayTime(HI_UNF_KEYLED_TIME_S stKeyLedTime);  
HI_S32 KEYLED_LED_SetFlashPin(HI_UNF_KEYLED_LIGHT_E enPin);  
HI_S32 KEYLED_LED_SetFlashFreq(HI_UNF_KEYLED_LEVEL_E enLevel);  
HI_S32 KEYLED_LED_DisplayLED(HI_KEYLED_DISPLAY_S LedData);
```

**Step 4** Add the functions of the new driver to the global structure **g\_stKeyLedOpt** in the parameter **KEYLEDSelectType** in **source/msp/drv/keyled/drv\_keyled\_intf.c**.



**Step 5** Modify the **Makefile** file in **source/msp/drv/keyled**, add the new file to **Makefile**, and recompile and install the KEYLED.

----End

## Notes

The preceding steps add only one KEYLED driver on Linux and do not involve development dedicated for specific modules (such as standby and loader). You can also add drivers to these modules as required by referring to the preceding steps, but the file paths and APIs need to be modified based on the developed module.

## Sample

None



# Contents

---

<b>13 IR .....</b>	<b>1</b>
13.1 Overview .....	1
13.1.1 Overall Architecture .....	1
13.1.2 Drive Architecture .....	1
13.2 Important Concepts .....	2
13.3 Features .....	4
13.4 Development Guide.....	4
13.4.1 Obtaining the Key Value and Key Status of the IR Remote Control.....	5
13.4.2 Querying and Enabling/Disabling a Remote Control Protocol .....	6
13.4.3 Developing the Driver for a Remote Control Protocol .....	6



# Figures

<b>Figure 13-1</b> Position of the IR module in the system .....	1
<b>Figure 13-2</b> IR driver architecture .....	2
<b>Figure 13-3</b> Process for obtaining the key value and key status of the IR remote control .....	5



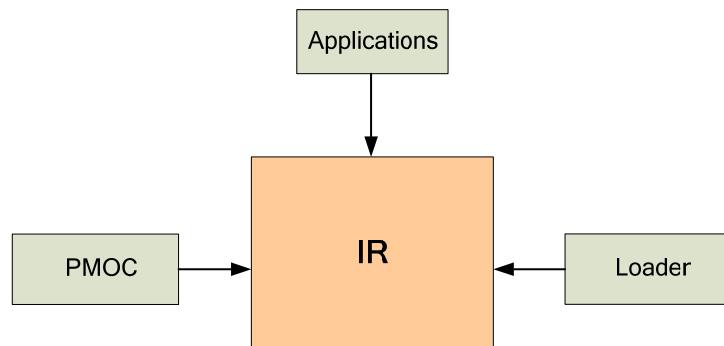
# 13 IR

## 13.1 Overview

### 13.1.1 Overall Architecture

The IR module is used to obtain key values and key states of a remote control over the IR interface. [Figure 13-1](#) shows the position of the IR module in the system.

**Figure 13-1** Position of the IR module in the system



The IR module relies only on the chip. It obtains key values and key states for the PMOC and loader modules and upper-layer applications.

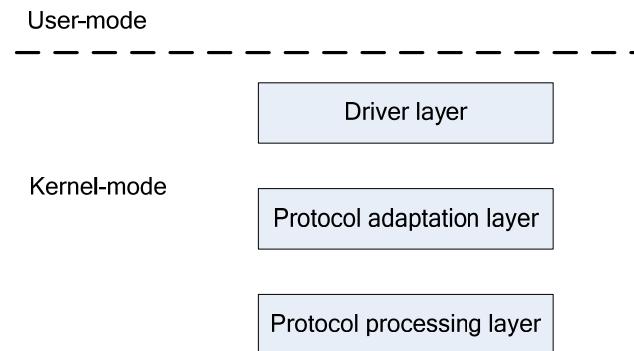
### 13.1.2 Drive Architecture

#### Architecture Diagram

[Figure 13-2](#) shows the IR driver architecture.



**Figure 13-2** IR driver architecture



## Driver Layer

The driver handles interrupts for the IR module, traverses all protocols at the bottom half, and calls the match function to identify the protocol that a received IR frame complies with. If no match is detected, the driver discards the current frame.

If the current frame cannot be parsed this time, the driver waits for about 200 ms and attempts to parse the frame again. If the attempt fails, the driver discards the current frame.

### NOTE

The driver waits 200 ms to ensure that IR key press information is not lost.

## Protocol Adaptation Layer

The driver initializes various protocols, provides protocol traversing interfaces for the upper layer, and offers space for storing protocol descriptors to the lower layer.

## Protocol Processing Layer

The driver determines whether a frame of symbols starting from a specific symbol complies with the current protocol.

If a specific frame complies with the current protocol, the driver parses key values from the frame.

If an error occurs during parsing or a frame complying with the current protocol cannot be parsed, the driver handles the error.

## 13.2 Important Concepts

[Remote protocol]

The common protocols for data decoding of IR signals include the NEC, Philips, and Sony protocols.

[Remote key value]

The values obtained by decoding the IR signals using the IR module based on different protocols range from 12 bits to 48 bits.



### [Remote key status]

There are three key states: key down, key hold, and key up.

### [Remote raw level]

The IR module may receive an unprocessed level pair that contains a low level followed by a high level or a high level followed by a low level.

Typically a level pair indicates a bit0 or bit1 of the key values of the remote control.

### [NEC simple]

A frame in NEC Simple format consists of a start code (lead code), a data code, and a burst signal. The start code consists of a start code (low level) and an end code (high level). The valid bits and the definition of each bit in the data code are determined by the code type.

The typical protocol parameters are as follows (in  $\mu$ s):

```
pulse, space, factor
{9000, 4500, 10},      /* header phase */
{560, 560, 50},        /* b0 phase */
{560, 1690, 50},       /* b1 phase */
{560, 40000, 50},      /* burst phase */
```

32 bits, including the total number of bits 0 and 1

#### NOTE

The **factor** column specifies the upper and lower limits for the pulse and space of a specific symbol during infrared transfer.

Use the lead code as an example. The upper and lower limits for the lead code pulse (header phase) are calculated as follows:

$[9000 \times (100 - 10)/100, 9000 \times (100 + 10)/100]$

### [NEC full]

NEC full is basically the same as NEC Simple, only that if a complete data frame (first frame) is received after the key is held down for more than one frame duration, the subsequently received data frame is still a complete data frame.

### [RC6]

A frame in RC6 format consists of a lead code, a start bit, a toggle bit, and a data code. The level width of each part is defined by Philips.

The IR driver supports remote controls where each frame contains 16 bits, 20 bits, and 32 bits.

### [RC5]

A frame in RC5 format consists of start pulses, a toggle bit, and a data code. The level width of each part is defined by Philips.

### [SONY/TC9012]

For details about frame formats, see the *Hi37XX Data Sheet*.

### [Card remote control protocol]

Each frame transmitted from a card remote control consists of one start bit, eight data bits, one end bit, and one stuffing bit. This version of driver uses the start bit as a lead code and the stuffing bit as an end bit to identify whether a frame of infrared signal complies with the card



remote control protocol. The driver skips over the end bit and parses only the eight data bits. The card remote control protocol is forbidden by default for security reasons and can be enabled by software. The driver does not parse a level pair received from a card remote control when the protocol is forbidden.

[Remote control protocol with dual lead codes]

The remote control protocol with dual lead codes is similar to the NEC simple or NEC full protocol except that a lead code level is inserted to each frame of the infrared signal. There are two types of remote control protocol with dual lead codes:

- Protocol with dual lead codes, similar to the NEC simple protocol
- Protocol with dual lead codes, similar to the NEC full protocol

## 13.3 Features

The IR module provides the following functions:

- IR module management, including initializing and deinitializing the IR module and setting related parameters. The following APIs are provided:
  - HI\_UNF\_IR\_Init: Initializes the IR module.
  - HI\_UNF\_IR\_DeInit: Deinitializes the IR module.
  - HI\_UNF\_IR\_Enable: Enables the IR module.
  - HI\_UNF\_IR\_EnableKeyUp: Sets whether to report the key up status of the keys.
  - HI\_UNF\_IR\_EnableRepKey: Sets whether to report repeated keys.
  - HI\_UNF\_IR\_SetRepKeyTimeoutAttr: Sets the interval of reporting repeated keys of the KEY device.
  - HI\_UNF\_IR\_SetFetchMode: Sets whether to obtain key values or raw levels from the IR driver.
  - HI\_UNF\_IR\_Reset: Clears the buffer for key values and symbols.
- Remote control key obtaining, including obtaining the key values, key states, key protocols, or raw levels. The following APIs are provided:
  - HI\_UNF\_IR\_GetValueWithProtocol: Obtains the key value, key status, and key protocol of the remote control.
  - HI\_UNF\_IR\_GetSymbol: Obtains the raw level of the remote control key.
- IR remote control protocol management, including enabling or disabling the remote control protocols and obtaining the enable status of the remote control protocols. The following APIs are provided:
  - HI\_UNF\_IR\_EnableProtocol: Enables a remote control protocol.
  - HI\_UNF\_IR\_DisableProtocol: Disables a remote control protocol.
  - HI\_UNF\_IR\_GetProtocolEnabled: Obtains the enable status of a remote control protocol.

## 13.4 Development Guide

The IR module is used in the following scenarios:

- Obtaining the key value and key status of the IR remote control



- Querying and enabling/disabling a remote control protocol
- Developing the driver for a new remote control protocol

### 13.4.1 Obtaining the Key Value and Key Status of the IR Remote Control

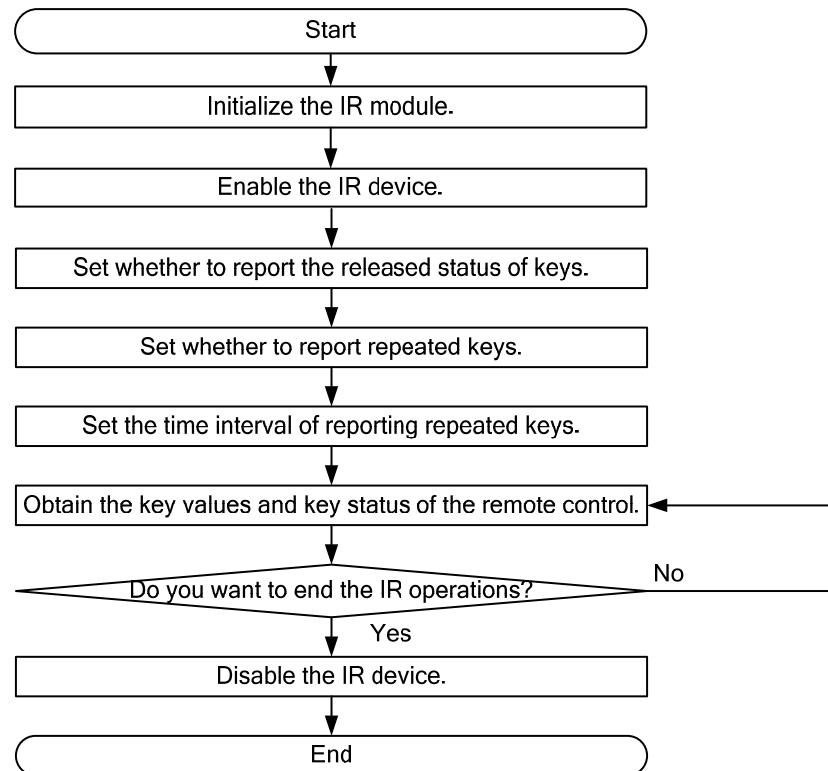
#### Scenario

You need to obtain the key value and key status in many scenarios. For example, key values must be obtained to determine the operation in the STB menu or to determine whether to wake up the system in standby mode.

#### Working Process

Figure 13-3 shows the working process for obtaining the key value and key status of the IR remote control.

**Figure 13-3** Process for obtaining the key value and key status of the IR remote control



#### Notes

When you call `HI_UNF_IR_GetValueWithProtocol` to obtain the key value, if you do not need to obtain the protocol type, you can set the third parameter `pszProtocolName` to **NULL** and the fourth parameter `s32NameSize` to **0**.



## Sample

See [sample\\_ir.c](#).

### 13.4.2 Querying and Enabling/Disabling a Remote Control Protocol

#### Scenario

The current SDK supports the compilation of various remote control protocols in the code. However, the protocols may be disabled by default. This scenario applies when the development engineers do not know which protocols are enabled or a disabled protocol is to be used.

#### Working Process

To query and enable/disable a remote control protocol, perform the following steps:

- Step 1** Initialize the IR module by calling `HI_UNF_IR_Init`.
- Step 2** Query the status of a remote control protocol by calling `HI_UNF_IR_GetProtocolEnabled`.
- Step 3** Determine whether to enable or disable a remote control protocol based on the obtained status. Call `HI_UNF_IR_EnableProtocol` to enable a remote control protocol or `HI_UNF_IR_DisableProtocol` to disable a remote control protocol.
- Step 4** Deinitialize the IR module by calling `HI_UNF_IR_DeInit`.

----End

#### Notes

If the driver for a remote control protocol is not compiled into the code, a code indicating failure is returned when you obtain or set the status of the remote control protocol.

## Sample

See [sample\\_ir.c](#).

### 13.4.3 Developing the Driver for a Remote Control Protocol

#### 13.4.3.1 Scenario

Remote control protocols that are not supported by the SDK need to be developed.

#### 13.4.3.2 Working Process

The methods for extending IR protocols are as follows:

- Modify `ir_protocols` in the `drv_ir_protocols_descript.c` file.
- Compile a new driver and dynamically register the IR protocol.

If you use the first method, you need to compare the HiSilicon SDK with your modifications and manually integrate code into the version based on the comparison result during each version switching, which is inefficient.



If you use the second method, you need to compile a driver and you will find that version switching is easy.

## Modifying ir\_protocols

To modify ir\_protocols, perform the following steps:

**Step 1** Check the **ir\_protocols** array. Check for description items that are the same as the protocol to be added in the lead code, bit 0, bit 1, end code, and bit count. If there are no such items, go to Step 2.

**Step 2** Add a new remote control protocol.

- Add a component to the IR\_PROTOCOL\_IDX enumeration type to describe the protocol.
- Declare initialization and deinitialization processes in **ir\_protocol.h** as required. Implement initialization in the **ir\_protocol\_init** function and deinitialization in the **ir\_protocol\_exit** function.
- Provide the APIs for the hook functions **phase\_match** and **frame\_parser**. They are described in the code.
  - The **phase\_match** hook function is called by the driver layer to check whether a specific frame complies with the current protocol.
  - The **frame\_parser** hook function is called by the driver layer to parse a frame that is identified as a protocol-compliant frame by **phase\_match**.
- Add description items to **ir\_protocols**. You are advised to add them before the RC6 description item. For details about the reason, see the notes.

**Step 3** Verify as follows based on the type of the added protocol:

- If the protocol type is NEC, check whether the **priv** component is greater than or equal to or duplicate with **MAX\_NECK\_INFR\_NR**. Ensure that it is less than **MAX\_NECK\_INFR\_NR** and has no duplicate value.
- If the protocol is RC5, verify that the **priv** component is less than **MAX\_RC5\_INFR\_NR** and has no duplicate value after adding description items to **ir\_protocols**. The RC5 protocol must be placed at the end of the **ir\_protocols** array.
- If the protocol is RC6, verify that the **priv** component is less than **MAX\_RC6\_INFR\_NR** and has no duplicate value after adding description items to **ir\_protocols**. The RC6 protocol must be placed before the RC5 protocol and after the NEC protocol.
- The preceding steps for other remote control protocols are similar. Note that the factors of each item (such as the start bit, bit 0, bit 1, end bit, and repetition code) that compose a frame must conform to the upper and lower limits of the level pair for the current remote control protocol and be different from those for other remote control protocols especially in the lead code to avoid lead code overlapping between two protocols.

----End

## Dynamically Registering the IR Protocol

To dynamically register the IR protocol, perform the following steps:

**Step 1** Create the **ir\_s2/custom/ir\_custom.c** file:

```
#include <linux/device.h>
```



```
#include <linux/module.h>
#include <linux/moduleparam.h>
#include <linux/fs.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <asm/delay.h>
#include <linux/poll.h>
#include <mach/hardware.h>
#include <linux/interrupt.h>
#include <linux/himedia.h>
#include <linux/types.h>
#include <linux/sched.h>
#include <linux/spinlock.h>
#include <linux/list.h>

#include "ir.h"
#include "../ir_protocol.h"
#include "../ir_utils.h"
extern int ir_unregister_protocol(struct ir_protocol *ip);
extern int ir_register_protocol(struct ir_protocol *ip);

static struct ir_protocol ir_prot_custom [] =
{
    {{0}, NULL, IR_PROT_BUTT, INFR_IFFY, {{0}}, NULL, 0, NULL}
};

static int ir_custom_init(void)
{
    int ret;
    int i;
    struct ir_protocol *ip;
    for (i = 0; i < ARRAY_SIZE(ir_prot_custom); i++) {
        ip = &ir_prot_custom[i];
        if (!ip->ir_code_name || !ip->match || !ip->handle
            || ip->idx == IR_PROT_BUTT)
            break;
        ret = ir_register_protocol(ip);
        if (ret) {
            printk("Fail to register ir_prot_custom[%d], name %s\n", i, ip-
>ir_code_name);
        }
    }
    return 0;
}

static void ir_custom_exit(void)
```



```
{  
    int ret;  
    int i;  
    struct ir_protocol *ip;  
    for (i = 0; i < ARRAY_SIZE(ir_prot_custom); i++) {  
        ip = &ir_prot_custom[i];  
        if (!ip->ir_code_name || !ip->match || !ip->handle  
            || ip->idx == IR_PROT_BUTT)  
            break;  
        ret = ir_unregister_protocol(ip);  
        if (ret) {  
            printk("Fail to register ir_prot_custom[%d], name %s\n", i, ip->ir_code_name);  
        }  
    }  
    return ;  
}  
module_init(ir_custom_init);  
module_exit(ir_custom_exit);
```

**Step 2** Create the **ir\_s2/custom/Makefile** file:

```
ifndef SRC_ROOT  
export SRC_ROOT := $(PWD)/../../../../..  
endif  
  
EXTRA_CFLAGS += -I$(DRV_SRC_BASE) -I$(SRC_ROOT)/pub/include/  
EXTRA_CFLAGS += -I$(KCOM_DIR)/include -I$(KCOM_DIR)/drv/include -  
I$(KCOM_DIR)/api/include  
EXTRA_CFLAGS += -I$(KECS_DIR)/include -I$(KECS_DIR)/drv/include  
EXTRA_CFLAGS += $(CFG_CFLAGS) -DSDK_VERSION=$(CFG_SDK_VERSION)  
  
obj-m += custom.o  
custom-y := ir_custom.o
```

**Step 3** Add the IR frame description item of the new remote control in the **ir\_prot\_custom** array in **ir\_custom.c**.

**Step 4** Add match and parsing functions **frame\_parser** and **phase\_match** and other required hook functions.

**Step 5** Add new C code to **custom/Makefile** so that the code can be compiled. For example, add the C code **a.c** after **custom-y := ir\_custom.o**. In the statement, **custom** is the prefix in the name of the .ko file to be generated.

**Step 6** Add the statement **obj-m += custom/** to **ir\_s2/Makefile**.

**Step 7** Compile the SDK and copy the generated **custom.ko** file to the board file system.

----End



## Debugging the Remote Control

Debug the remote control as follows:

- Change the print level of the IR module from FATAL to DEBUG to record the symbols transferred from hardware to the software buffer and the key values successfully parsed by software.
- For a new remote control protocol, use a sample that contains a **-m 1** parameter to determine the infrared waveform, and implement match and parsing functions.

### 13.4.3.3 Notes

## RC5 Protocol

Take the following precautions for the RC5 protocol:

- The levels of bits 0 and 1 in RC5 are determined by the level transition direction, and RC5 has no lead code different from bits 0 and 1. When the driver attempts to identify the protocol that a series of symbols starting from a specific symbol comply with, it checks the RC5 protocol in the last position. This prevents the symbol in which the level width of bits 0 and 1 complies with RC5 from being considered as the start of a RC5 frame.
- The levels of bits 0 and 1 in RC5 are determined by the level transition direction. A series of high and low levels may be combined and reported by the IR module to software. In this case, software may determine that a burst bit comes before its expected position. Additional efforts are required to check whether a burst bit comes before its expected position.

## RC6 Protocol

Take the following precautions for the RC6 protocol:

- The levels of bits 0 and 1 in RC6 are determined by the level transition direction. A series of high and low levels may be combined and reported by the IR module to software. In this case, software may determine that a burst bit comes before its expected position. Additional efforts are required to check whether a burst bit comes before its expected position.
- RC6 has a lead code different from bits 0 and 1. Therefore, RC6 description items must be placed before RC5 ones in the IR protocol linked list.

## Card Remote Control Protocol

A frame in the card remote control protocol consists of one start bit (bit 0), eight data bits, one end bit (bit 1), and one stuffing bit. The level width of the stuffing bit is slightly different from that of bits 0 and 1.

Each frame that complies with the card remote control protocol basically consists of a series of bits 0 and 1. If the lead code is lost but data bits exist in a frame complying with another protocol, the frame may be mistakenly parsed as a frame complying with the card remote control protocol. Therefore, the card remote control protocol is forbidden by default. You can enable it by using applications.



## Remote Control Protocol on the Guangdong Network

Remote controls on the Guangdong network are special. Remote control data on the Guangdong network complies with the dual-lead code protocol similar to NEC simple, but this protocol has two versions that are different only in the lead code of the repetition code.

When a frame without the lead code is parsed, errors may occur. The driver parses such a frame with the protocol version it identifies first. When users press a key on a remote control complying with one of the protocol versions (or similar protocols) on the Guangdong network, the driver may parse a frame received as a frame complying with the other version. The case is directional. For example, if protocol B is registered before protocol A in the driver, protocol A may be parsed as protocol B, but protocol B will not be parsed as protocol A.

## Sequence for Arranging IR Protocols

Based on the descriptions in preceding sections, the sequence requirements for arranging IR protocols are as follows:

- Protocols with a specific start bit, end bit, and fixed bit count must be placed in front. For example, NEC simple and NEC full are this type of protocols.
- The RC6 protocol must be placed in the next position.
- The RC5 protocol must be placed after the RC6 protocol.
- The remote control protocols that are not embedded in the driver and are added by users must be placed at the end by default.

## Key Up Event

The ioctl command **CMD\_IR\_ENABLE\_KEYUP** controls whether to report the key up events. If the command value is **1**, key up events are reported. If the command value is **0**, key up events are not reported. The driver reports the key up events by default.

The delay for reporting a key up event is 300 ms by default. Exercise caution when modifying this parameter. If the delay is too long, several consecutive key presses will be misreported as a key hold event. If the delay is too short, holding a key would be misreported as a DOWN/HOLD..UP/DOWN/..UP sequence.

The delay time 300 ms is the debug value for NEC remote controls, which are more popular than RC5 or RC6 remote controls. If a key is repeatedly pressed on an RC5 or RC6 remote control, a DOWN/HOLD/HOLD.../UP sequence but not a DOWN/UP/DOWN/UP sequence is reported.

The driver does not process this issue by default. To rectify this issue, adjust the module parameter **key\_hold\_timeout**.

## Key Hold Event

The ioctl command **CMD\_IR\_ENABLE\_REPKEY** controls whether to report the key hold events.

If the command value is **1**, the key hold events are reported. If the command value is **0**, the key hold events are not reported.

The default interval for reporting the key hold events is 200 ms. You can set it by calling **HI\_UNF\_IR\_SetRepKeyTimeoutAttr**.



### 13.4.3.4 Sample

None



---

# Contents

---

<b>14 GPIO .....</b>	<b>1</b>
14.1 Overview .....	1
14.2 Important Concepts .....	1
14.3 Features .....	1
14.4 Development Guide.....	2



# Figures

**Figure 14-1** Working process of the GPIO module ..... 2



# 14 GPIO

## 14.1 Overview

The GPIO module is used to read and write to GPIO pins.

## 14.2 Important Concepts

None

## 14.3 Features

The GPIO module provides the following functions:

- Sets/Obtains the input/output mode of the GPIO pin.
  - HI\_UNF\_GPIO\_SetDirBit: Sets the input/output mode of the GPIO pin.
  - HI\_UNF\_GPIO\_GetDirBit: Obtains the input/output mode of the GPIO pin.
- Reads/Writes the level status of the GPIO pin.
  - HI\_UNF\_GPIO\_ReadBit: Reads the level status of the GPIO pin.
  - HI\_UNF\_GPIO\_WriteBit: Writes the level status of the GPIO pin.
- Sets the GPIO interrupt function in input mode.
  - HI\_UNF\_GPIO\_SetIntType: Sets the interrupt type.
  - HI\_UNF\_GPIO\_SetIntEnable: Enables/Disables the interrupt.
  - HI\_UNF\_GPIO\_QueryInt: Queries the GPIO pin that generates the interrupt.
- Sets/Obtains the I/O output mode of the GPIO pin.
  - HI\_UNF\_GPIO\_SetOutputType: Sets the I/O output mode of the GPIO pin.
  - HI\_UNF\_GPIO\_GetOutputType: Obtains the I/O output mode of the GPIO pin.



## 14.4 Development Guide

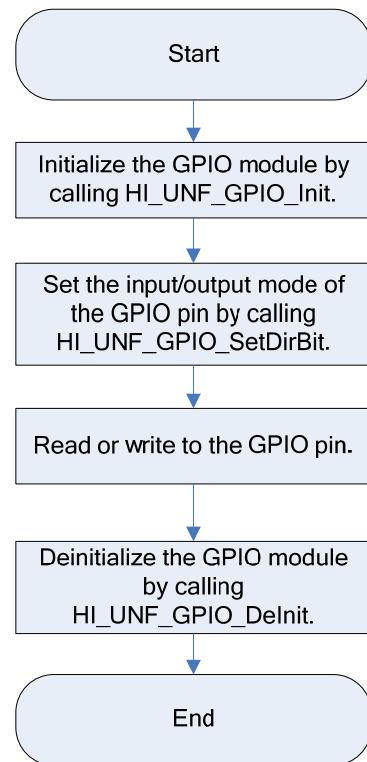
### Scenario

Data needs to be read from or written to a GPIO pin

### Working Process

Figure 14-1 shows the working process of the GPIO module.

**Figure 14-1** Working process of the GPIO module



### Notes

- Before using the GPIO function, you must set the multiplexing function of required pins. This ensures that external pins are used as GPIO pins.
- Before enabling a GPIO pin to output data, ensure that the pin mode is set to output; before enabling a GPIO pin to input data, ensure that the pin mode is set to input.
- Only some GPIO pins of some chips support the configuration of I/O output mode. For details, see the corresponding data sheet.
- The pin ID is calculated as follows:

Pin ID = GPIO group ID x 8 + GPIO pin ID in the group

Both pin group ID and pin ID are numbered from 0. For example, GPIO1\_2 indicates pin 2 in group 1, and the pin ID is 10 (1 x 8 + 2).



For Hi3796C V100 and Hi3798C V100, the IDs of available GPIO groups are 0–14, 18, and 19. GPIO18 contains the pins GPIO\_STB0–GPIO\_STB7, and GPIO19 contains the pins GPIO1\_STB0–GPIO1\_STB7.

## Sample

The reference code is as follows:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include "hi_unf_gpio.h"

int gpio_wrrd_demo(void)
{
    int s32Status;
    HI_U32 u32Temp;
    HI_BOOL bBitVal;

    HI_UNF_GPIO_Init ();

    /*Set pin 3 in GPIO group 0 to input.*/
    s32Status = HI_UNF_GPIO_SetDirBit(3, HI_TRUE);
    if ( s32Status!= HI_SUCCESS ){
        HI_UNF_GPIO_DeInit();
        return s32Status;
    }

    /*Read data from pin 3 in GPIO group 0.*/
    s32Status = HI_UNF_GPIO_ReadBit(3, &bBitVal);
    if ( s32Status!= HI_SUCCESS ){
        HI_UNF_GPIO_DeInit();
        return s32Status;
    }
    else
    {
        printf("the value of gpio[%d] is %s\n", 3, bBitVal?"HIGH":"LOW");
    }

    /*Set pin 1 in GPIO group 0 to output.*/
    s32Status = HI_UNF_GPIO_SetDirBit (1, HI_FALSE);
    if ( s32Status!= HI_SUCCESS ){
        HI_UNF_GPIO_DeInit();
        return s32Status;
    }
```



```
}

/*Write data to pin 1 in GPIO group 0.*/
s32Status = HI_UNF_GPIO_WriteBit(1, HI_TRUE);
if ( s32Status!= HI_SUCCESS ){
    HI_UNF_GPIO_DeInit();
    return s32Status;
}

HI_UNF_GPIO_DeInit();
return 0;
}
```



# Contents

---

<b>15 I<sup>2</sup>C .....</b>	<b>1</b>
15.1 Overview .....	1
15.2 Important Concepts .....	1
15.3 Features .....	1
15.4 Development Guide.....	2



## Figures

**Figure 15-1** Process for reading and writing to an I<sup>2</sup>C device ..... 2



# 15 I<sup>2</sup>C

## 15.1 Overview

The HiSilicon HD chips provide the standard I<sup>2</sup>C bus and support the I<sup>2</sup>C bus simulated by the GPIO, facilitating control over and access to various external devices.

## 15.2 Important Concepts

[I<sup>2</sup>C]

The I<sup>2</sup>C bus is a two-wire serial bus developed by Philips for connecting micro controllers and peripherals.

## 15.3 Features

The I<sup>2</sup>C module supports the following functions:

- Creates/Destroys GPIO-simulated I<sup>2</sup>C channels.
  - HI\_UNF\_I2C\_CreateGpioI2c: Creates a GPIO-simulated I<sup>2</sup>C channel.
  - HI\_UNF\_I2C\_DestroyGpioI2c: Destroys a GPIO-simulated I<sup>2</sup>C channel.
- Reads/Writes data over standard I<sup>2</sup>C/GPIO-simulated I<sup>2</sup>C channels.
  - HI\_UNF\_I2C\_Read: Reads data from a component over an I<sup>2</sup>C channel.
  - HI\_UNF\_I2C\_Write: Writes data to a component over an I<sup>2</sup>C channel.
- Sets the standard I<sup>2</sup>C communication rate.
  - HI\_UNF\_I2C\_SetRate: Sets the standard I<sup>2</sup>C communication rate (only supports several rates).
  - HI\_UNF\_I2C\_SetRateEx: Sets the standard I<sup>2</sup>C communication rate (supports many rates).

## 15.4 Development Guide

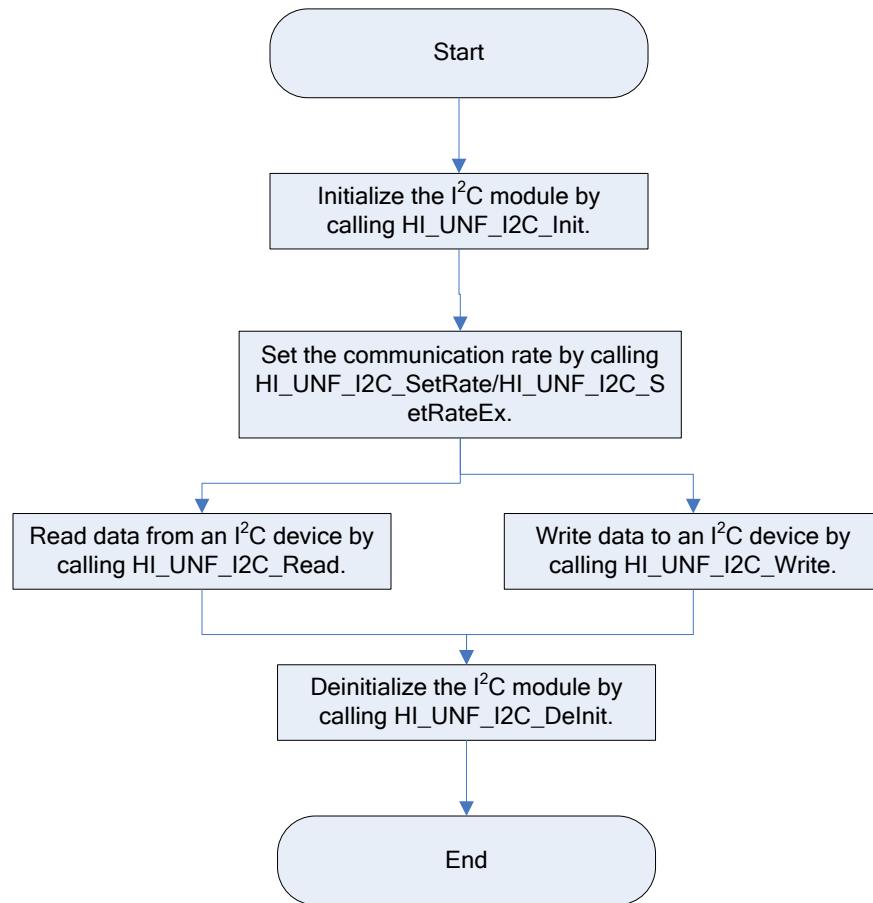
### Scenario

The I<sup>2</sup>C channel is used to communicate with electrically erasable programmable read-only memory (E2PROM).

### Working Process

Before using an I<sup>2</sup>C device (such as reading or writing to an I<sup>2</sup>C device), you must initialize the I<sup>2</sup>C module. [Figure 15-1](#) shows the process for reading and writing to an I<sup>2</sup>C device.

**Figure 15-1** Process for reading and writing to an I<sup>2</sup>C device



### Notes

- The chip provides eight I<sup>2</sup>C modules. [Table 15-1](#) describes the mapping between the I<sup>2</sup>C modules and I<sup>2</sup>C channel IDs as well as pins.

**Table 15-1** Mapping between the I<sup>2</sup>C modules and I<sup>2</sup>C channel IDs as well as pins

I2C Channel ID	I2C Module	Pins
0	I2C0	I2C0_SDA/I2C0_SCL
1	I2C1	I2C1_SDA/I2C1_SCL
2	I2C2	I2C2_SDA/I2C2_SCL
3	I2C3	I2C3_SDA/I2C3_SCL
4	I2C4	I2C4_SDA/I2C4_SCL
5	I2C5	I2C5_SDA/I2C5_SCL
6	I2C_ADC	For internal use, with no pin
7	I2C_QAM	For internal use, with no pin

- Initialize the I<sup>2</sup>C module by calling the initialization API before performing any other operations.
- Deinitialize the I<sup>2</sup>C module by calling HI\_UNF\_I2C\_DeInit when it is no longer used.
- Ensure that the I<sup>2</sup>C driver is loaded before using the I<sup>2</sup>C bus for communication.
- You are allowed to set the standard I<sup>2</sup>C communication rate, which is 400 kHz by default, but not the GPIO-simulated I<sup>2</sup>C communication rate.
- After creating a GPIO-simulated I<sup>2</sup>C channel by calling HI\_UNF\_I2C\_CreateGpioI2c, you can use it in the same way as the standard I<sup>2</sup>C channel before destroying the channel by calling HI\_UNF\_I2C\_DestroyGpioI2c.

## Sample

The reference code is as follows:

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "hi_unf_i2c.h"

#define TEST_E2PROM_PORT 1 /*Assume that the E2PROM is mounted on I2C
channel 1.*/

HI_S32 main(HI_S32 argc, HI_CHAR **argv)
{
    HI_S32 s32Ret = HI_FAILURE;
    HI_U32 u32Loop;
    HI_U8 au8WriteBuf[32];
    HI_U8 au8ReadBuf[32];
```



```
s32Ret = HI_UNF_I2C_Init();
if (HI_SUCCESS != s32Ret)
{
    printf("%s: %d ErrorCode=0x%x\n", __FILE__, __LINE__, s32Ret);
    return s32Ret;
}

printf("data write:\n");
for (u32Loop = 0; u32Loop < sizeof(au8WriteBuf); u32Loop++)
{
    au8WriteBuf[u32Loop] = u32Loop & 0xff;

    printf("0x%02x ", au8WriteBuf[u32Loop]);
}
printf("\n");

/* Write data to E2PROM */
s32Ret = HI_UNF_I2C_Write(TEST_E2PROM_PORT, 0xA0, 0, 2, au8WriteBuf,
sizeof(au8WriteBuf));
if (s32Ret != HI_SUCCESS)
{
    printf("call HI_I2C_Write failed.\n");
    HI_UNF_I2C_DeInit();
    return s32Ret;
}

usleep(10000);

memset(au8ReadBuf, 0, sizeof(au8ReadBuf));
/* Read data from E2PROM */
s32Ret = HI_UNF_I2C_Read(TEST_E2PROM_PORT, 0xA0, 0, 2, au8ReadBuf,
sizeof(au8ReadBuf));
if (s32Ret != HI_SUCCESS)
{
    printf("call HI_I2C_Read failed.\n");
    HI_UNF_I2C_DeInit();
    return s32Ret;
}

printf("\ndata read:\n");
for (u32Loop = 0; u32Loop < sizeof(au8ReadBuf); u32Loop++)
{
    printf("0x%02x ", au8ReadBuf[u32Loop]);
}
```



```
printf("\n\n");

/*check whether the read data is equal to the write data */
for (u32Loop = 0; u32Loop < 32; u32Loop++)
{
    if (au8WriteBuf[u32Loop] != au8ReadBuf[u32Loop])
    {
        printf("Error:au8WriteBuf[%d] = %#x, au8ReadBuf[%d] = %#x\n",
u32Loop, au8WriteBuf[u32Loop], u32Loop, au8ReadBuf[u32Loop]);
    }
}

HI_UNF_I2C_DeInit();

return HI_SUCCESS;
}
```



---

# Contents

---

<b>16 SCI.....</b>	<b>1</b>
16.1 Overview .....	1
16.2 Features .....	1
16.3 Development Guide.....	2



# Figures

---

**Figure 16-1** Working process of the SCI module ..... 2



# 16 SCI

## 16.1 Overview

The SCI module enables smart cards to communicate with the STB. In the following scheme, the SCI module supports the smart cards that comply with the T0, T1, and T14 protocols.

## 16.2 Features

The SCI module supports the following functions:

- Sets hardware-related parameters.
  - HI\_UNF\_SCI\_ConfigVccEn: Sets the valid level of the VCCEN signal line.
  - HI\_UNF\_SCI\_ConfigDetect: Sets the valid level of the DETECT signal line.
  - HI\_UNF\_SCI\_ConfigClkMode: Sets the I/O output mode of the clock line.
  - HI\_UNF\_SCI\_ConfigResetMode: Sets the I/O output mode of the reset signal line.
  - HI\_UNF\_SCI\_ConfigVccEnMode: Sets the I/O output mode of the VCCEN signal line.
- Sets and obtains communication-related parameters.
  - HI\_UNF\_SCI\_Open: Sets the communication protocol and clock frequency.
  - HI\_UNF\_SCI\_SetEtuFactor: Sets the specified elementary time unit (ETU) clock rate factor and baud rate regulator factor.
  - HI\_UNF\_SCI\_SetGuardTime: Sets the extra interval for transmitting two consecutive bytes from the terminal to an IC card.
  - HI\_UNF\_SCI\_SetCharTimeout: Sets the character timeout period of T0 or T1\*/.
  - HI\_UNF\_SCI\_SetBlockTimeout: Sets the block timeout period of T1.
  - HI\_UNF\_SCI\_SetTxRetries: Sets the maximum number of transmission retries after a check error occurs.
  - HI\_UNF\_SCI\_GetParams: Obtains the communication parameters.
  - HI\_UNF\_SCI\_GetCardStatus: Obtains the current status of the smart card.
- Sends/Receives data.
  - HI\_UNF\_SCI\_ResetCard: Resets the smart card.
  - HI\_UNF\_SCI\_GetATR: Obtains the answer to reset (ATR).
  - HI\_UNF\_SCI\_Send: Sends data to a smart card.



- HI\_UNF\_SCI\_Receive: Receives data from a smart card.
- Supports PPS.
  - HI\_UNF\_SCI\_NegotiatePPS: Initiates PPS negotiation.
  - HI\_UNF\_SCI\_GetPPSResponData: Obtains PPS negotiation data.

## 16.3 Development Guide

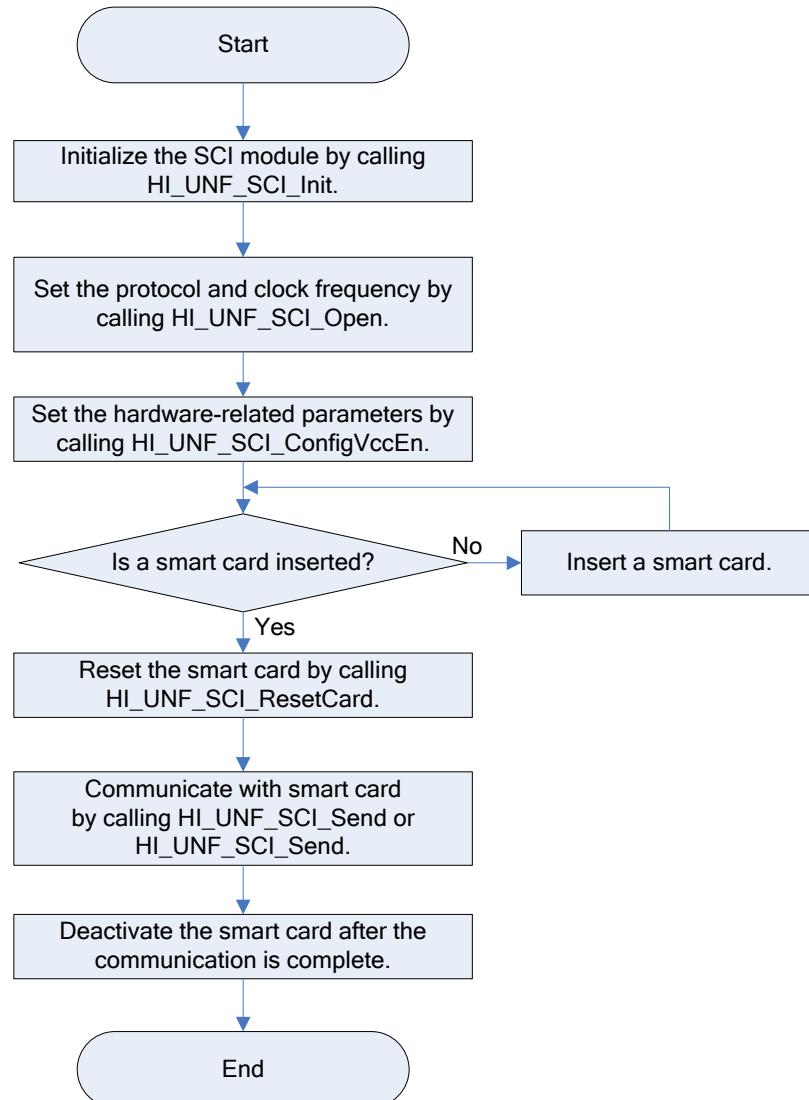
### Scenario

The SCI module is used to communicate with the smart card (sending/receiving data).

### Working Process

[Figure 16-1](#) shows the working process of the SCI module.

**Figure 16-1** Working process of the SCI module





The details are as follows:

- Step 1** Initialize the SCI module by calling HI\_UNF\_SCI\_Init.
- Step 2** Enable an SCI by calling HI\_UNF\_SCI\_Open.
- Step 3** Set the valid levels of the VCCEN and DETECT signal lines and the I/O output modes of the clock line, reset signal line, and VCCEN signal line by calling HI\_UNF\_SCI\_ConfigVccEn, HI\_UNF\_SCI\_ConfigDetect, and HI\_UNF\_SCI\_ConfigClkMode, HI\_UNF\_SCI\_ConfigResetMode, and HI\_UNF\_SCI\_ConfigVccEnMode based on the hardware.
- Step 4** Check whether a smart card is inserted by calling HI\_UNF\_SCI\_GetCardStatus. If no card is inserted, you need to insert a smart card.
- Step 5** Reset the smart card by calling HI\_UNF\_SCI\_ResetCard.
- Step 6** Obtain the status of the smart card by calling HI\_UNF\_SCI\_GetCardStatus. If the smart card status is less than HI\_UNF\_SCI\_STATUS\_READY 26 seconds after the reset operation, the card is considered invalid. According to the 7816-3 protocol, the maximum ATR timeout period is (9600 x 32 x 372) clock cycles. The 26 seconds here are the maximum ATR timeout value calculated when the actual clock frequency is 4.5 MHz.
- Step 7** Obtain the ATR data by calling HI\_UNF\_SCI\_GetATR or transmit an instruction that can be identified by the smart card for communication by calling HI\_UNF\_SCI\_Send.
- Step 8** Deactivate the smart card by calling HI\_UNF\_SCI\_DeactiveCard.

----End

## Notes

A smart card can be read or written only after being activated. A smart card may fail to be activated due to card damage or protocol incompatibility. If the protocol is incompatible, you need to read all the documents provided by the smart card vendor, because instruction sets are related to the design of the smart card. Do not remove the smart card during operations.

## Sample

The reference code is as follows:

```
#include <sys/types.h>
#include "hi_unf_sci.h"

/*Define the instructions that can be identified by a smart card.*/
static HI_U8 SCICMD[] = { ..... };
static HI_U8 DataBuf[255];

HI_S32 main(void)
{
    HI_U8 ATRCount;
    HI_U8 i;
    HI_U8 ATRBuf[ 255];
```



```
HI_S32 m = 0;
HI_S32 s32Ret = HI_SUCCESS;
HI_U32 CardStatus = 0;
HI_U32 times = 0;
HI_U32 u32ActualLength = 0

s32Ret = HI_UNF_SCI_Init();
if (HI_SUCCESS != s32Ret)
{
    break;
}

s32Ret = HI_UNF_SCI_ConfigVccEn(0, HI_UNF_SCI_LEVEL_LOW);
s32Ret |= HI_UNF_SCI_ConfigDetect(0, HI_UNF_SCI_LEVEL_LOW);
s32Ret |= HI_UNF_SCI_ConfigClkMode(0, HI_UNF_SCI_CLK_MODE_CMOS);
s32Ret |= HI_UNF_SCI_ConfigResetMode(0, HI_UNF_SCI_CLK_MODE_CMOS);
s32Ret |= HI_UNF_SCI_ConfigVccEnMode(0,
HI_UNF_SCI_CLK_MODE_CMOS);if (HI_SUCCESS != s32Ret)
{
    break;
}

//T0 smart card
HI_UNF_SCI_GetCardStatus(0, &CardStatus);

while (CardStatus <= HI_UNF_SCI_STATUS_NOCARD) //Check whether the card is
inserted.
{
    printf("Please Insert Card \n");
    sleep(1);
    HI_UNF_SCI_GetCardStatus(0, &CardStatus);
}

s32Ret = HI_UNF_SCI_ResetCard(0, HI_TRUE);
if (HI_SUCCESS != s32Ret)
{
    break;
}

while (CardStatus < HI_UNF_SCI_STATUS_READY) //Check whether the
card is activated.
{
    sleep(1);
    HI_UNF_SCI_GetCardStatus(0, &CardStatus);
```



```
        times++;
        if (times > 26)
        {
            printf(" The Card is Wrong! \n");
            HI_UNF_SCI_Close(0);
            HI_UNF_SCI_DeInit();
            return HI_FAILURE;
        }
    }

    HI_UNF_SCI_GetATR(0, ATRBuf, sizeof(ATRBuf), &ATRCount); //Obtain ATR data.
    printf("ATR DATA \n");
    for (i = 0; i < ATRCount; i++)
    {
        printf("0x%x", ATRBuf[i]);
    }

    s32Ret = HI_UNF_SCI_Send(0, SCICMD, sizeof(SCICMD),
&u32ActualLength, 5000);
    if (HI_SUCCESS != s32Ret)
    {
        printf("Send Error\n");
        break;
    }

    //The actual size of received data is related to the transmitted
    command.
    s32Ret = HI_UNF_SCI_Receive(0, DataBuf, 32, &u32ActualLength,
5000);
    if (HI_SUCCESS != s32Ret)
    {
        printf("Receive Error\n");
        break;
    }
    else
    {
        printf("Receive Data\n");
        for (m = 0; m < u32ActualLength; m++)
        {
            printf("0x%x", DataBuf [m]);
        }
    }
}while (0);
```



```
    HI_UNF_SCI_Close(0);  
    HI_UNF_SCI_DeInit();  
  
    return 0;  
}
```



# Contents

---

<b>17 Cipher.....</b>	<b>1</b>
17.1 Overview .....	1
17.2 Important Concepts .....	2
17.3 Features .....	2
17.4 Development Guide.....	2
17.4.1 Encrypting/Decrypting One Data Segment.....	3
17.4.2 Encrypting/Decrypting Multiple Data Segments .....	5
17.4.3 Calculating the Hash Value .....	6
17.4.4 Calculating the HMAC Value .....	6
17.4.5 Calculating the AES CBC-MAC Value.....	7
17.4.6 Obtaining Random Numbers.....	7
17.4.7 Encrypting/Decrypting Data by Using RSA .....	8
17.4.8 RSA Signing and Signature Verifying Process.....	9



## Figures

---

<b>Figure 17-1 Scenario 1 .....</b>	4
<b>Figure 17-2 Scenario 2 .....</b>	4



# 17 Cipher

## 17.1 Overview

The cipher module supports data encryption and decryption using the data encryption standard (DES)/3DES, advanced encryption standard (AES) algorithms, RSA encryption/decryption, and hash digest algorithms which comply with the FIPS46-3 and FIPS 197 standards. The DES/3DES and AES operating modes comply with the FIPS-81 and NIST special 800-38a standards.

- Supports the operating modes of electronic codebook (ECB), cipher block chaining (CBC), 1-/8-/128-cipher feedback (CFB), 128-output feedback (OFB), and counter (CTR) for the AES algorithm. These operating modes comply with the NIST special 800-38a standard.
- Supports the operating modes of ECB, CBC, 1-/8-/64-CFB, and 1-/8-/64-OFB for the DES or 3DES algorithm. These operating modes comply with the FIPS-81 standard.
- Encrypts and decrypts one or more blocks at a time in ECB, CBC, CFB, or OFB operating mode.
- Encrypts and decrypts one block at a time in CTR operating mode using the AES algorithm.
- Transmits data in DMA mode in the cipher. When the cipher encryption and decryption APIs are used, the input or output address parameter must be the physical address.
- SHA1 and SHA256 supported by the hash and multiple channels for software
- Encryption using public keys and decryption using private keys, encryption using private keys and decryption using public keys, signing, and verification supported by the RSA. Data filling methods in different modes comply with PKCS#1 standards.
- NO\_PADDING, BLOCK\_YTPE\_0, BLOCK\_YTPE\_1, BLOCK\_YTPE\_2, RSAES\_OAEP\_SHA1, RSAES\_OAEP\_SHA256, and RSAES\_PKCS1\_V1\_5 modes for RSA encryption/decryption
- RSASSA\_PKCS1\_V15\_SHA1, RSASSA\_PKCS1\_V15\_SHA256, RSASSA\_PKCS1\_PSS\_SHA1, and RSASSA\_PKCS1\_PSS\_SHA256 modes for RSA signing and verification

### NOTE

In block cryptography, data to be encrypted/decrypted is divided into several blocks by using one of the following operating modes: ECB, CBC, CFB, OFB, and CTR. In ECB mode, each block is separately encrypted/decrypted and independent of each other. In non-ECB mode, blocks are dependent on each other, and the initial vector (IV) is used in the first block to ensure the uniqueness of each message. For details about the algorithms, see the related documents.



## 17.2 Important Concepts

None

## 17.3 Features

The cipher module provides the following functions:

- Encrypts/Decrypts one data segment. The following APIs are provided:
  - HI\_UNF\_CIPHER\_Encrypt: Encrypts one data segment.
  - HI\_UNF\_CIPHER\_Decrypt: Decrypts one data segment.
- Encrypts/Decrypts multiple data segments. The following APIs are provided:
  - HI\_UNF\_CIPHER\_EncryptMulti: Encrypts multiple data segments.
  - HI\_UNF\_CIPHER\_DecryptMulti: Decrypts multiple data segments.
- Calculates the hash value. The following APIs are provided:
  - HI\_UNF\_CIPHER\_HashInit: Initializes the hash module.
  - HI\_UNF\_CIPHER\_HashUpdate: Inputs data for hash calculation.
  - HI\_UNF\_CIPHER\_HashFinal: Outputs the calculated hash value.
- Calculates the hash message authentication code (HMAC) value. The following API is provided:
  - HI\_UNF\_CIPHER\_HashInit: Initializes the hash module.
  - HI\_UNF\_CIPHER\_HashUpdate: Inputs data for HMAC calculation.
  - HI\_UNF\_CIPHER\_HashFinal: Outputs the calculated HMAC value.
- Calculates the AES CBC-MAC value. The following API is provided:  
HI\_UNF\_CIPHER\_CalcMAC: Calculates the AES CBC-MAC value for specified data.
- Obtains random numbers. The following API is provided:  
HI\_UNF\_CIPHER\_GetRandomNumber: Obtains random numbers.
- Calculates the RSA value. The following APIs are provided:
  - HI\_UNF\_CIPHER\_RsaPublicEncrypt: Encrypts the plain text using public keys.
  - HI\_UNF\_CIPHER\_RsaPrivateDecrypt: Decrypts the cipher text using private keys.
  - HI\_UNF\_CIPHER\_RsaPrivateEncrypt: Encrypts the plain text using private keys.
  - HI\_UNF\_CIPHER\_RsaPublicDecrypt: Decrypts the cipher text using public keys.
  - HI\_UNF\_CIPHER\_RsaSign: Signs the user data using private keys.
  - HI\_UNF\_CIPHER\_RsaVerify: Verifies the validity and integrity of user data using public keys.

## 17.4 Development Guide

The cipher module is used in the following scenarios:

- Encrypting/Decrypting one data segment
- Encrypting/Decrypting multiple data segments
- Calculating the hash value



- Calculating the HMAC value
- Calculating the AES CBC-MAC value
- Obtaining random numbers
- Encrypting/Decrypting data using the RSA
- Signing and verifying data using the RSA

## 17.4.1 Encrypting/Decrypting One Data Segment

### Scenario

When a data segment in the physical memory needs to be encrypted/decrypted, you can obtain the physical address of the data segment and encrypt/decrypt it by calling the API of the cipher module at the user layer.

### Working Process

To encrypt a data segment in the physical memory, perform the following steps:

- Step 1** Initialize the cipher module by calling HI\_UNF\_CIPHER\_Init.
- Step 2** Obtain a cipher handle by calling HI\_UNF\_CIPHER\_CreateHandle.
- Step 3** Set encryption/decryption options by calling HI\_UNF\_CIPHER\_ConfigHandle.
- Step 4** Encrypt a data segment in the physical memory by calling HI\_UNF\_CIPHER\_Encrypt.
- Step 5** Release the cipher handle by calling HI\_UNF\_CIPHER\_DestroyHandle.
- Step 6** Deinitialize the cipher module by calling HI\_UNF\_CIPHER\_DeInit.

**----End**

### Notes

Note the following when using the cipher module:

- You must obtain a cipher handle before encryption or decryption. Release the handle if it is not used for a long time. It is recommended that you obtain a handle for encryption and a handle for decryption. Each handle is used only for encryption or decryption.
- You can encrypt/decrypt only data in a consecutive physical memory. You can obtain a physical memory by calling HI\_MMZ\_New, and then map the virtual address of the physical memory by calling HI\_MMZ\_Map.
- The input address parameter must be the physical address for the data when you encrypt/decrypt data by calling HI\_UNF\_CIPHER\_Encrypt or HI\_UNF\_CIPHER\_Decrypt.
- The source address and destination address for encryption and decryption can be the same, that is, data can be encrypted and decrypted at the same address (the same buffer is used for storing both the ciphertext and plaintext).
- During cipher encryption/decryption in non-ECB mode, the IV must be used.
- The IV needs to be configured in the following two scenarios (taking data block decryption as an example):

[Scenario 1]

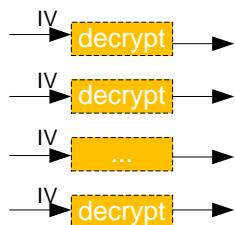


The IV needs to be updated each time the cipher module is called. In this case, set **stChangeFlags.bit1IV** to **1**, and properly configure the IV value.

For details about the API calling sequence, see the following:

```
HI_UNF_CIPHER_ConfigHandle() //should set stChangeFlags.bit1IV = 1  
and update u32IV  
HI_UNF_CIPHER_Decrypt()  
HI_UNF_CIPHER_ConfigHandle() //should set stChangeFlags.bit1IV = 1  
and update u32IV  
HI_UNF_CIPHER_Decrypt()  
...  
HI_UNF_CIPHER_ConfigHandle() //should set stChangeFlags.bit1IV = 1  
and update u32IV  
HI_UNF_CIPHER_Decrypt()
```

**Figure 17-1** Scenario 1



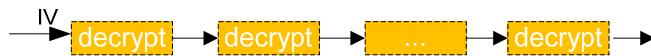
[Scenario 2]

The initial vector IV needs to be updated only when the cipher module is called for the first time. In this case, set **stChangeFlags.bit1IV** to **1**, and properly configure the IV value.

For details about the API calling sequence, see the following:

```
HI_UNF_CIPHER_ConfigHandle() //should set stChangeFlags.bit1IV = 1 and  
update u32IV  
HI_UNF_CIPHER_Decrypt() HI_UNF_CIPHER_Decrypt()...  
HI_UNF_CIPHER_Decrypt()
```

**Figure 17-2** Scenario 2



The IV value must be configured based on the actual scenario.

- If the member **bKeyByCA** of the data structure **HI\_UNF\_CIPHER\_CTRL\_S** is set to **HI\_FALSE**, the normal mode is used, indicating that the key needs to be manually configured for data encryption/decryption. For example:

```
memcpy(CipherCtrl.u32Key, u8KeyBuf, 32);
```



For details, see the samples of the cipher module. If bKeyByCA is set to **HI\_TRUE**, the embedded key in the chip is used for data encryption/decryption. For details, see the related documents for the advanced secure CA chips and anti-copy applications of common chips.

## Sample

See `sample_cipher.c` in the SDK.

### 17.4.2 Encrypting/Decrypting Multiple Data Segments

#### Scenario

When multiple data segments in the physical memory need to be encrypted/decrypted, you can obtain the physical addresses of the data segments and encrypt/decrypt them by calling the API of the cipher module at the user layer.

When multiple data segments are encrypted/decrypted, each data segment is calculated by using the vector configured by `HI_UNF_CIPHER_ConfigHandle`. Each data segment is calculated independently. That is, the calculation result of the previous data segment is not used in the calculation of the next data segment, and the calculation result of the previous function invocation does not affect that of the next function invocation.

#### Working Process

To encrypt three data segments in the physical memory, perform the following steps:

- Step 1** Initialize the cipher module by calling `HI_UNF_CIPHER_Init`.
- Step 2** Obtain a cipher handle by calling `HI_UNF_CIPHER_CreateHandle`.
- Step 3** Set encryption/decryption options by calling `HI_UNF_CIPHER_ConfigHandle`.
- Step 4** Encrypt multiple data segments in the physical memory by calling `HI_UNF_CIPHER_EncryptMulti`.
- Step 5** Release the cipher handle by calling `HI_UNF_CIPHER_DestroyHandle`.
- Step 6** Deinitialize the cipher module by calling `HI_UNF_CIPHER_DeInit`.

----End

#### Notes

- See the notes in section " [Signing and verifying data using the RSA](#) ."

## Sample

See `sample_multicipher.c` in the SDK.



## 17.4.3 Calculating the Hash Value

### Scenario

The SHA1 or SHA256 algorithm can be used to calculate the hash value of data.

### Working Process

To calculate the hash value, perform the following steps:

- Step 1** Initialize the cipher module by calling HI\_UNF\_CIPHER\_Init.
- Step 2** Select the hash algorithm and initialize the hash module by calling HI\_UNF\_CIPHER\_HashInit.
- Step 3** Input data by calling HI\_UNF\_CIPHER\_HashUpdate. The data can be calculated by block.
- Step 4** Complete the input and output the hash value by calling HI\_UNF\_CIPHER\_HashFinal.
- Step 5** Deinitialize the cipher module by calling HI\_UNF\_CIPHER\_DeInit.

----End

### Notes

This working process supports multiple software channels. Multiple hash calculation tasks can be started at the same time. That is, when a hash calculation task is started in step 2 and has not yet been completed (that is, before step 4 is executed), a new channel can be applied to start another hash calculation task until there is no channel available.

At most eight hash software channels are supported. Eight channels can be enabled at the same time. However, only one channel is implementing the calculation at the same time.

### Sample

See sample\_hash.c in the SDK.

## 17.4.4 Calculating the HMAC Value

### Scenario

The SHA1 or SHA256 algorithm can be used to calculate the HMAC value of data.

### Working Process

To calculate the HMAC value, perform the following steps:

- Step 1** Initialize the cipher module by calling HI\_UNF\_CIPHER\_Init.
- Step 2** Select the hash algorithm, set the key for HMAC calculation, and initialize the hash module by calling HI\_UNF\_CIPHER\_HashInit.
- Step 3** Input data by calling HI\_UNF\_CIPHER\_HashUpdate. The data can be input by block.
- Step 4** Complete the input and output the HMAC value by calling HI\_UNF\_CIPHER\_HashFinal.



**Step 5** Deinitialize the cipher module by calling HI\_UNF\_CIPHER\_DeInit.

----End

## Notes

None

## Sample

See `sample_hash.c` in the SDK.

## 17.4.5 Calculating the AES CBC-MAC Value

### Scenario

The AES CBC-MAC value needs to be calculated. For details about the algorithm, see the *rfc4493. The AES-CMAC Algorithm*.

### Working Process

To calculate the AES CBC-MAC value, perform the following steps:

**Step 1** Initialize the cipher module by calling HI\_UNF\_CIPHER\_Init.

**Step 2** Obtain a cipher handle by calling HI\_UNF\_CIPHER\_CreateHandle.

**Step 3** Set cipher parameters by calling HI\_UNF\_CIPHER\_ConfigHandle.

**Step 4** Calculate the CBC-MAC value by calling HI\_UNF\_CIPHER\_CalcMAC.

**Step 5** Deinitialize the cipher module by calling HI\_UNF\_CIPHER\_DeInit.

----End

## Notes

This working process supports multiple software channels. Multiple HMAC calculation tasks can be started at the same time. That is, when an HMAC calculation task is started in step 2 and has not yet been completed (that is, before step 4 is executed), a new channel can be applied to start another HMAC calculation task until there is no channel available.

HMAC and hash calculation tasks share eight software channels. Eight channels can be enabled at the same time. However, only one channel is implementing the calculation at the same time.

## Sample

See `sample_cbcmac.c` in the SDK.

## 17.4.6 Obtaining Random Numbers

### Scenario

To obtain the true random numbers generated by the hardware, perform the following steps:



- Step 1** Initialize the cipher module by calling HI\_UNF\_CIPHER\_Init.
- Step 2** Obtain 32-bit random numbers by calling HI\_UNF\_CIPHER\_GetRandomNumber.
- Step 3** Deinitialize the cipher module by calling HI\_UNF\_CIPHER\_DeInit.
- End

## Notes

None

## Sample

See `sample_rng.c` in the SDK.

## 17.4.7 Encrypting/Decrypting Data by Using RSA

### Scenario

Encrypt/Decrypt data by using the RSA algorithm. For details about the algorithm, see the *rfc3447. RSA Cryptography Specifications*.

### Working Process

To encrypt or decrypt data by using the RSA asymmetric algorithm, perform the following steps:

- Step 1** Initialize the cipher device by calling HI\_UNF\_CIPHER\_Init.
- Step 2** Encrypt/Decrypt the data using the RSA algorithm by calling one of the following APIs based on the used key:
- HI\_UNF\_CIPHER\_RsaPublicEncrypt for encryption using the public key
  - HI\_UNF\_CIPHER\_RsaPrivateDecrypt for decryption using the private key
  - HI\_UNF\_CIPHER\_RsaPrivateEncrypt for encryption using the private key
  - HI\_UNF\_CIPHER\_RsaPublicDecrypt for decryption using the public key
- Step 3** Stop a cipher device by calling HI\_UNF\_CIPHER\_Deinit.
- End

## Notes

The bit width for the RSA key can be 1024, 2048, or 4096 bits. According to the rule of the RSA algorithm, the values of the plain text and cipher text must be smaller than that of the public key  $N$ . Therefore, the length of the data to be encrypted/decrypted must be less than or equal to the length of the key. Typically, 0 is added to the upper bit of the data to be encrypted/decrypted so that its length becomes equal to that of the public key  $N$  but its value is smaller than  $N$ . The PKCS#1 standard defines the data stuffing methods, which are Block Type 0, Block Type 1, Block Type 2, RSAES-OAEP, and RSAES-PKCS1-v1\_5.

The RSA is an asymmetric algorithm. That is, when the public key is used to encrypt data, the private key must be used to decrypt the data, and vice versa.



## Sample

For details about the sample, see **sample\_rsa\_enc.c** in the SDK.

## 17.4.8 RSA Signing and Signature Verifying Process

### Scenario

Sign the data and verify the signature by using the RSA algorithm. For details about the algorithm, see the *rfc3447. RSA Cryptography Specifications*.

### Working Process

To sign the data using the RSA asymmetric algorithm or verify the signature, perform the following steps:

- Step 1** Initialize the cipher device by calling `HI_UNF_CIPHER_Init`.
- Step 2** Sign the data or verify the signature by calling the following interfaces:
  - `HI_UNF_CIPHER_RsaSign` for signing with a private key
  - `HI_UNF_CIPHER_RsaVerify` for signature verification using a public key
- Step 3** Stop a cipher device by calling `HI_UNF_CIPHER_Deinit`.

----End

### Notes

The bit width for the RSA key can be 1024, 2048, and 4096 bits. According to the rule of the RSA algorithm, the values of the plain text and cipher text must be smaller than that of the public key  $N$ . Therefore, the length of the data to be encrypted/decrypted must be less than or equal to the length of the key. Typically, calculate the HASH value of the data to be encrypted/decrypted, stuff the HASH value to the data equal to the public key  $N$  in length but smaller than  $N$  in value, and then encrypt the data. The PKCS#1 standard defines two data stuffing methods, which are RSASSA-PSS and RSAES-PKCS1-v1\_5.

The RSA is an asymmetric algorithm. It signs the data using the private key and verifies the signature using the public key.

## Sample

For details about the sample, see **sample\_rsa\_sign.c** in the SDK.



# Contents

---

<b>18 WDG .....</b>	<b>1</b>
18.1 Overview .....	1
18.2 Important Concepts .....	1
18.3 Features .....	1
18.4 Development Guide.....	1
18.4.1 Clearing the WDG Reset Signal .....	1
18.4.2 Resetting the System.....	4



## Figures

<b>Figure 18-1</b> Process for clearing the WDG reset signal .....	2
<b>Figure 18-2</b> Process for resetting the system.....	4



# 18 WDG

## 18.1 Overview

The watchdog (WDG) module resets the system when an exception occurs. By default, the WDG module generates a system reset signal periodically. You can deassert the WDG reset or disable the WDG if you do not want to reset the system.

## 18.2 Important Concepts

None

## 18.3 Features

The WDG module provides the following APIs:

- `HI_UNF_WDG_Reset`: Resets the system.
- `HI_UNF_WDG_Enable`: Enables the WDG.
- `HI_UNF_WDG_Disable`: Disables the WDG.
- `HI_UNF_WDG_SetTimeout`: Sets the timeout period.
- `HI_UNF_WDG_GetTimeout`: Obtains the timeout period.
- `HI_UNF_WDG_Clear`: Clears the WDG reset signal.

## 18.4 Development Guide

### 18.4.1 Clearing the WDG Reset Signal

#### Scenario

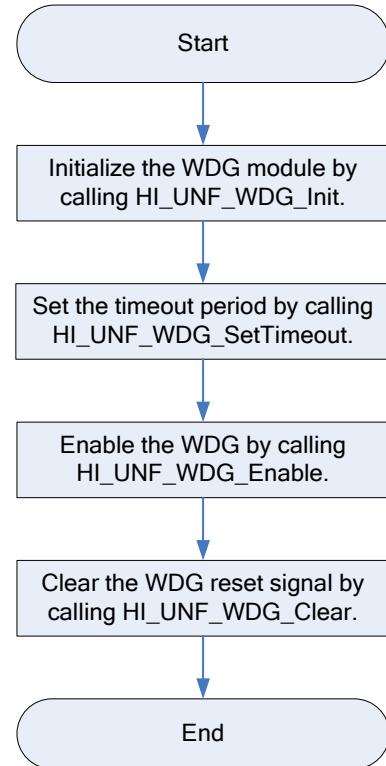
If the WDG reset signal is cleared within the configured timeout period, the system runs properly. Otherwise, the WDG resets the system when the configured timeout period is due.



## Working Process

Figure 18-1 shows the process for clearing the WDG reset signal.

**Figure 18-1** Process for clearing the WDG reset signal



## Notes

- Initialize the WDG module by calling the initialization API before performing any other operations.
- The interval of feeding the WDG ranges from 1,000 ms to 356,000 ms.
- The timeout needs to be set to an integral multiple of 1000 ms (1s). If the timeout is not an integral multiple of 1000 ms (1s), the WDG driver automatically rounds the timeout period up to an integral multiple of 1000 ms (1s). For example, if you set the timeout to 1,020 ms, the WDG driver rounds the timeout period up to 2s.
- The default interval of feeding the WDG is 60,000 ms.
- After the WDG is enabled, the WDG driver does not automatically deassert the WDG reset. In this case, upper-layer applications need to deassert the WDG reset periodically at the interval for .
- If the WDG reset is not deasserted in time, the WDG transmits a reset signal through the WDG signal output pin to reset the system. Note that the WDG signal needs to be connected to the system reset pin when this function is used.

## Sample

The reference code is as follows:



```
#include <stdio.h>
#include "hi_unf_wdg.h"
#define WDG_NO    (0)

int main(void)
{
    int s32Status;
    unsigned int u32Time;
    int u32Loop;

    u32Time = 19000;
    /*Initialize the WDG.*/
    s32Status = HI_UNF_WDG_Init ();
    if (HI_SUCCESS != s32Status)
    {
        printf("open WDG device failure\n");
        return -1;
    }

    /*Set the timeout interval.*/
    s32Status = HI_UNF_WDG_SetTimeout(WDG_NO, u32Time);
    if (HI_SUCCESS != s32Status)
    {
        printf("set wdg time failure\n");
        HI_UNF_WDG_DeInit ();
        return -1;
    }

    /*Enable the WDG.*/
    s32Status = HI_UNF_WDG_Enable(WDG_NO);
    if (HI_SUCCESS != s32Status)
    {
        printf("enable wdg failure\n");
        HI_UNF_WDG_DeInit ();
        return -1;
    }

    for (u32Loop = 0; u32Loop < 5; u32Loop++)
    {
        /* Delay a period of time.*/
        sleep(10);
        s32Status = HI_UNF_WDG_Clear (WDG_NO);
        if (HI_SUCCESS != s32Status)
        {
```



```
        printf("clear wdg failure\n");
        HI_UNF_WDG_DeInit ();
        return -1;
    }
}

HI_UNF_WDG_DeInit ();
return 0;
}
```

## 18.4.2 Resetting the System

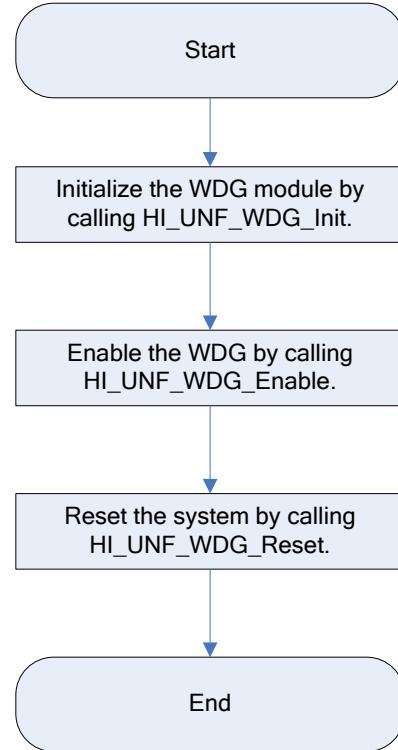
### Scenario

The system needs to be reset in some scenarios, for example, when system upgrade is detected.

### Working Process

[Figure 18-2](#) shows the process for resetting the system.

**Figure 18-2** Process for resetting the system



### Notes

You must enable the WDG by calling HI\_UNF\_WDG\_Enable before resetting the system by calling HI\_UNF\_WDG\_Reset.



## Sample

The reference code is as follows:

```
#include <stdio.h>
#include "hi_unf_wdg.h"
#define WDG_NO (0)

int main(void)
{
    int s32Status;

    /*Initialize the WDG.*/
    s32Status = HI_UNF_WDG_Init ();
    if (HI_SUCCESS != s32Status)
    {
        printf("open WDG device failure\n");
        return -1;
    }

    /*Set the WDG timeout.*/
    u32Value = 6000;      //ms
    s32Ret = HI_UNF_WDG_SetTimeout(WDG_NO, u32Value);
    if (HI_SUCCESS != s32Ret)
    {
        printf("WDG SetTimeout failure \n");
        HI_UNF_WDG_DeInit();
        return -1;
    }

    /*Enable the WDG.*/
    s32Status = HI_UNF_WDG_Enable(WDG_NO);
    if (HI_SUCCESS != s32Status)
    {
        printf("enable wdg failure\n");
        HI_UNF_WDG_DeInit();
        return -1;
    }
    s32Status = HI_UNF_WDG_Reset(WDG_NO);
    HI_UNF_WDG_DeInit ();
    return 0;
}
```



# Contents

---

<b>21 PMOC .....</b>	<b>1</b>
21.1 Overview .....	1
21.3 Features .....	2
21.4 Development Guide.....	2
21.4.1 Standard Standby .....	2
21.4.2 USB Mouse and Keyboard Wakeup.....	4
21.4.3 Network Wakeup.....	4
21.4.4 System Restart After Wakeup .....	5
21.4.5 Intelligent Standby .....	6
21.4.6 Development of Running Code for the Standby Mode .....	7



# Figures

---

**Figure 21-1** Position of the PMOC module in the system ..... 1

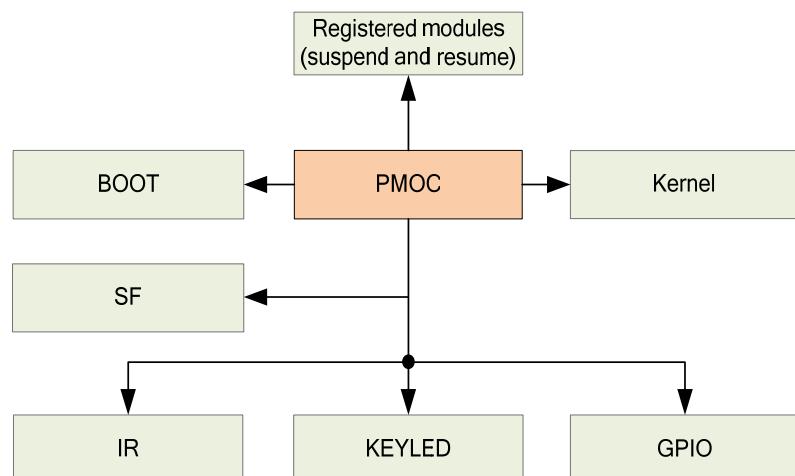


# 21 PMOC

## 21.1 Overview

The PMOC module manages power consumption of the chip. It enables the system to enter the standby mode or wakes up the system from the standby mode. [Figure 21-1](#) shows the position of the PMOC module in the system.

**Figure 21-1** Position of the PMOC module in the system



The PMOC module relies on the IR, KEYLED, and GPIO modules, SF, BOOT, kernel, and application modules that are registered with the kernel, and provides the standby wakeup function for upper-layer applications.

## 21.2 Important Concepts

[Standby mode]

The standby mode is a low-power waiting state of the system. In this mode, most unused modules are stopped, and parameters of running programs are stored in the memory.

[Wakeup]



The system resumes to normal mode from the standby mode when certain conditions are met.

## 21.3 Features

The PMOC module implements the standby and wakeup functions of the system. The following APIs are provided:

- `HI_UNF_PMOC_Init`: Initializes the PMOC module.
- `HI_UNF_PMOC_DeInit`: Deinitializes the PMOC module.
- `HI_UNF_PMOC_SwitchSystemMode`: Sets the mode to be switched to the standby mode and obtains the wakeup method. It is a block interface.
- `HI_UNF_PMOC_SetStandbyDispMode`: Sets the displayed contents in standby mode.
- `HI_UNF_PMOC_ReadSystemMode`: Obtains the current system mode.
- `HI_UNF_PMOC_SetScene`: Sets the operating scenario.
- `HI_UNF_PMOC_SetDevType`: Sets the types of the front panels and remote controls used in standby mode.
- `HI_UNF_PMOC_SetWakeUpAttr`: Sets the standby wakeup attributes.
- `HI_UNF_PMOC_GetWakeUpAttr`: Obtains the configured standby wakeup parameters.
- `HI_UNF_PMOC_SetPwrOffGpio`: Sets the level of some GPIO pins, which are used to separately control some power supplies to reduce power consumption in standby mode.(the implementation of this interface depends on board hardware)
- `HI_UNF_PMOC_GetStandbyPeriod`: Obtains the duration of the standby state after wakeup.
- `HI_UNF_PMOC_EnterSmartStandby`: Enters the intelligent standby mode.
- `HI_UNF_PMOC_QuitSmartStandby`: Exits the intelligent standby mode.
- `HI_UNF_PMOC_GetWakeUpMode`: Obtains the parameters that trigger standby.

## 21.4 Development Guide

The PMOC module is used in the following scenarios:

- Standard standby
- USB mouse and keyboard wakeup
- Network wakeup
- System restart after wakeup
- Intelligent standby
- Development of running code for the standby mode

### 21.4.1 Standard Standby

#### Scenario

In the standard standby scenario, most modules are powered off, and the system enters deep sleep and responds only to the KEYLED (GPIO), IR, and timer interrupts. The power consumption is minimized in this scenario.



## Working Process

The working process for the standard standby scenario is as follows:

1. Exit or pause applications, and close all applications that access the DDR.
2. Initialize the PMOC module by calling HI\_UNF\_PMOC\_Init.
3. Set the types of the remote controls and front panels supported by the standby mode by calling HI\_UNF\_PMOC\_SetDevType.
4. Set the standby scenario to the standard standby scenario by calling HI\_UNF\_PMOC\_SetScene.
5. Set the wakeup conditions by calling HI\_UNF\_PMOC\_SetWakeUpAttr. The conditions include the following:
  - Wakeup wait time
  - Key value for wakeup over the infrared device
  - Key value for wakeup through the front panel (or GPIO interface ID, supporting only the GPIO group that is not powered off in standby mode. For details about the GPIO group IDs, see the corresponding chip data sheet.)
6. Set the contents displayed on the front panel by calling HI\_UNF\_PMOC\_SetStandbyDispMode.
  - Time
  - Program channel
  - No display
7. Enable the system to enter the standby mode by calling HI\_UNF\_PMOC\_SwitchSystemMode.
8. Wait for the KEYLED (GPIO), IR, or timer interrupt.
9. Deinitialize the PMOC module by calling HI\_UNF\_PMOC\_DeInit.
10. Start the applications again.

**----End**

## Notes

The PMOC driver code supports only one front panel. Because the HI\_UNF\_PMOC\_SetDevType cannot determine whether the front panel driver code is compiled into the running code, you must ensure that the required front panel types are compiled into the driver code before setting the front panel types by calling HI\_UNF\_PMOC\_SetDevType.

Note the following if the SATA hard drive or USB flash drive exist in the system during the wakeup from the standby mode:

- Uninstall the driver of the SATA controller or USB controller from the kernel before enabling the system to enter the standby mode. If there are mounted USB devices, unmount them first.
- Load the driver of the SATA controller or USB controller to the kernel after waking up the system but before starting applications. Then you can mount USB devices.



## Sample

See `sample_pmoc.c`.

### 21.4.2 USB Mouse and Keyboard Wakeup

#### Scenario

In the USB mouse and keyboard wakeup scenario, only some modules are stopped in standby mode, and modules such as the USB module work normally. The power consumption in this scenario is lower than that in the normal scenario, but higher than that in the standard standby scenario. The system is woken up when the left or right button of the USB mouse or a key on the USB keyboard is pressed.

#### Working Process

The working process for the USB mouse and keyboard wakeup scenario is as follows:

1. Initialize the PMOC module by calling `HI_UNF_PMOC_Init`.
2. Set the type of devices supported by the standby mode by calling `HI_UNF_PMOC_SetDevType`.
3. Set the wakeup conditions by calling `HI_UNF_PMOC_SetWakeUpAttr`. Set **bMouseKeyboardEnable** to **1** to enable the USB mouse and keyboard wakeup.
4. Set the contents displayed on the front panel by calling `HI_UNF_PMOC_SetStandbyDispMode`.
5. Enable the system to enter the standby mode by calling `HI_UNF_PMOC_SwitchSystemMode`.
6. Wait for the USB interrupt to wake up the system.
7. Deinitialize the PMOC module by calling `HI_UNF_PMOC_DeInit`.

----End

#### Notes

Ensure that the connected USB device supports standby wakeup. Otherwise, a failure code is returned when you set the standby wakeup conditions, and the system cannot enter the standby mode.

## Sample

See `sample_pmoc.c`.

### 21.4.3 Network Wakeup

#### Scenario

In the network forwarding standby scenario, only some modules are disabled, and modules such as the network module work in low-power mode. In this scenario, the system is woken up when a specific network packet is received over any of the configured network ports. The specific packet can be a unicast packet, a magic packet, or a wakeup frame.



## Working Process

The working process for the network wakeup scenario is as follows:

1. Initialize the PMOC module by calling HI\_UNF\_PMOC\_Init.
2. Set the type of devices supported by the standby mode by calling HI\_UNF\_PMOC\_SetDevType.
3. Set the wakeup conditions by calling HI\_UNF\_PMOC\_SetWakeUpAttr. The conditions include the following:
  - a. ID of the port for network wakeup
  - b. Type of the network wakeup packet
  - c. Filtering parameters of the wakeup frame (only for wakeup frame wakeup)
4. Set the contents displayed on the front panel by calling HI\_UNF\_PMOC\_SetStandbyDispMode.
5. Enable the system to enter the standby mode by calling HI\_UNF\_PMOC\_SwitchSystemMode.
6. Wait for the network interrupt to wake up the system.
7. Deinitialize the PMOC module by calling HI\_UNF\_PMOC\_DeInit.

## Notes

You can set multiple network ports to receive the network wakeup packet.

Typically unicast packet wakeup is not recommended, because the system is woken up by a common unicast packet sent from the peer end as long as the peer end has recorded the local MAC address.

## Sample

See `sample_pmoc.c`.

### 21.4.4 System Restart After Wakeup

#### Scenario

In the system restart after wakeup scenario, when the system is woken up in either of the preceding two modes, the entire system is restarted.

#### Working Processes

The working process for the system restart after wakeup is as follows:

1. Initialize the PMOC module by calling HI\_UNF\_PMOC\_Init.
2. Set the type of devices supported by the standby mode by calling HI\_UNF\_PMOC\_SetDevType.
3. Set the wakeup conditions (such as the keyboard value of the remote control or network wakeup packet) by calling HI\_UNF\_PMOC\_SetWakeUpAttr, and set **bResumeResetEnable** to **1** to enable restart wakeup.



4. Set the contents displayed on the front panel by calling HI\_UNF\_PMOC\_SetStandbyDispMode.
5. Enable the system to enter the standby mode by calling HI\_UNF\_PMOC\_SwitchSystemMode.
6. Wait the configured wakeup interrupt to wake the system up.
7. The system restarts after being woken up.

## Notes

The mode of system restart after wakeup can work with any of the preceding wakeup source (remote control, front panel, or network) and focuses on the behavior of the system after wakeup.

## Sample

See `sample_pmoc.c`.

### 21.4.5 Intelligent Standby

#### Scenario

The intelligent standby is a light standby mode. In this standby scenario, the kernel does not enter the standby mode but works in normal mode, and you can specify some modules to be disabled or enter the low-power state.

#### Working Processes

The working process for the intelligent standby is as follows:

1. The program or service running at the upper layer exits or stops the current service.
2. The control program calls HI\_UNF\_PMOC\_EnterSmartStandby to disable the specified modules in the order of module deinitialization.
3. The control program detects whether the configured wakeup conditions are met.
4. If the conditions are met, the control program calls HI\_UNF\_PMOC\_QuitSmartStandby to enable the modules disabled in step 2 zz in the order of module initialization.
5. The control program restarts or resumes the service that is stopped before the system enters the intelligent standby mode.

## Notes

The modules disabled in intelligent standby mode are indicated by the parameters set in HI\_UNF\_PMOC\_EnterSmartStandby. Each bit indicates one module. The bit value 0 indicates that the module is disabled and the bit value 1 indicates that the module is not disabled.

## Sample

None



## 21.4.6 Development of Running Code for the Standby Mode

### Scenario

The running code for the standby mode is a program that separately runs on the low-power chip (LPC). It starts running after the Linux system enters hibernation. It displays time, program channel, or other characters on the LED, responds to the IR, KEYLED, and timer interrupt, wakes up the master chip and exits programs, and enters the Linux system wakeup program. If new standby devices such as the IR or KEYLED device need to be developed, and the code must be modified.

### Working Process

For details, see section "19.4 Development Guide."

### Notes

The development of the running code for the standby mode varies according to LPCs, but the basic code structure and development method are the same. For details, see section "19.4 Development Guide."

### Sample

None



# Contents

---

<b>19 MCU.....</b>	<b>1</b>
19.1 Overview .....	1
19.2 Important Concepts .....	1
19.3 Features .....	1
19.4 Development Guide.....	2
19.4.1 Modifying the Front Panel Type Supported by the MCU Module .....	2



# Figures

**Figure 19-1** Modifying USR\_BIN\_SWITCH ..... 3



# 19 MCU

## 19.1 Overview

The micro control unit (MCU) module provides the standby and wakeup functions in low-power mode for the PMOC module.

- The LED displays the time, channel, or other characters in standby mode.
- The following wakeup modes are supported:
  - Wakeup by pressing keys on the front panel (KEYLED)
  - Wakeup by using the IR remote control
  - Wakeup by using the timer
  - Wakeup over the network
  - Wakeup by using the USB mouse or keyboard

The MCU module is located within the PMOC module.

## 19.2 Important Concepts

[MCU]

The MCU integrates the CPU, RAM, ROM, timer, and multiple I/O interfaces in a single chip, providing various control combinations for different application scenarios.

## 19.3 Features

The MCU allows the LED to work when the system is in standby mode and enables the system to be woken up by responding to the interrupt in standby mode. No API is provided. The MCU internally specifies the content in the RAM to exchange parameters with the PROC module.

### Code Structure

The functions for controlling streams include the main functions and interrupt service routine (ISR) interrupt handling function.



The code of the MCU includes the following:

- Code related to the standby process:
  - do\_suspend: Switches the current mode to the low-power mode, configures the memory, and powers off the module.
  - do\_resume: Powers on the module in the power-off area, deasserts CPU reset.
- Code related to the KEYLED module:
  - keyled\_pt6961: Implements the display of the LED on the PT6961 front panel and handles keys in low-power mode.
  - keyled\_pt6964: Implements the display of the LED on the PT6964 front panel and handles keys in low-power mode.
  - keyled\_ct1642: Implements the display of the LED on the CT1642 front panel and handles keys in low-power mode.
  - keyled\_ct1642\_inner: Implements the display of the LED on the CT1642 front panel and handles keys in low-power mode by hardware logic.
  - keyled\_fd650: Implements the display of the LED on the FD650 front panel and handles keys in low-power mode.
- Code related to the IR module: ir\_std (supports the NEC, TC9012, and Sony protocols) and ir\_raw (supports the NEC, NEC resembling, RC5, and RC6 protocols).
- Code related to the timer module: Counts the time and wakes up the system in a scheduled manner.
- Code related to the debugging (DBG) module: Provides debugging APIs for the low-power mode, including the APIs for displaying ASCII characters on a terminal over a serial port, storing debugging information in the MCU RAM, and displaying debugging information on the front panel LED.

## Capability Set

The supported IR protocols and front panels are as follows:

- IR protocols: NEC simple, NEC full, TC9012, Sony, RC5, and RC6
- Front panel drivers: PT6961, PT6964, CT1642, and FD650

## 19.4 Development Guide

This section describes the common application scenario of the MCU module.

### 19.4.1 Modifying the Front Panel Type Supported by the MCU Module

#### Scenario

When the used front panel is different from the default front panel compiled in the SDK, you need to modify the MCU code, generate the running code, and put the code into the PMOC module so that the PMOC module can use the code in low-power mode.



## Working Process

To modify the front panels supported by the MCU and compile the code, perform the following steps:

- Step 1** Double-click the project file to open it in the C51 compiler.
- Step 2** Modify the head file **keyled.h** of the KEYLED module in the MCU code. Enable the macro of the front panel to be supported (you can enable only one at a time). For example, if you want to use PT6964, define the following macro:

```
#define KEYLED_PT6964
```

- Step 3** Compile the project. A target file **x5hdtest.hex** is generated in the current folder.
- Step 4** Copy the **\tools\windows\hex2char\hex2char.exe** file to the directory where the target file is located. Run **hex2char.exe** in the Windows command line interface (CLI) to generate **x5hdtest.bin** and **output.txt** by running the following command:

```
hex2char.exe x5hdtest.hex
```

- Step 5** Debug the program.

Save **x5hdtest.bin** in **sample\pmoc** and modify the value of **USR\_BIN\_SWITCH** in **sample\_pmoc.c** to **1**. The standby effect of **x5hdtest.bin** can be tested after recompilation. See [Figure 19-1](#).

**Figure 19-1** Modifying USR\_BIN\_SWITCH

```
#include <time.h>
#include "hi_unf_ecs.h"

/*
when you want use your x5hdtest.bin, please re-define USR_BIN_SWITCH to 1.
Also don't forget put x5hdtest.bin to this path: "SDK\sample\pmoc"
*/
#define USR_BIN_SWITCH 1//0

HI_S32 main(int argc, char *argv[])
{
    HI_S32 i;
    HI_S32 ret;
    HI_U32 tmp;
    time_t time0;
    struct tm *time1;
    HI_UNF_PMOC_MODE_E mode;
    HI_UNF_PMOC_WKUP_S wakeup = {0};
    HI_UNF_PMOC_DEV_TYPE_S devType = {0};
    HI_UNF_PMOC_ACTUAL_WKUP_E actual ;
    HI_UNF_PMOC_STANDBY_MODE_S standbyMode = {0};
#if USR_BIN_SWITCH
    FILE *fb;
    HI_U32 len;
    HI_U8 *ptr;
```

- Step 6** Compile and generate a .ko file.

In the SDK directory, run **make msp\_install** and then **make rootbox\_install; make fs**. Replace the rootfs of the board, or replace **hi\_pmoc.ko** on the board with the compiled **hi\_pmoc.ko** to update the MCU code.

----End



## Notes

The MCU code must be compiled on Windows. The C51 compiler is recommended.

The SDK directories and file names mentioned in the preceding steps may vary according to SDK versions.

## Sample

None



# Contents

---

<b>20 PVR .....</b>	<b>1</b>
20.1 Overview .....	1
20.2 Important Concepts .....	2
20.3 Features .....	2
20.4 Development Guide.....	3
20.4.1 Recording Streams .....	4
20.4.2 Playing Streams.....	6
20.4.3 Time-Shift Playing .....	8



# Figures

**Figure 20-1 Position of the PVR module in the system ..... 1**



# 20 PVR

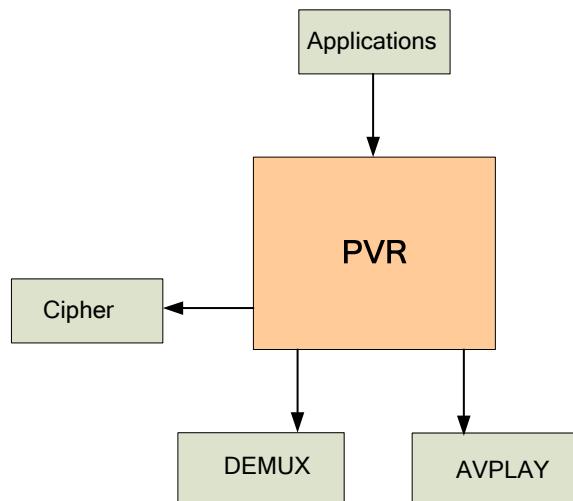
## 20.1 Overview

The PVR module provides the following functions:

- Records streams (a maximum of 7-channel simultaneous recording).
- Plays recorded streams (a maximum of 4-channel playback).
- Records and plays streams simultaneously (time-shift function).
- Supports event registration. This function enables you to register a callback function for the events (for example, the disk is full or the playback is complete) during recording and playback, and then process such events.

[Figure 20-1](#) shows the position of the PVR module in the system.

**Figure 20-1** Position of the PVR module in the system



Depending on the DEMUX, AVPLAY, and cipher modules, the PVR module provides superior recording and playback solutions for applications.



## 20.2 Important Concepts

[Rewind recording]

Rewind recording indicates that when the record file reaches the pre-configured size, or the recording duration reaches the pre-configured length, the PVR does not increase the file size any more, but overwrites the recorded data to ensure constant size or time of the recorded file.

[Time-shift playback]

Time-shift playback indicates that the program that is being recorded is being played at the same time.

[Trick playback]

Trick playback indicates the playback mode such as fast forward, rewind, and slow forward.

## 20.3 Features

The PVR module provides the following functions:

- Records live signals. The following APIs are provided:
  - HI\_UNF\_PVR\_RecInit: Initializes the PVR recording module.
  - HI\_UNF\_PVR\_RegisterEvent: Registers the callback function for the event to be processed (optional).
  - HI\_UNF\_PVR\_RecCreateChn: Requests a recording channel.
  - HI\_UNF\_PVR\_RecStartChn: Enables a recording channel.
  - HI\_UNF\_PVR\_RecPauseChn: Pauses a recording channel.
  - HI\_UNF\_PVR\_RecResumeChn: Resumes a recording channel
  - HI\_UNF\_PVR\_RecGetStatus: Obtains the status of the current recording channel.
  - HI\_UNF\_PVR\_SetUsrDataInfoByFileName: Sets user data.
  - HI\_UNF\_PVR\_GetUsrDataInfoByFileName: Obtains user data.
  - HI\_UNF\_PVR\_RecStopChn: Disables a recording channel.
  - HI\_UNF\_PVR\_RecDestroyChn: Releases a recording channel.
  - HI\_UNF\_PVR\_UnRegisterEvent: Deregister a callback function for an event.
  - HI\_UNF\_PVR\_RecDeInit: Deinitializes the PVR recording module.
- Plays recorded files. The following APIs are provided:
  - HI\_UNF\_PVR\_PlayInit: Initializes the PVR playback module.
  - HI\_UNF\_PVR\_RegisterEvent: Registers the callback function for the event to be processed (optional).
  - HI\_UNF\_PVR\_PlayCreateChn: Requests a playback channel.
  - HI\_UNF\_PVR\_PlayStartChn: Enables a playback channel.
  - HI\_UNF\_PVR\_PlayTrickMode: Starts playing in trick mode.
  - HI\_UNF\_PVR\_PlayPauseChn: Pauses a playback channel.
  - HI\_UNF\_PVR\_PlayResumeChn: Resumes a playback channel.
  - HI\_UNF\_PVR\_PlayStep: Starts playback by frame.
  - HI\_UNF\_PVR\_PlaySeek: Seeks a specified position to play.



- HI\_UNF\_PVR\_PlayGetStatus: Obtains the status of the current playback channel.
- HI\_UNF\_PVR\_PlayStopChn: Disables a playback channel.
- HI\_UNF\_PVR\_PlayDestroyChn: Releases a playback channel.
- HI\_UNF\_PVR\_UnRegisterEvent: Deregister a callback function for an event (optional).
- HI\_UNF\_PVR\_PlayDeInit: Deinitializes the PVR playback module.
- Plays the program that is being recorded. The following APIs are provided:
  - HI\_UNF\_PVR\_RecInit: Initializes the PVR recording module.
  - HI\_UNF\_PVR\_PlayInit: Initializes the PVR playback module.
  - HI\_UNF\_PVR\_RegisterEvent: Registers the callback function for the event to be processed (optional).
  - HI\_UNF\_PVR\_RecCreateChn: Requests a recording channel.
  - HI\_UNF\_PVR\_RecStartChn: Enables a recording channel.
  - HI\_UNF\_PVR\_PlayStartTimeShift: Enables a time-shift channel.
  - HI\_UNF\_PVR\_PlayCreateChn: Requests a playback channel.
  - HI\_UNF\_PVR\_PlayStartChn: Enables a playback channel.
  - HI\_UNF\_PVR\_PlayTrickMode: Starts playing in trick mode.
  - HI\_UNF\_PVR\_PlayPauseChn: Pauses a playback channel.
  - HI\_UNF\_PVR\_PlayResumeChn: Resumes a playback channel.
  - HI\_UNF\_PVR\_RecPauseChn: Pauses a recording channel.
  - HI\_UNF\_PVR\_RecResumeChn: Resumes a recording channel
  - HI\_UNF\_PVR\_PlayStep: Starts playback by frame.
  - HI\_UNF\_PVR\_PlaySeek: Seeks a specified position to play.
  - HI\_UNF\_PVR\_PlayGetStatus: Obtains the status of the current playback channel.
  - HI\_UNF\_PVR\_PlayStopTimeShift: Disables a time-shift channel.
  - HI\_UNF\_PVR\_PlayStopChn: Disables a playback channel.
  - HI\_UNF\_PVR\_PlayDestroyChn: Releases a playback channel.
  - HI\_UNF\_PVR\_RecStopChn: Disables a recording channel.
  - HI\_UNF\_PVR\_RecDestroyChn: Releases a recording channel.
  - HI\_UNF\_PVR\_UnRegisterEvent: Deregister a callback function for an event (optional).
  - HI\_UNF\_PVR\_PlayDeInit: Deinitializes the PVR playback module.
  - HI\_UNF\_PVR\_RecDeInit: Deinitializes the PVR recording module.

## 20.4 Development Guide

The PVR module is used in the following scenarios:

- Recording streams
- Playing streams
- Time-shift playing



## 20.4.1 Recording Streams

### Scenario

Streams are recorded but not played at the same time. You can record a program while watching another program.

### Working Process

To record a program, perform the following steps:

- Step 1** Initialize the DEMUX module, and bind a DEMUX to the port from which streams are received.
- Step 2** Initialize the PVR recording module by calling HI\_UNF\_PVR\_RecInit.
- Step 3** Create a PID channel by calling HI\_UNF\_DMX\_CreateChannel, set the PID, and then enable the channel. If the streams with multiple PIDs need to be recorded, you need to create multiple PID channels.
- Step 4** (Optional) Register a callback function for the events to be processed by calling HI\_UNF\_PVR\_RegisterEvent.
- Step 5** Create a recording channel by calling HI\_UNF\_PVR\_RecCreateChn. You need to set the attributes of the recording channel when creating it.
- Step 6** Enable the recording channel by calling HI\_UNF\_PVR\_RecStartChn. After this API is called, stream recording starts.
- Step 7** Disable the recording channel by calling HI\_UNF\_PVR\_RecStopChn. After this API is called, stream recording stops.
- Step 8** Release the recording channel by calling HI\_UNF\_PVR\_RecDestroyChn.
- Step 9** (Optional) Create another DEMUX channel to record streams with a different PID if required.
- Step 10** (Optional) Deregister a registered callback function by calling HI\_UNF\_PVR\_UnRegisterEvent.
- Step 11** Deinitialize the PVR recording module by calling HI\_UNF\_PVR\_RecDeInit.
- Step 12** Deinitialize the DEMUX module.

----End

### Notes

Note the following:

- Before recording streams, you need to configure the DEMUX module, bind DEMUXs to ports, create PID recording channels, set PIDs, and enable the PID recording channels. For details, see the development guide of the DEMUX module.
- Multiple recording channels are supported. The recording channels cannot be used for simultaneous audio/video recording and playback.
- If the name of a file that is being recorded is the same as the name of an existing file, the existing file is replaced with the new one.



- You are advised to name the recorded TS file **xxx.ts**. The file **xxx.ts.idx** is also generated during recording besides **xxx.ts**.
- If the size of the recorded file is greater than 4 GB, the file is divided into multiple files. The size of each file is less than 4 GB and the files are named **xxx.ts**, **xxx.ts.0001**, **xxx.ts.0002**, ..., **xxx.ts.000N** in sequence. However, there is only one index file **xxx.ts.idx**.
- If you want to record files of a fixed length, you need to set the maximum length of each file. If the recording mode is not the rewind recording mode, recording stops when the maximum length is reached.

Note the following during rewind recording:

- Before rewind recording, you must set the length or recording time period of each recorded file. If the specified length or time period is reached, the recorded streams are overwritten from the start of the file, and the index values in the index file are looped back and are overwritten from the start of the index file.
- The rewind information related to file recording is saved in the header of the index file. You can ignore this information and consider the streams to be sequential streams.

A space is reserved in the index file for saving the information required by users such as program names and audio/video PIDs of the recorded programs. Note the following:

- You can set the maximum size of the user information by setting the value of **u32UsrDataInfoSize**. After a recording channel is created, the value of **u32UsrDataInfoSize** cannot be changed, and the specified space in the index file is occupied.
- The user information can be set by calling **HI\_UNF\_PVR\_SetUsrDataInfoByFileName** only after a recording channel is created. Note that the size of the user information cannot be greater than that specified in the recording attributes.
- You can obtain the user information during playback by calling **HI\_UNF\_PVR\_GetUsrDataInfoByFileName**.

The PVR module allows events that occur during recording to be reported by calling callback functions. For details about the event type, see the *HMS API Development Reference*. Note the following during event registration:

- You can register and deregister an event by calling **HI\_UNF\_PVR\_RegisterEvent** and **HI\_UNF\_PVR\_UnRegisterEvent** respectively.
- You can register the same callback function or different callback functions for different events.
- You can transfer a parameter pointer when registering a callback function. After you call this function, the parameter pointer is returned.
- Event registration is irrelevant to channel IDs. However, after a callback function is called, the ID of the channel where the event occurs is transferred to the callback function.

Note the following when initializing the PVR module:

- Initialize the DEMUX module and AVPLAY module before initializing the PVR module, because most operations of the PVR module are based on these two modules.
- Initialize the PVR recording module before starting recording.
- Ensure that all the recording channels are released before deinitializing the PVR recording module. Otherwise, the deinitialization fails.



- Ensure that all PVR recording channels are stopped before being released. Otherwise, the channels fail to be released.

## Sample

See `sample\pvr\sample_pvr_rec.c`.

### 20.4.2 Playing Streams

#### Scenario

Recorded programs need to be played.

#### Working Process

To play a recorded program, perform the following steps:

- Step 1** Initialize the DEMUX module and AVPLAY module, and bind a DEMUX to the port from which the TSs to be played are received.
- Step 2** Create a player and a TS buffer.
- Step 3** Initialize the PVR playback module by calling `HI_UNF_PVR_PlayInit`.
- Step 4** (Optional) Register a callback function for the events to be processed by calling `HI_UNF_PVR_RegisterEvent`.
- Step 5** Create a playback channel by calling `HI_UNF_PVR_PlayCreateChn`. You need to set the attributes of the playback channel when creating it.
- Step 6** Enable the playback channel by calling `HI_UNF_PVR_PlayStartChn`. After this API is called, stream playback starts.
- Step 7** Play streams in different modes by calling the following APIs:
  - Call `HI_UNF_PVR_PlayTrickMode` to play streams in trick mode.
  - Call `HI_UNF_PVR_PlayPauseChn` to pause playback and call `HI_UNF_PVR_PlayResumeChn` to resume playback.
  - Call `HI_UNF_PVR_PlayStep` to play streams by frame.
  - Call `HI_UNF_PVR_PlaySeek` to seek streams for playback.
- Step 8** (Optional) Obtain the current status of the playback channel by calling `HI_UNF_PVR_PlayGetStatus`.
- Step 9** Disable the playback channel by calling `HI_UNF_PVR_PlayStopChn`. After this API is called, stream playback stops.
- Step 10** Release the playback channel by calling `HI_UNF_PVR_PlayDestroyChn`.
- Step 11** (Optional) Deregister the registered callback function by calling `HI_UNF_PVR_UnRegisterEvent`.
- Step 12** Deinitialize the PVR playback module by calling `HI_UNF_PVR_PlayDeInit`.
- Step 13** Deinitialize the DEMUX module and AVPLAY module.

----End



## Notes

Note the following when playing streams:

- Before playing streams, you need to configure the DEMUX module, bind DEMUXs to ports, create a player for PVR playback, and create a TS buffer for storing the streams to be played.
- Multiple playback channels are supported, but a playback channel cannot be used for PVR playback and live playback at the same time.
- The file that is being recorded can be played at the same time. This playback mode is time-shift playback.
- During playback, pause, trick playback, and step playback are supported. For trick playback, fast forward, slow forward, and rewind at 2, 4, 8, 16, 32, or 64 times the original speed are supported. However, slow backward is not supported. For step playback, step forward by frame but not step backward by frame is supported.
- During time-shift playback, an event is reported if the recording speed reaches the playback speed or vice versa.
- If a recorded file is played, an event is reported when the start or the end of the file is reached.

The PVR module allows events that occur during playback to be reported by calling callback functions. For details about the event type, see the *HMS API Development Reference*. Note the following during event registration:

- You can register and deregister an event by calling `HI_UNF_PVR_RegisterEvent` and `HI_UNF_PVR_UnRegisterEvent` respectively.
- You can register the same callback function or different callback functions for different events.
- You can transfer a parameter pointer when registering a callback function. After you call this function, the parameter pointer is returned.
- Event registration is irrelevant to channel IDs. However, after a callback function is called, the ID of the channel where the event occurs is transferred to the callback function.

Note the following when initializing the PVR module:

- Initialize the DEMUX module and AVPLAY module before initializing the PVR module, because most operations of the PVR module are based on these two modules.
- Initialize the PVR playback module before starting playback.
- Ensure that all the playback channels are released before deinitializing the PVR playback module. Otherwise, the deinitialization fails.
- Ensure that all PVR playback channels are stopped before being released. Otherwise, the channels cannot be released.

## Sample

See `sample\pvr\sample_pvr_play.c`.



## 20.4.3 Time-Shift Playing

### Scenario

A program needs to be played when being recorded.

### Working Process

To play a program that is being recorded, perform the following steps:

- Step 1** Initialize the DEMUX module and AVPLAY module, and bind a DEMUX to the port from which streams are received.
- Step 2** Initialize the PVR recording module by calling HI\_UNF\_PVR\_RecInit.
- Step 3** Initialize the PVR playback module by calling HI\_UNF\_PVR\_PlayInit.
- Step 4** (Optional) Register a callback function for the events to be processed by calling HI\_UNF\_PVR\_RegisterEvent.
- Step 5** Create a recording channel by calling HI\_UNF\_PVR\_RecCreateChn. You need to set the attributes of the recording channel when creating it.
- Step 6** Enable the recording channel by calling HI\_UNF\_PVR\_RecStartChn. After this API is called, stream recording starts.
- Step 7** Wait for some time.
- Step 8** Bind a DEMUX to the port from which TSs are received for playing, and create a player and a TS buffer.
- Step 9** Perform any of the following operations:
  - Enable the time-shift channel by calling HI\_UNF\_PVR\_PlayStartTimeShift. After this API is called, a playback channel is created and then enabled automatically.
  - Enable a playback channel by calling HI\_UNF\_PVR\_PlayCreateChn and then play the file that is being recorded by calling HI\_UNF\_PVR\_PlayStartChn using the same file name.
- Step 10** Play streams in different modes by calling the following APIs:
  - Call HI\_UNF\_PVR\_PlayTrickMode to play streams in trick mode.
  - Call HI\_UNF\_PVR\_PlayPauseChn to pause playback and call HI\_UNF\_PVR\_PlayResumeChn to resume playback.
  - Call HI\_UNF\_PVR\_PlayStep to play streams by frame.
  - Call HI\_UNF\_PVR\_PlaySeek to seek streams for playback.
- Step 11** (Optional) Obtain the current status of the time-shift channel by calling HI\_UNF\_PVR\_PlayGetStatus.
- Step 12** Stop time-shift playback by calling HI\_UNF\_PVR\_PlayStopTimeShift, or HI\_UNF\_PVR\_PlayStopChn and HI\_UNF\_PVR\_PlayDestroyChn. After HI\_UNF\_PVR\_PlayDestroyChn is called, stream playback is stopped.
- Step 13** Disable the recording channel by calling HI\_UNF\_PVR\_RecStopChn. After this API is called, stream recording stops.
- Step 14** Release the recording channel by calling HI\_UNF\_PVR\_RecDestroyChn.



**Step 15** (Optional) Deregister a registered callback function by calling `HI_UNF_PVR_UnRegisterEvent`.

**Step 16** Deinitialize the PVR playback module by calling `HI_UNF_PVR_PlayDeInit`.

**Step 17** Deinitialize the PVR recording module by calling `HI_UNF_PVR_RecDeInit`.

**Step 18** Deinitialize the DEMUX module and AVPLAY module.

----End

## Notes

Simultaneous recording and playback can be implemented during time shifting by calling either of the following APIs:

- `HI_UNF_PVR_PlayStartTimeShift`
- `HI_PVR_PlayCreateChn` and `HI_PVR_PlayStartChn`

Note the following:

- You must start recording live programs before time-shift playback. Typically, the rewind mode is used during recording. Otherwise, the file size cannot be controlled.
- You can enable or disable the time-shift function for created recording channel by calling `HI_UNF_PVR_PlayStartTimeShift` or `HI_UNF_PVR_PlayStopTimeShift` respectively.
- You need to create a recording channel and a playback channel with the same file name. Then the contents that are being recorded can be played.

Note the following when initializing the PVR module:

- Initialize the DEMUX module and AVPLAY module before initializing the PVR module, because most operations of the PVR module are based on these two modules.
- Initialize the PVR recording and playback modules before starting recording or playing.
- Ensure that all the recording or playback channels are released before deinitializing the PVR recording or playback module. Otherwise, the deinitialization fails.
- Ensure that all PVR recording or playback channels are stopped before being released. Otherwise, the channels cannot be released.

## Sample

See `sample\pvr\sample_pvr_timeshift.c`.



# Contents

---

<b>22 Teletext.....</b>	<b>1</b>
22.1 Overview .....	1
22.1.1 Application Architecture .....	1
22.1.2 Obtaining Teletext.....	2
22.1.3 Structure.....	2
22.2 Important Concepts .....	3
22.3 Features .....	4
22.4 Development Guide.....	4
22.4.1 Managing the Module .....	4
22.4.2 Injecting Teletext Data .....	7
22.4.3 Switching the Program.....	7
22.4.4 Enabling and Disabling Teletext Display.....	9
22.4.5 Triggering Teletext Display.....	9
22.4.6 Interacting with the User.....	11
22.4.7 Drawing Characters .....	16
22.5 FAQs .....	21
22.5.1 How Do I Use External Font Libraries?.....	21
22.5.2 Does the Teletext Module Support Multiple Color Modes?.....	22



# Figures

<b>Figure 22-1</b> Application architecture of the teletext module .....	1
<b>Figure 22-1</b> Teletext data transfer.....	2
<b>Figure 22-1</b> Structure of program teletext data in the PMT.....	3
<b>Figure 22-1</b> Processing for managing the teletext module .....	6
<b>Figure 22-1</b> Process for displaying teletext pages .....	10
<b>Figure 22-1</b> Process for decoding and drawing teletext pages .....	11
<b>Figure 22-1</b> Working process of the teletext module.....	14
<b>Figure 22-1</b> Relationships between the teletext module and other modules.....	15
<b>Figure 22-2</b> Special teletext drawing situation .....	19



# Tables

<b>Table 22-1</b> HI_UNF_TTX_Create () input parameters.....	5
<b>Table 22-2</b> Memory allocation.....	5
<b>Table 22-3</b> HI_UNF_TTX_SwitchContent () input parameters .....	7
<b>Table 22-4</b> Default key processing mode.....	12
<b>Table 22-5</b> Specified command processing mode.....	13
<b>Table 22-6</b> Callback functions .....	16
<b>Table 22-7</b> Parameters of the character drawing function .....	17
<b>Table 22-8</b> Parameters of the dynamic redefined character drawing function.....	18



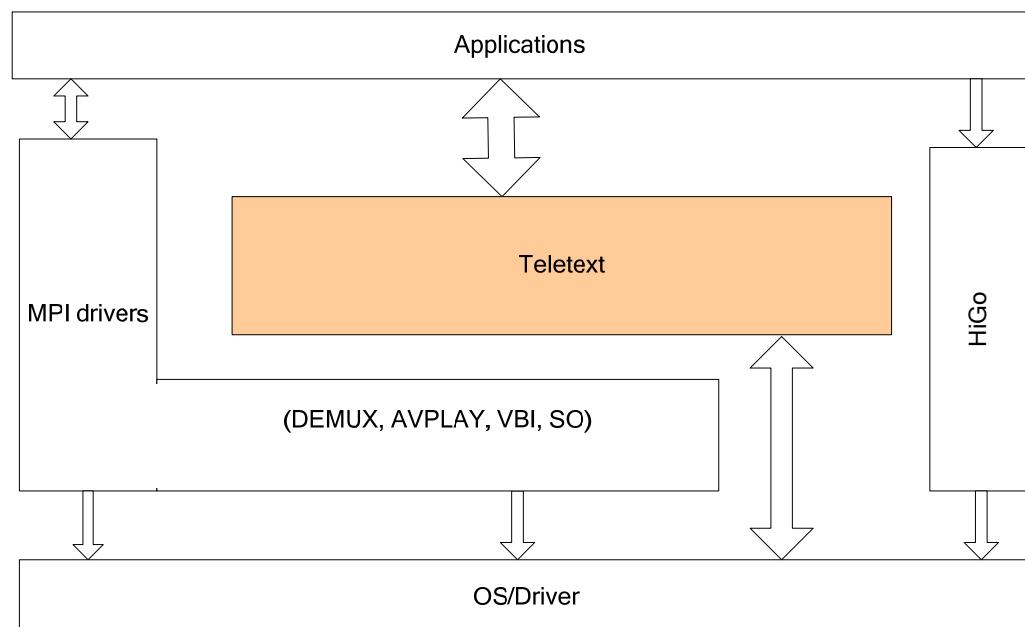
# 22 Teletext

## 22.1 Overview

### 22.1.1 Application Architecture

The teletext module parses the teletext data in media streams into teletexts or subtitles and displays them on the TV screen. It supports the decoding of the teletexts and subtitles, which complies with the EN 300472 (V1.3.1) and EN 300706 (V1.2.1) standards. [Figure 22-1](#) shows the application architecture of the teletext module.

**Figure 22-1** Application architecture of the teletext module





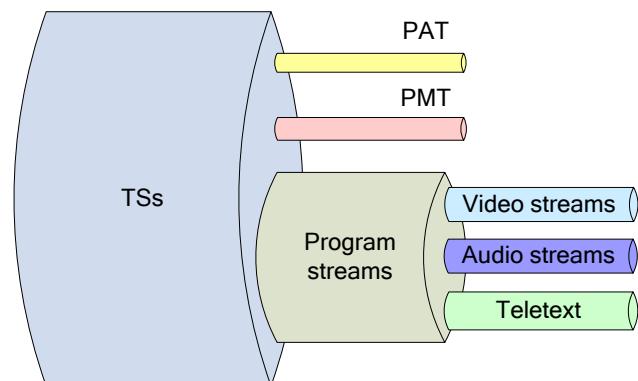
## 22.1.2 Obtaining Teletext

The teletext data streams occupy one PID of the TSs and are encapsulated in PES format. The process for obtaining the teletext data PID is similar to that for obtaining the audio/video data PID. The details are as follows:

- Step 1** Search for the program map table (PMT) based on the program association table (PAT) (the teletext module does not parse the PAT and PMT).
- Step 2** Search for the stream PID whose stream type is 0x06.
- Step 3** Check whether the descriptor is 0x56 (teletext descriptor). If yes, this PID is for teletext streams. Otherwise, end the obtaining process.
- Step 4** Parse the teletext descriptor to obtain the teletext language, type, and initial page address (including the magazine number and page number).

Pay attention to cyclic parsing of this descriptor. That is, there are multiple teletext types in the descriptor. Different teletext types have different languages. For details, see the DVB EN300468 standard. [Figure 22-1](#) shows the teletext data transfer.

**Figure 22-1** Teletext data transfer



----End

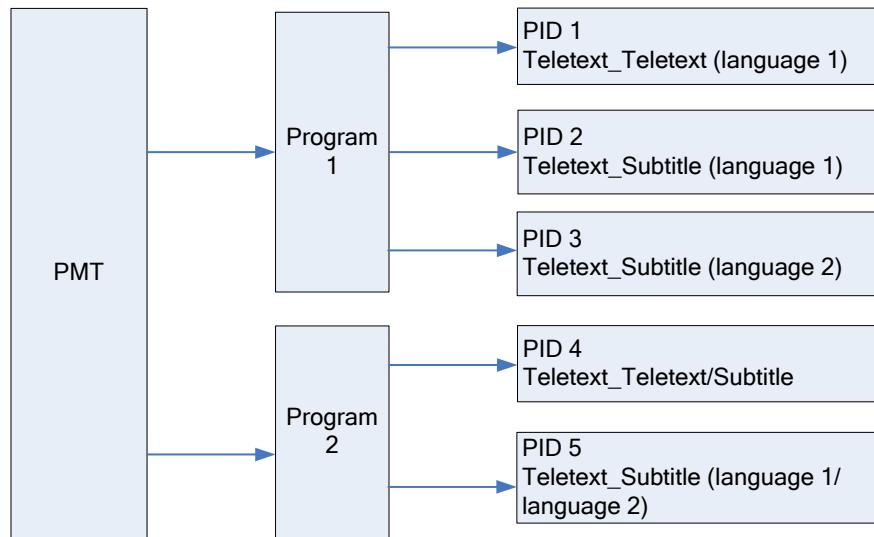
## 22.1.3 Structure

[Figure 22-1](#) shows the structure of program teletext data in the PMT.

- Separate PID. This is the main format of the teletext data transfer. A PID may include teletexts, subtitles, the mixing of teletexts and subtitles (see the PID 4 of program 2), or the mixing of subtitles in different languages (see PID5 of program 2).
- Multiple PIDs. One PID contains only one type of information (see PID1, PID2, and PID3 of program 1). This format is rare.



**Figure 22-1** Structure of program teletext data in the PMT



## 22.2 Important Concepts

### [Page]

A page is a frame displayed on the TV screen. Each page is divided into 24 rows and 40 columns. Each row corresponds to a common data packet in the teletext data. Each page has a page ID that consists of three digits, one is the magazine ID, and the other two are the page ID in the magazine. The page ID ranges from 100 to 899. Teletext data is displayed by page.

### [Magazine]

A magazine can be considered as an electronic book, which consists of many pages. It indicates the set of a type of information. There are at most eight different magazines, numbered from 1 to 8. Each magazine has a maximum of 100 pages, and each page consists of several sub-pages. Therefore, a magazine may contain several hundred pages of information.

### [Display standard]

The teletext has four display levels: level 1, level 1.5, level 2.5, and level 3.5. Level 1 defines the basic teletext page, level 1.5 extends the character system based on level 1, and level 2.5 and level 3.5 define the enhanced teletext page.

### [Output mode]

The teletext module supports vertical blanking interval (VBI) output and OSD output. In OSD output mode, teletext data is decoded and then displayed on the TV screen by the STB. The control over the display page can be implemented by using the remote control of the STB. In VBI output mode, the teletext PES data can be directly transferred to the TV set that supports VBI decoding. The control over the display page can be implemented by using the remote control of the TV set. The remote control for the TV set that supports VBI teletext has the **TEXT** and **SUBTITLE** keys. Pressing the **TEXT** key displays or hide teletexts, and pressing the **SUBTITLE** key displays or hides subtitles.



## 22.3 Features

The teletext module currently supports the main functions defined in DVB Teletext Level 3.5. The teletext module has the following features:

- Supports teletexts and subtitles.
- Provides all font library characters required by teletext, including Latin, Cyrillic, Greek, Hebrew, Arabic, and mosaic. Each character is a 12 x 10 dot matrix.
- Supports multiple languages. Languages are supported based on the upper-layer APIs.
- Supports customized background color and foreground color.
- Supports multiple pages, that is, supports standard page controlling keys, including the digital keys, direction keys, and color keys. The applications can switch to the selected page.
- Supports the functions such as page flash, automatic playback, transparent effect, and information concealing.
- Supports the separate OSD/VBI output and simultaneous OSD and VBI outputs.
- Supports the X/26 character extension.
- Supports object definition and DRCS.
- Supports CLUT redefinition.

The following functions are added to facilitate the integration and extension of the application layer:

- Supports font library extension. The applications can use other font libraries that comply with the teletext standard based on the actual situation.
- Allows the applications to control all the OSD drawing operations. The applications determine the display size and position of the teletext page.
- Supports external memory allocation for teletext pages. The applications allocate the memory for the teletext page and transmit it to the teletext module.

## 22.4 Development Guide

The teletext module is used in the following scenarios:

- Managing the module
- Injecting teletext data
- Switching the program
- Enabling and disabling teletext display
- Triggering teletext display
- Interacting with the user
- Drawing characters

### 22.4.1 Managing the Module

#### Scenario

Managing the teletext module typically involves initializing and deinitializing the module, and creating and destroying instances. The following APIs are provided:



- HI\_UNF\_TTX\_Init
- HI\_UNF\_TTX\_DeInit
- HI\_UNF\_TTX\_Create (For details about the parameters, see [Table 22-1](#).)
- HI\_UNF\_TTX\_Destroy

**Table 22-1** HI\_UNF\_TTX\_Create () input parameters

No.	Parameter	Description
1	pu8MemAddr	These two parameters are used to allocate the memory for the teletext module. The memory is allocated within the teletext module by default. When the memory is insufficient, the memory can be allocated by the application layer. For details, see <a href="#">Table 22-2</a> .
2	u32MemSize	Callback function. For details, see section <a href="#">22.4.7 "Drawing Characters."</a>
3	pfnCB	Whether to enable the flash function. The memory usage is reduced if this function is disabled.
4	bFlash	Whether to use the navigation bar. Memory usage is reduced if the navigation bar is not used.
5	bNavigation	

**Table 22-2** Memory allocation

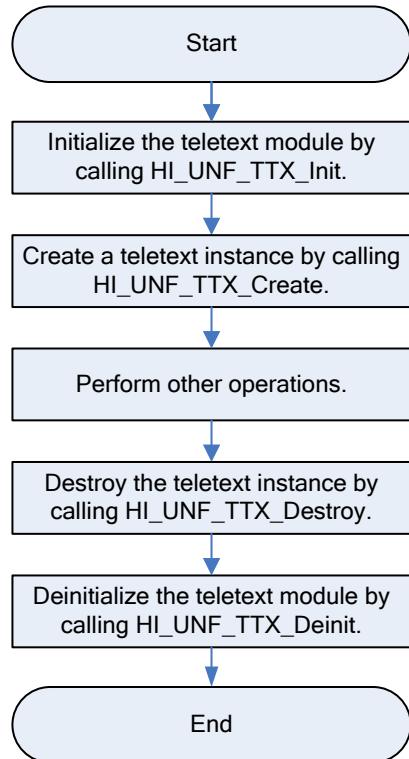
Allocation Mode	Parameter description
Allocated at the application layer	<b>pu8MemAddr</b> is the start address for allocating the memory. <b>u32MemSize</b> is allocated memory size. The value of <b>u32MemSize</b> cannot be less than 80 KB. Otherwise, the teletext instance cannot be created. It is recommended that the value of <b>u32MemSize</b> be not less than 800 KB. Otherwise, data may be received again during page switching, which results in longer wait time and therefore poorer user experience. If the memory allocated at the application layer is less than 2.8 MB, the teletext module cannot store all teletext pages. When the allocated memory is used up, the teletext module replaces a page in the memory with a newly received page by using an algorithm.
Allocated in the teletext module	<b>pu8MemAddr</b> is set to <b>HI_NULL</b> , or <b>u32MemSize</b> is set to <b>0</b> . The memory is allocated in the teletext module in malloc mode. The allocated memory is 2.8 MB. Ensure that the memory is sufficient. Otherwise, an exception occurs. During the calling of <b>HI_UNF_TTX_SwitchContent()</b> , all the teletext page data required by the teletext type of the current program can be saved, and the wait time for page switching can be reduced in this memory allocation mode.



## Working Process

Figure 22-1 shows the process for managing the teletext module.

**Figure 22-1** Processing for managing the teletext module



## Notes

Note the following:

- The teletext module cannot be used to initialize the DISPLAY, DEMUX, and AVPLAY devices. Therefore, you must initialize these devices before using the teletext module-based applications.
- Initialize the teletext module by calling `HI_UNF_TTX_Init` before using its functions.
- Deinitialize the teletext module by calling `HI_UNF_TTX_DeInit` when the teletext functions are not required.

The teletext module currently supports only one instance but not multiple instances. You can create an instance in the following two scenarios:

- If the memory is sufficient, call `HI_UNF_TTX_Create` during system startup.
- If the memory is insufficient, create a teletext instance when the teletext content needs to be displayed, but the instance may fail to be created due to insufficient memory. Frequent calling of `HI_UNF_TTX_Create` may result in memory fragments.

## Sample

For details, see `\sample\teletext` of the SDK.



## 22.4.2 Injecting Teletext Data

### Scenario

After a teletext instance is created, teletext data is received by using the DEMUX and then injected into the teletext instance. The API used in this scenario is HI\_UNF\_TTX.InjectData.

### Working Process

None

### Notes

HI\_UNF\_TTX.InjectData is called repeatedly to inject teletext data. It is recommended that one thread is used to implement the process.

### Sample

For details, see \sample\teletext of the SDK.

## 22.4.3 Switching the Program

### Scenario

If teletext subtitle display is enabled, the subtitles must be switched when the program is switched. In addition, when the user is selecting teletexts and subtitles, the teletext display content must also be switched. The API used in this scenario is HI\_UNF\_TTX.SwitchContent. [Table 22-3](#) describes the parameters.

**Table 22-3** HI\_UNF\_TTX.SwitchContent () input parameters

No.	Switched Content	Description
1	enType	Teletext type. Only teletexts and subtitles are supported.
2	u32ISO639LanCode	ISO639 language code, parsed from the PMT table. It helps the decoder to determine the character set for some special streams.
3	stInitPgAddr	<p>Address for the start page of teletexts.</p> <p>The value of <b>u8MagazineNum</b> ranges from 0 to 7.</p> <p>The value of <b>u8MagazineNum</b> ranges from 0 to 99.</p> <p><b>u16PageSubCode</b> is currently meaningless and can be set to any value.</p> <p>Note the following:</p> <p>The start page is specified based on the following priority (from high to low).</p> <ul style="list-style-type: none"><li>• The start page is specified by <b>8MagazineNum</b> and <b>u8PageNum</b>.</li><li>• The start page is specified by the X/30 packet.</li></ul>



No.	Switched Content	Description
		<ul style="list-style-type: none"><li>The start page is page 100 by default.</li></ul> <p>If the start page does not need to be specified, set <b>u8MagazineNum</b> and <b>u8PageNum</b> to <b>0xFF</b>.</p> <p>The teletext descriptor (0x56) in the PMT includes the description of multiple languages. Teletext multiple languages are identified by different page addresses. The languages can be switched by setting different start page addresses.</p> <p>The teletexts in each language occupy 1 to <math>N</math> pages.</p> <p>The subtitles in each language has only one page, which must be valid. Otherwise, the subtitle data cannot be received. The subtitle data is synchronized with the AV data based on the AV PTS during output.</p>

## Working Process

During program switching, the teletext module must specify the type of the data to be received and receive the new program data. The details are as follows:

- Step 1** The application stops injecting the old program data into the teletext module.
- Step 2** Disable the teletext output by calling `HI_UNF_TTX_Output`.
- Step 3** Set the type of data to be received (teletext or subtitle) by calling `HI_UNF_TTX_SwitchContent`. (This function also clears old program data.)Otherwise, the teletext module still receives the data of the type set last time, and the data in the memory is the data of the old program.
- Step 4** Enable the teletext output by calling `HI_UNF_TTX_Output`.
- Step 5** The application injects the data of the new program into the teletext module.

----End

## Notes

If you set a teletext page address as the initial page address by calling `HI_UNF_TTX_SwitchContent`, the teletext module receives data of this page in priority. After data of this page is received, the teletext module calls the callback function to display this page.

The teletext module cannot process and store teletexts and subtitles at the same time, and therefore it cannot output teletexts and subtitles at the same time.  
`HI_UNF_TTX_SwitchContent` is used to switch the output between teletexts and subtitles.

## Sample

For details, see `\sample\teletext` of the SDK.



## 22.4.4 Enabling and Disabling Teletext Display

### Scenario

If a remote control has the **TTX** key, the user can press the key to display teletext contents, and press it again to hide the display. The user can also set the teletext output mode to OSD output, VBI output (no synchronization during VBI output), or OSD and VBI outputs at the same time. The API used in this scenario is `HI_UNF_TTX_Output`.

### Working Process

None

### Notes

None

### Sample

For details, see `\sample\teletext` of the SDK.

## 22.4.5 Triggering Teletext Display

### Scenario

The message mechanism is used during teletext page display. When teletext pages need to be displayed, the teletext module sends a message to the application for page display. After obtaining the message, the application displays the pages by calling the API provided by the teletext module. The teletext module decodes the data and sends a message to the application. Then the application draws the teletext contents, that is, the application implements all OSD drawing operations. The API used in this scenario is `HI_UNF_TTX_Display`.

### Working Process

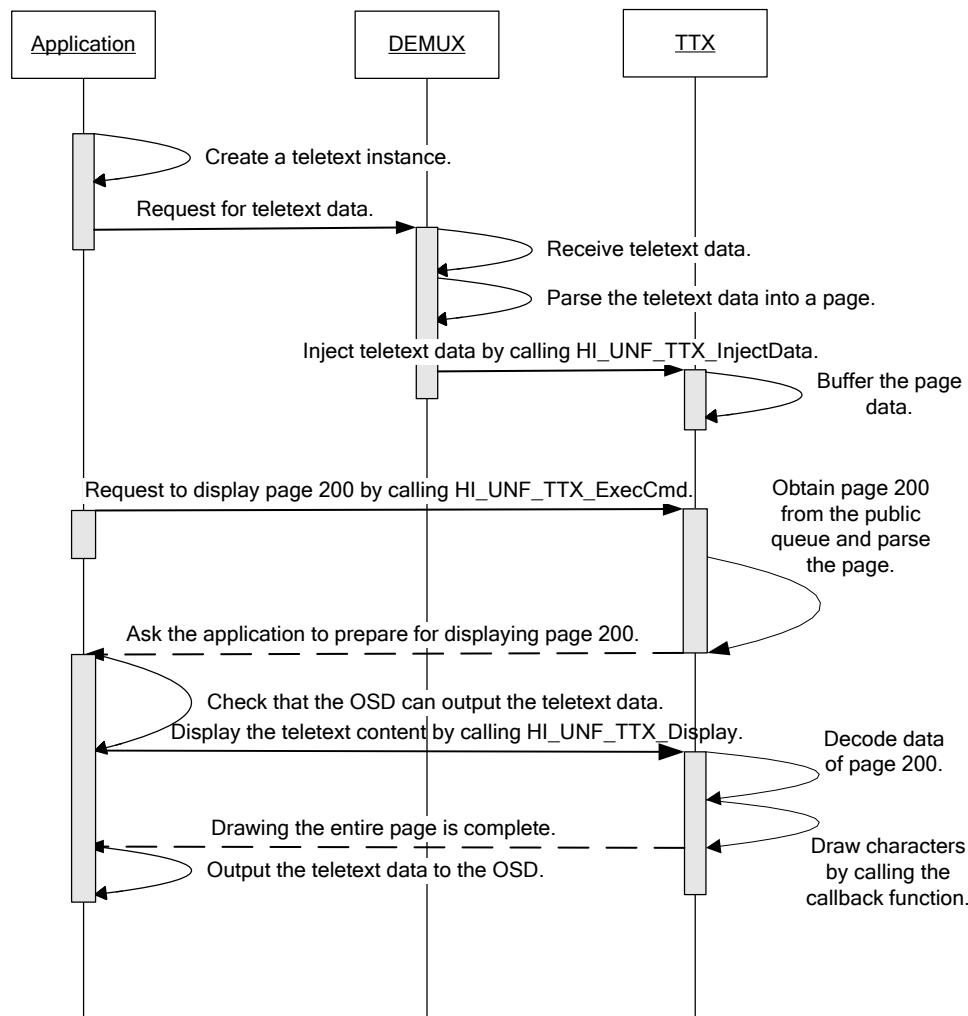
[Figure 22-1](#) shows the process for displaying teletext pages. The details are as follows:

- Step 1** The application receives the teletext data from the DEMUX and organizes the data into teletext pages and saves the pages.
- Step 2** The application requests a page. If the queried page exists, the application transmits the data of this page into the public queue.
- Step 3** The teletext module obtains a page from the public queue for parsing and then sends a message to ask the application to prepare for displaying the page.
- Step 4** Upon receiving the message, the application calls the API provided by the teletext module to parse and display the teletext page. The parsing module calls the callback function to draw the character after each character is parsed. This process is repeated until the entire page is parsed.
- Step 5** After the page is parsed, the content to be displayed is drawn in the buffer. Then the callback function is called to refresh the display buffer to display the teletext page. [Figure 22-1](#) shows the process for decoding and drawing teletext pages.

----End

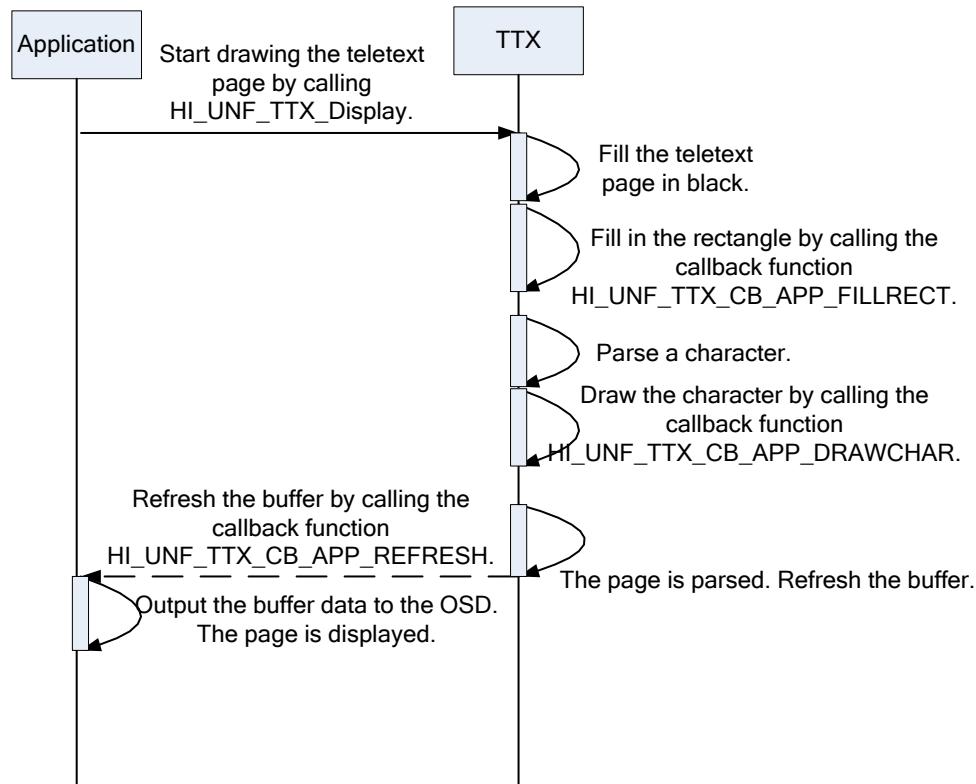


**Figure 22-1** Process for displaying teletext pages





**Figure 22-1** Process for decoding and drawing teletext pages



#### NOTE

That character parsing and drawing are cyclic.

### Notes

The teletext module sends messages to the application by using the callback function, but the application cannot call `HI_UNF_TTX_Display` in the callback function. Otherwise, a function reentrance issue of the teletext module occurs.

It is recommended that the application use the message mechanism and call `HI_UNF_TTX_Display` in a separate thread.

### Sample

For details, see `\sample\teletext` of the SDK.

## 22.4.6 Interacting with the User

### Scenario

The user needs to display or hide the teletext contents by using a remote control, or request the teletext module to display a specific page. The API used in this scenario is `HI_UNF_TTX_ExecCmd`.



The interaction can be processed in the following modes:

- Default key processing mode. The upper layer transmits keys to the teletext module for processing. For details, see [Table 22-1](#).
- Specified command processing mode. The upper layer processes keys and transmits the specified commands to the teletext module. For details, see [Table 22-2](#).

**Table 22-4** Default key processing mode

No.	Key Type	Description
1	HI_UNF_TTX_KEY_0 to HI_UNF_TTX_KEY_9	Digital keys 0 to 9. A specific page can be opened by pressing three digital keys. The magazine key cannot be key 0 or 9.
2	HI_UNF_TTX_KEY_PREVIOUS_PAGE	Previous page
3	HI_UNF_TTX_KEY_NEXT_PAGE	Next page
4	HI_UNF_TTX_KEY_PREVIOUS_SUBPAGE	Previous subpage
5	HI_UNF_TTX_KEY_NEXT_SUBPAGE	Next sub page
6	HI_UNF_TTX_KEY_PREVIOUS_MAGAZINE	Previous magazine
7	HI_UNF_TTX_KEY_NEXT_MAGAZINE	Next magazine
8	HI_UNF_TTX_KEY_RED	The first link in the color navigation bar, corresponding to the <b>RED</b> key on the remote control.
9	HI_UNF_TTX_KEY_GREEN	The second link in the color navigation bar, corresponding to the <b>GREEN</b> key on the remote control.
10	HI_UNF_TTX_KEY_YELLOW	The third link in the color navigation bar, corresponding to the <b>YELLOW</b> key on the remote control.
11	HI_UNF_TTX_KEY_CYAN	The fourth link in the color navigation bar, corresponding to the <b>CYAN</b> key on the remote control. (If there is no <b>CYAN</b> key on the remote control, use the <b>BLUE</b> key.)
12	HI_UNF_TTX_KEY_INDEX	Start page
13	HI_UNF_TTX_KEY_REVEAL	Display or hide the concealing information.
14	HI_UNF_TTX_KEY_HOLD	Automatically play or stop playing sub pages.



No.	Key Type	Description
15	HI_UNF_TTX_KEY_MIX	Switch between transparent and opaque background.
16	HI_UNF_TTX_KEY_UPDATE	Update the current page.

**Table 22-5** Specified command processing mode

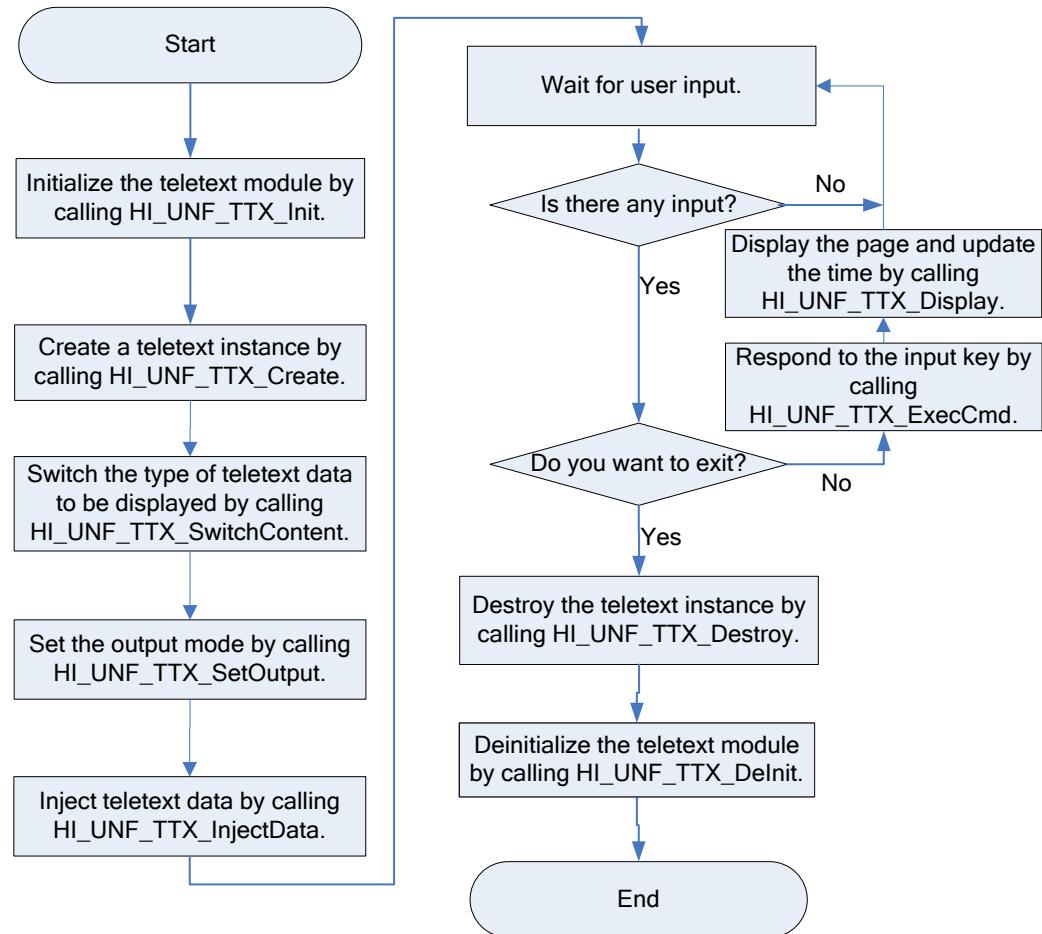
Command Type	Description	Parameter Type
HI_UNF_TTX_CMD_OPE_NPAGE	Displays a specific page in the magazine. This can be replaced by using three digital keys.	(HI_UNF_TTX_PAGE_ADDR_S *)
HI_UNF_TTX_CMD_GETPAGEADDR	Obtain the addresses for the current page, start page, and link page.	(HI_UNF_TTX_GETPAGEADDR_S *)
HI_UNF_TTX_CMD_CHECKPAGE	Check whether the page specified in the parameter is received.	(HI_UNF_TTX_CHECK_PARAM_S *)

## Working Process

[Figure 22-1](#) shows the working process of the teletext module.



**Figure 22-1** Working process of the teletext module



The details are as follows:

- Step 1** Initialize the teletext module by calling `HI_UNF_TTX_Init`.
- Step 2** Create a teletext instance by calling `HI_UNF_TTX_Create`.
- Step 3** Switch the type of the teletext data to be displayed by calling `HI_UNF_TTX_SwitchContent`.
- Step 4** Set the teletext output mode by calling `HI_UNF_TTX_Output`.
- Step 5** Inject teletext data by calling `HI_UNF_TTX.InjectData`.
- Step 6** Wait for the user to input keys. If a key is input, it is transmitted to the teletext module.
- Step 7** Respond to a key input or command request by calling `HI_UNF_TTX_ExecCmd`. Then transmit drawing information to the application to notify the application of the OSD drawing operation.
- Step 8** After the application receives the OSD drawing information, it calls `HI_UNF_TTX_Display` to perform the OSD drawing operation, such as displaying a teletext page, updating time, or flashing.
- Step 2** Repeat Step 7 and Step 8.



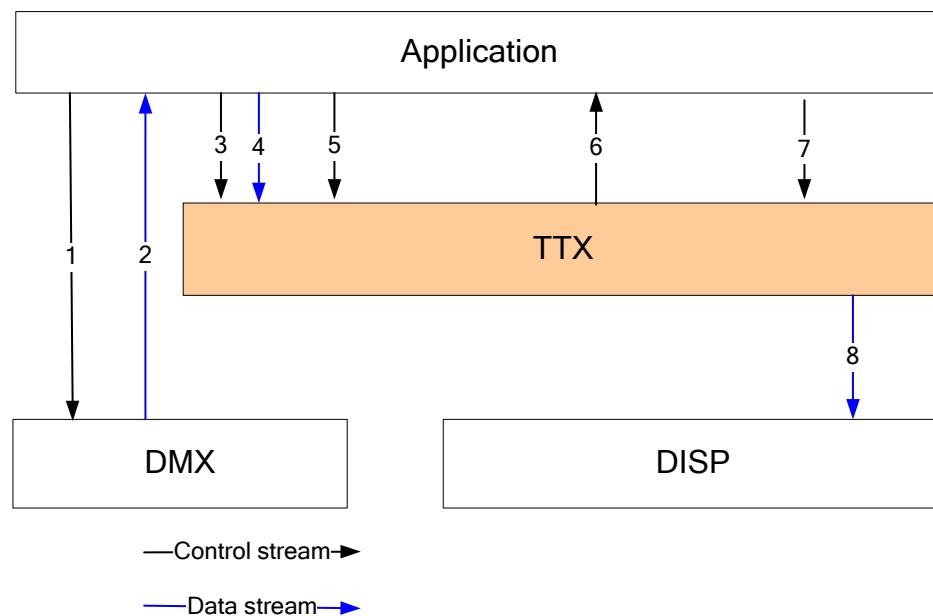
**Step 9** Stop receiving teletext data, disable the teletext output, and destroy the teletext instance to release the system resources by calling HI\_UNF\_TTX\_Destroy.

**Step 10** Deinitialize the teletext module by calling HI\_UNF\_TTX\_DeInit.

----End

[Figure 22-1](#) shows the relationships between the teletext module and other modules.

**Figure 22-1** Relationships between the teletext module and other modules



[Figure 22-1](#) is described as follows:

- 1: Start the DEMUX for receiving data.
- 2: The DEMUX returns data to the application.
- 3: Create a teletext instance and prepare to receive teletext data.
- 4: The application injects data into the teletext module by calling HI\_UNF\_TTX\_InjectData.
- 5: The application requests to display a page by calling HI\_UNF\_TTX\_ExecCmd.
- 6: The teletext module sends a message to the application by calling the callback function.
- 7: Display the page by calling HI\_UNF\_TTX\_Display.
- 8: Display the page on the OSD.

## Notes

None



## Sample

For details, see `\sample\teletext` of the SDK.

### 22.4.7 Drawing Characters

#### Scenario

The teletext module does not contain font libraries. Therefore, the character sets to be displayed are implemented by using the callback function mechanism. The API used in this scenario is `HI_UNF_TTX_CB_FN`. [Table 22-6](#) describes the callback functions.

**Table 22-6** Callback functions

Callback Function Type ( <code>HI_UNF_TTX_CB_E</code> )	Description	Parameter Type
<code>HI_UNF_TTX_CB_TTX_TO_APP_MSG</code>	Sends the OSD drawing information to the application. The application calls the related functions to perform different drawing operations based on the received drawing handle.	( <code>HI_HANDLE *</code> )
<code>HI_UNF_TTX_CB_APP_FILLRECT</code>	Fills the rectangle with the specific color.	( <code>HI_UNF_TTX_FILLRECT_S *</code> )
<code>HI_UNF_TTX_CB_APP_DRAWCHAR</code>	Character drawing type. Obtains the specified characters from the font libraries and draws the characters with the specified background and foreground colors on the OSD.	<code>HI_UNF_TTX_DRAWCHAR_S*</code>  For details about the parameters, see <a href="#">Table 22-2</a> .   <b>NOTE</b> If the background color and foreground color of characters are the same, only the foreground color is filled, and the background color is ignored. For details, see <a href="#">Figure 22-2</a> .
<code>HI_UNF_TTX_CB_APP_DRAWDRCSSCHAR</code>	Dynamic redefined character drawing type. Draws the dynamic redefined character in the specified area of the OSD based on the specified background color and the color of each pixel of the dynamic redefined character.	<code>HI_UNF_TTX_DRAWDRCSCHAR_S*</code>  For details about the parameters, see <a href="#">Table 22-8</a> .
<code>HI_UNF_TTX_CB_APP_REFRESHLAYER</code>	Graphics layer refresh type. Displays the teletext page content on the OSD.	<code>HI_UNF_TTX_REFRESHLAYER_S*</code>  Note: This callback function



Callback Function Type (HI_UNF_TTX_CB_E)	Description	Parameter Type
		is required only in dual-buffer drawing mode.
HI_UNF_TTX_CB_GETLOCALTIME	Obtains the local time and displays it in the upper right corner. If <b>HI_FAILURE</b> is returned, the displayed time starts from 12: 00:00 by default.	HI_UNF_TTX_TIME_S*
HI_UNF_TTX_CB_GETPTS	Obtains the PTS of the current stream. AV PTS is recommended. (If <b>HI_FAILURE</b> is returned, the subtitles are displayed without synchronization.)	HI_S64 *
HI_UNF_TTX_CB_CREATETE_BUFF	Creates a temporary buffer in multi-buffer drawing mode.	HI_UNF_TTX_BUFFER_PARAM_S *  Note: In single buffer mode, <b>HI_SUCCESS</b> must be returned by the callback function.
HI_UNF_TTX_CB_DESTROY_BUFF	Releases the temporary buffer in multi-buffer mode.	None

**Table 22-7** Parameters of the character drawing function

No.	Parameter	Description
1	HI_UNF_TTX_PAGEAREA_S { HI_U32 u32Row: 8; HI_U32 u32Column: 8; HI_U32 u32RowCount: 8; HI_U32 u32ColumnCount: 8; } *pstPageArea;	<b>pstPageArea</b> specifies the area of the teletext page that a character is located. <b>u32Row</b> indicates the ID of the row in which a character is displayed on the teletext page. <b>u32Column</b> indicates the ID of column in which the character is displayed on the teletext page. <b>u32RowCount</b> indicates the number of rows that the character occupies. For example, a normal character occupies one row, but a character with additional height occupies two rows. <b>u32ColumnCount</b> indicates the number of columns occupied by a character. For example, a normal character occupies one row, but a character with additional width occupies two columns.
2	HI_UNF_TTX_CHARATTR_	<b>pstCharAttr</b> is the character attribute



No.	Parameter	Description
	<pre>S { HI_U32 u32Index: 8; HI_BOOL bContiguous: 1; HI_UNF_TTX_G0SET_E enG0Set: 3; HI_UNF_TTX_G2SET_E enG2Set: 3 HI_UNF_TTX_CHARSET_E enCharSet: 3; HI_U32 u8NationSet : 6; u8Reserved: 8; }* pstCharAttr;</pre>	<p>that specifies the position of the character in a font library. <b>u32Index</b> is the index (0x20–0x7f) of a character in a character set. <b>bContiguous</b> indicates whether the character is consecutive mosaic and is valid when <b>enCharSet</b> is <b>G1</b>.</p> <p><b>enG0Set</b> indicates that the character set G0 is used and is valid when <b>enCharSet</b> is <b>G0</b>. <b>enG2Set</b> indicates that the character set G2 is used and is valid when <b>enCharSet</b> is <b>G2</b>.</p> <p><b>enCharSet</b> is the character set of the character. <b>u8NationSet</b> is a country sub character set and is valid when <b>enCharSet</b> is <b>G0</b> and <b>enG0Set</b> is <b>Latin</b>. The position of a character in a specific font library can be determined by using these parameters.</p> <p>Note: For details about the concepts in the character attributes, see the EN 300706 (V1.2.1) standard.</p>
3	HI_UNF_TTX_COLOR u32Foreground	Foreground color of a character
4	HI_UNF_TTX_COLOR u32Background	Background color of a character

**Table 22-8** Parameters of the dynamic redefined character drawing function

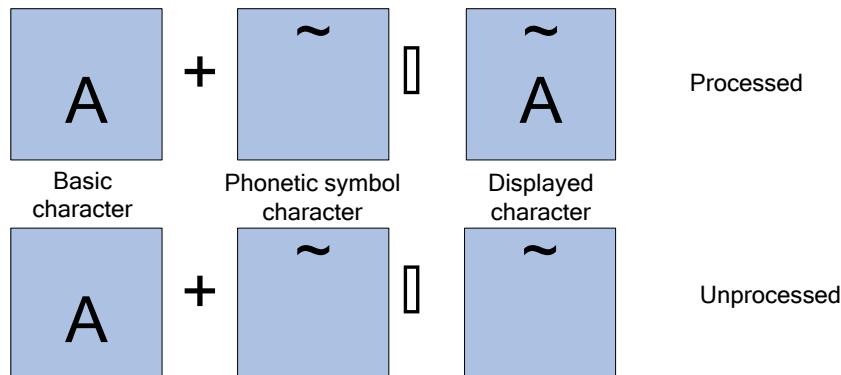
No.	Parameters	Description
1	<pre>HI_UNF_TTX_PAGEAREA_S { HI_U32 u32Row : 8; HI_U32 u32Column : 8; HI_U32 u32RowCount : 8; HI_U32 u32ColumnCount : 8; }*pstPageArea;</pre>	<p><b>pstPageArea</b> specifies the area of the teletext page where a character is located. <b>u32Row</b> indicates the ID of the row in which a character is displayed on the teletext page. <b>u32Column</b> indicates the ID of column in which the character is displayed on the teletext page. <b>u32RowCount</b> indicates the number of rows that the character occupies. For example, a normal character occupies one row, and a character with additional height occupies two rows. <b>u32ColumnCount</b> indicates the number of columns occupied by a character. For example, a normal character occupies one column, and a character with additional width occupies two columns.</p>



No.	Parameters	Description
2	HI_UNF_TTX_COLOR u32Background	Background color of a character
3	HI_UNF_TTX_COLOR* pu32DRCSColourInfo	Color of each pixel of a character, expressed by a 32-bit color value
4	HI_UNF_TTX_DRCS_SIZE_E enDRCSSize	Character size. The normal size is 12 x 10 pixels, and the minimized size is 6 x 5 pixels.

As shown in [Figure 22-2](#), there is a special situation in the process of drawing characters. The characters in a certain position of the teletext page consist of basic characters and phonetic symbol characters. The basic characters and then the phonetic symbol characters are drawn in sequence. Without special processing, the phonetic symbol characters overwrite the basic characters. Because the character sets to which phonetic symbol characters belong are G2 character set (the Arabic G2 character set contains no phonetic symbol character), and the code of phonetic symbol characters is between 0x40 and 0x50 (excluding 0x40 and 0x50), the drawing function can identify phonetic symbol characters based on this feature and draws only the foreground color but not the background color for phonetic symbol characters. In this way, basic characters will not be overwritten by phonetic symbol characters.

**Figure 22-2** Special teletext drawing situation



The following is a sample of the callback function.

```
HI_S32 TTX_SampleCallBack(HI_HANDLE hTTX, HI_UNF_TTX_CB_E enCB, HI_VOID *pvCBParam)
{
    HI_S32 s32Ret = 0;
    HI_UNF_TTX_FILLRECT_S *pstFillrectparam;
    HI_UNF_TTX_DRAWCAHR_S *pstDrawparam;
    HI_UNF_TTX_REFRESHLAYER_S *pstRefreshparam;
    HI_HANDLE hDispalyHandle;
    if ((HI_NULL == pvCBParam) || (enCB > HI_UNF_TTX_CB_BUTT))
    {
        return HI_FAILURE;
```



```
        }

        switch (enCB)
        {
            case HI_UNF_TTX_CB_APP_FILLRECT:
            {
                pstFillrectparam = (HI_UNF_TTX_FILLRECT_S *)pvCBParam;
                TTX_FillRect(pstFillrectparam);
                s32Ret = HI_SUCCESS;
            }
            break;

            case HI_UNF_TTX_CB_APP_DRAWCHAR:
            {
                pstDrawparam = (HI_UNF_TTX_DRAWCAHR_S *)pvCBParam;
                TTX_DrawChar(pstDrawparam);
                s32Ret = HI_SUCCESS;
            }
            break;

            case HI_UNF_TTX_CB_APP_DRAWDRCSCHAR:
            {
                pstDrawparam = (HI_UNF_TTX_DRAWDRCSCHAR_S *)pvCBParam;
                TTX_DrawDRCSChar(pstDrawparam);
                s32Ret = HI_SUCCESS;
            }
            break;

            case HI_UNF_TTX_CB_APP_REFRESH:
            {
                pstRefreshparam = (HI_UNF_TTX_REFRESHLAYER_S *)pvCBParam;
                TTX_ShowOsd(pstRefreshparam);
                s32Ret = HI_SUCCESS;
            }
            break;

            case HI_UNF_TTX_CB_GETLOCALTIME:
            {
                s32Ret = TTX_GetLocalTime((HI_UNF_TTX_TIME_S *)pvCBParam);
            }
            break;

            case HI_UNF_TTX_CB_GETPTS:
            {
                s32Ret = TTX_GetCurPlayingPTS((HI_S64 *)pvCBParam);
            }
            break;

            case HI_UNF_TTX_CB_TTX_TO_APP_MSG:
            {
                hDispalyHandle = *((HI_HANDLE *)pvCBParam);
            }
        }
```



```
    TTX_SendGUIMsg(&hDispalyHandle);
    s32Ret = HI_SUCCESS;
}
break;
case HI_UNF_TTX_CB_CREATE_BUFF:
{
    s32Ret = TTX_Create_Buffer((HI_UNF_TTX_BUFFER_PARAM_S
*)pvCBParam);
}
break;
case HI_UNF_TTX_CB_DESTROY_BUFF:
{
    s32Ret = TTX_Destroy_Buffer();
}
break;
default:
break;
}
return s32Ret;
}
```

## Working Process

None

## Notes

None

## Sample

For details, see \sample\teletext of the SDK.

## 22.5 FAQs

### 22.5.1 How Do I Use External Font Libraries?

#### Question

How do I use external font libraries?

#### Solution

The teletext module supports external libraries, which must contain all the characters specified in the teletext standards EN 300472 (V1.3.1) and EN 300706 (V1.2.1).



To use external libraries, perform the following steps:

- Step 1** Search for the positions of the characters specified in EN 300472 (V1.3.1) and EN 300706 (V1.2.1) in the external library and record the position information.
- Step 2** Find the position of a character in the external font library by using the position information table created in step 1 based on the character attributes (the index in a character set, whether consecutive mosaic occurs, G0 main character set, character set, and country character subset).
- Step 3** Draw the character by using the specified foreground color and background color on the specified position of the OSD based on the position of the character in the external font library.

See `sample_teletext_font.c`.

--End

## 22.5.2 Does the Teletext Module Support Multiple Color Modes?

### Question

Does the teletext module support multiple color modes?

### Solution

The teletext module uses the 32-bit (ARGB8888) color mode, but also supports multiple color modes. When the drawing function is implemented in HI\_UNF\_TTX\_CB\_FN, the 32-bit (ARGB8888) color mode can be replaced by another color mode, but the display effect differs from that of the 32-bit color mode, especially the transparency effect. If the 32-bit color transmitted by the teletext module is 0xABCD89, the 16-bit color converted from the 32-bit color is 0xACE8.

The 32-bit color In32Color is converted into the 16-bit color Out16Color. The formula is as follows:

```
Out16Color = ((In32Color>>28)&0xf)<<12 | ((In32Color>>20)&0xf)<<8  
| ((In32Color>>12)&0xf)<<4 | ((In32Color>>4)&0xf)
```

If the target graphical system uses the 8-bit color mode, you need to convert the 32-bit color into the 8-bit color. If the 32-bit color input by the teletext module is 0xABCD89, find the same or a similar color in the palette of the target graphical system. The index value of this color can be used as the 8-bit color value of 0xABCD89 in the target graphical system.



# Contents

---

<b>23 Subtitle .....</b>	<b>1</b>
23.1 Overview .....	1
23.1.1 Application Architecture .....	1
23.1.2 Structure.....	1
23.2 Important Concepts .....	3
23.3 Features .....	3
23.4 Development Guide.....	4
23.4.1 Managing the Module .....	4
23.4.2 Parsing DVB Subtitles .....	6
23.4.3 Parsing SCTE Subtitles.....	7
23.4.4 Selecting a Subtitle Stream .....	9
23.4.5 Switching the Program.....	10
23.4.6 Using Callback Functions .....	10
23.5 FAQs .....	12
23.5.1 How Are Subtitles Displayed and Synchronized?.....	12
23.5.2 How Are Subtitles Drawn by Using Callback Functions? .....	14
23.5.3 Why Is the Callback Function for Obtaining the System PTS Required When the Subtitle Stream Contains PTS Information?.....	16



# Figures

<b>Figure 23-1</b> Application architecture of the subtitle module .....	1
<b>Figure 23-2</b> DVB subtitle description in the PMT .....	2
<b>Figure 23-3</b> SCTE subtitle description in the PMT .....	3
<b>Figure 23-4</b> Processing for managing the subtitle module .....	5
<b>Figure 23-5</b> Process for parsing DVB subtitles .....	6
<b>Figure 23-6</b> Process for parsing SCTE subtitles.....	8



## Tables

<a href="#">Table 23-1 HI_UNF_SUBT_DATA_S parameters</a> .....	11
<a href="#">Table 23-2 HI_UNF_SUBT_GETPTS_FN parameters</a> .....	12



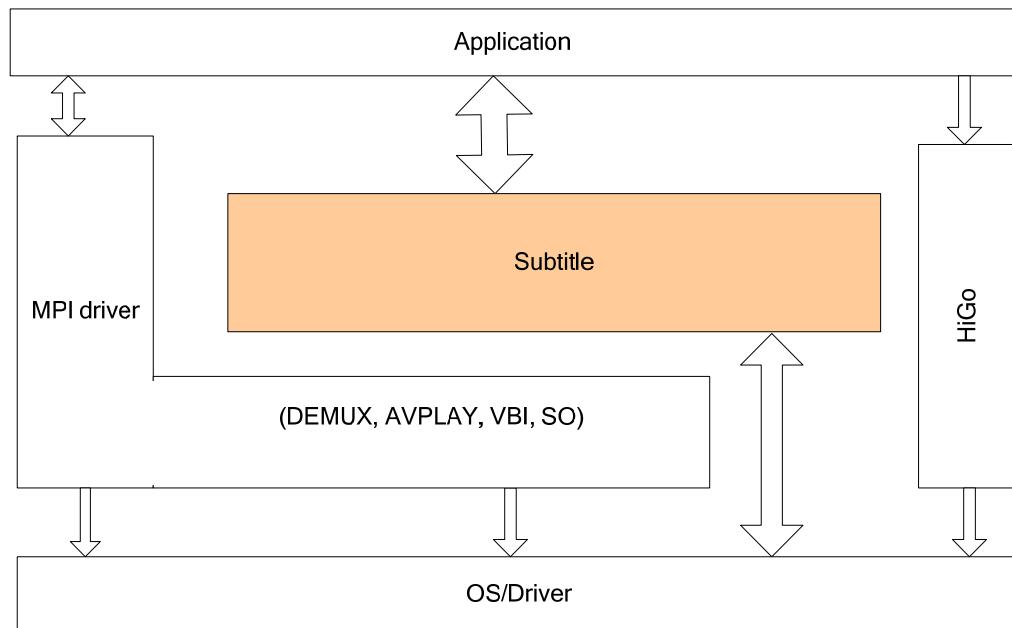
# 23 Subtitle

## 23.1 Overview

### 23.1.1 Application Architecture

The subtitle module decodes subtitle data in media streams into subtitles that can be displayed. Currently it supports the decoding of DVB and Society of Cable Telecommunications Engineers (SCTE) subtitles. [Figure 23-1](#) shows the application architecture of the subtitle module.

**Figure 23-1** Application architecture of the subtitle module



### 23.1.2 Structure

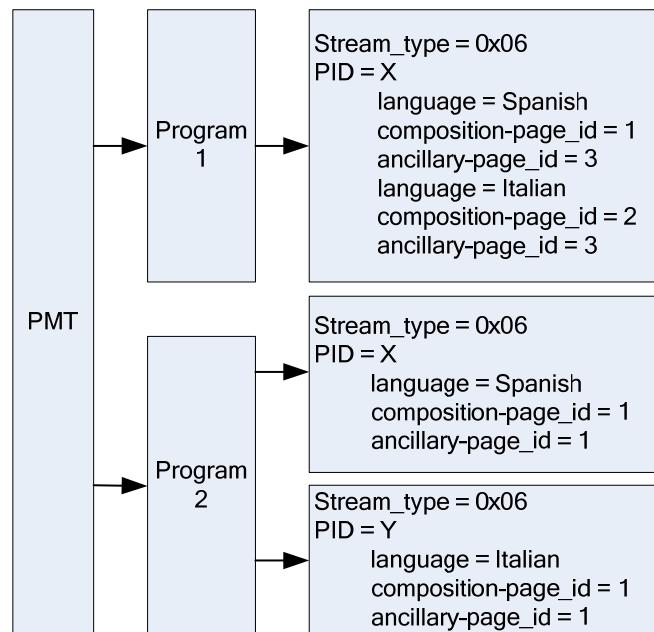
The subtitle module has two sub modules: DVB subtitle and SCTE subtitle. The two sub modules are independent of each other. The transmission modes of DVB subtitles and SCTE



subtitles in TSs are different and therefore descriptions about them in the PMT are also different. The following describes the structure of the two subtitles in the PMT.

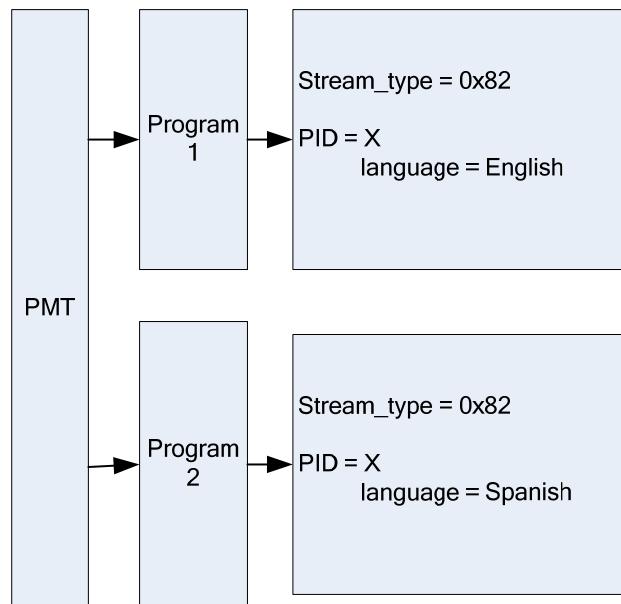
- DVB subtitles are defined as an independent ES and are multiplexed in TSs. The PID information of DVB subtitles can be obtained by parsing the PMT. In the PMT, the DVB subtitle stream type is 0x06, and data attributes include the subtitle PID, page PID (composition-page\_id), ancillary page ID (ancillary-page\_id), and language code. [Figure 23-2](#) shows the two formats of the description information in the PMT.
  - Single PID: Subtitles are distinguished by page ID.composition-page\_id indicates one subtitle stream. For example, in program 1, the streams with the same PID have two subtitle streams of different languages.
  - Multiple PIDs: Each subtitle has its PID. Each PID corresponds to one subtitle stream. For example, in program 2, there are two subtitles streams, which require two PESs to transmit.
- SCTE subtitles are not transmitted in PESs but as section data. In the PMT, the SCTE subtitle stream type is 0x82, and you can obtain the subtitle PID and language information in the PMT. See [Figure 23-3](#).

**Figure 23-2** DVB subtitle description in the PMT





**Figure 23-3** SCTE subtitle description in the PMT



## 23.2 Important Concepts

[Page]

A page is the collection of subtitles in the subtitle service on the screen, and it contains no region or multiple regions. A region is a rectangle area on the screen for displaying objects, and it contains no region or multiple objects. The object is the basic element displayed on the screen, which includes the station icon, subtitle, and map. Each object is a graphics unit. The pages are used to transmit all elements, including graphics, on the pages in a subtitle service.

[Ancillary page]

The ancillary page is used to transmit the subtitle elements shared by multiple subtitle services, such as the station icon.

[CLUT]

The color look-up table (CLUT) is used to convert the false colors of objects into the true color on the TV screen.

## 23.3 Features

The subtitle module decodes subtitle streams in TSs into subtitles in graphics or text format. It has the following two functions:

- Decodes PES data packets and DVB subtitles, complying with the ETSI EN3 00 743 V1.2.1 standard. DVB subtitles can be encoded in two formats: character set and bitmap format. In most cases, DVB subtitles are encoded in bitmap format. The decoding chip must support multi-layer OSDs, because the subtitles must occupy one OSD layer.



- Decodes section data and SCTE subtitles, complying with the SCTE 27 standard. The SCTE subtitle encoder converts the characters into the bitmap format for encoding, and the subtitle module decodes the graphics and converts them into subtitles.

## 23.4 Development Guide

The subtitle module is used in the following scenarios:

- Managing the module
- Parsing DVB subtitles
- Parsing SCTE subtitles
- Selecting a subtitle stream
- Switching the program
- Using callback functions

### 23.4.1 Managing the Module

#### Scenario

Managing the subtitle module typically involves initializing and deinitializing the module, and creating and destroying instances. You must initialize the subtitle module before using other APIs of the module, and deinitialize the module when it is no longer required. Creating an instance prepares resources required by the instance, and destroying an instance releases the resources that are allocated to the instance. Some initial parameters, such as subtitle numbers and other information, must be specified when you create an instance. The following APIs are provided:

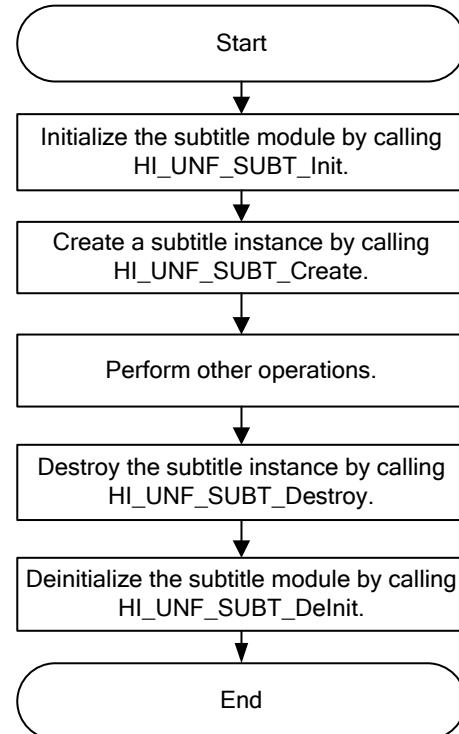
- HI\_UNF\_SUBT\_Init: Initializes the subtitle module.
- HI\_UNF\_SUBT\_DeInit: Deinitializes the subtitle module.
- HI\_UNF\_SUBT\_Create: Creates an instance. For details about the parameters, see the header file.
- HI\_UNF\_SUBT\_Destroy: Destroy an instance. Ensure that the instance is created before calling this API.

#### Working Process

Figure 23-4 shows the working process for managing the subtitle module.



**Figure 23-4** Processing for managing the subtitle module



## Notes

The subtitle module cannot be used to initialize the DISPLAY, DEMUX, and AVPLAY devices. Therefore, you must initialize these devices before using the subtitle module-based applications.

Each instance of the subtitle module supports the decoding of one type of subtitles, which must be specified in **enDataType** when the instance is created. The subtitle module creates the appropriate device for decoding based on the specified subtitle type. If the subtitle type is not specified, a DVB subtitle decoding instance is created by default.

Ensure that the subtitle module is initialized when creating an instance by calling `HI_UNF_SUBT_Create`. Destroy the instance to release resources when the instance is not required.

The subtitle module supports a maximum of eight instances. The attempt to create the ninth instance will fail.

## Sample

For details, see `\sample\subtitle` of the SDK.



## 23.4.2 Parsing DVB Subtitles

### Scenario

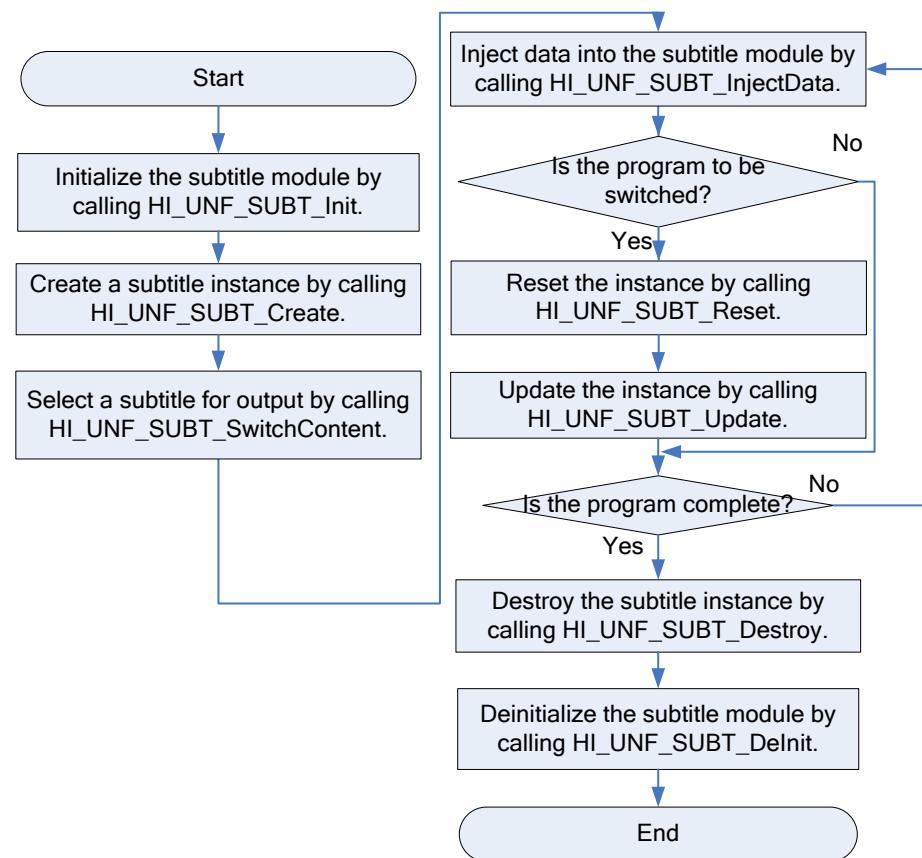
The DVB subtitle data comes from PESs. To decode DVB subtitles, set **enDataType** to **HI\_UNF\_SUBT\_DVB** when calling **HI\_UNF\_SUBT\_Create**. DVB subtitles are decoded by default. The following APIs are provided:

- **HI\_UNF\_SUBT.InjectData**: Injects PES data into the subtitle module.
- **HI\_UNF\_SUBT.SwitchContent**: Selects a subtitle for output.
- **HI\_UNF\_SUBT.Update**: Updates the subtitles of the new program into the subtitle module during program switching.
- **HI\_UNF\_SUBT.Reset**: Resets the subtitle module.

### Working Process

Figure 23-5 shows the process for parsing DVB subtitles.

**Figure 23-5** Process for parsing DVB subtitles



To parse DVB subtitles, perform the following steps:

**Step 1** Initialize the subtitle module by calling **HI\_UNF\_SUBT\_Init**.

**Step 2** Create a subtitle instance by calling **HI\_UNF\_SUBT\_Create**.



- Step 3** Select a subtitle by calling HI\_UNF\_SUBT\_SwitchContent.
- Step 4** Inject subtitle data by calling HI\_UNF\_SUBT\_InjectData.
- Step 5** (Optional) Reset the subtitle instance by calling HI\_UNF\_SUBT\_Reset if the program is to be switched.
- Step 6** Update the subtitle instance by calling HI\_UNF\_SUBT\_Update.
- Step 7** Destroy the subtitle instance to release system resources by calling HI\_UNF\_SUBT\_Destroy.
- Step 8** Deinitialize the subtitle module by calling HI\_UNF\_SUBT\_DeInit.

 **NOTE**

You can repeat step 3 and step 4.

----End

## Notes

Note the following:

- Before HI\_UNF\_SUBT\_InjectData is called, the type of the DEMUX channel for filtering PES data and parsing DVB subtitles must be HI\_UNF\_DMX\_CHAN\_TYPE\_PES. There is no requirement on the DEMUX ID, as long as the DEMUX supports the filtering of PES data. The DEMUX filter attribute **au8Mask[0]** must be **0xBD** because the subtitle stream ID is 0xBD.
- The PID in the parameters of HI\_UNF\_SUBT\_InjectData is used to distinguish subtitles. The descriptive data (such as the PID, page ID, and ancillary page ID) for each subtitle stream in the PMT can be in the two formats shown in [Figure 23-2](#), and the key attributes for distinguishing the subtitle streams are their PIDs, page IDs, and ancillary page IDs. The PID for each subtitle stream may be the same or different, and each subtitle instance can decode multiple subtitle streams. Therefore, the subtitle stream is distinguished by PID.
- The subtitle module buffers the data of each subtitle stream to ensure that the subtitle is displayed immediately after you switch to a program with multiple subtitle streams even though the subtitle data is broadcasted before video frames.
- The DEMUX task for receiving subtitle PES data can come with other DEMUX tasks because DVB subtitle PES data is small. The maximum size of one PES packet is 64 KB, and its broadcast time is typically more than 2 seconds. If the subtitle display time is very short, users may miss the subtitle.

## Sample

For details, see `\sample\subtitle` of the SDK.

### 23.4.3 Parsing SCTE Subtitles

#### Scenario

To decode SCTE subtitles, set **enDataType** to **HI\_UNF\_SUBT\_SCTE** when calling HI\_UNF\_SUBT\_Create. The SCTE subtitle device buffers the injected section data. The following APIs are provided:

- **HI\_UNF\_SUBT\_RegGetPtsCb**: Registers a callback function for obtaining video PTS.

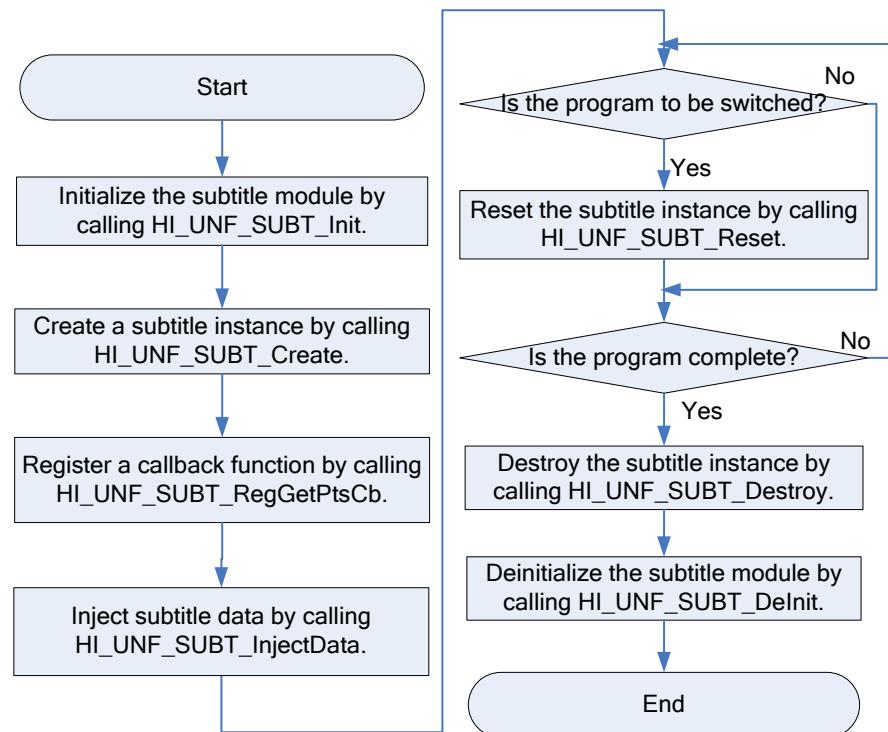


- HI\_UNF\_SUBT\_InjectData: Injects section data into the subtitle module.
- HI\_UNF\_SUBT\_Reset: Resets the subtitle module.

## Working Process

Figure 23-6 shows the process for parsing SCTE subtitles.

**Figure 23-6** Process for parsing SCTE subtitles



To parse SCTE subtitles, perform the following steps:

- Step 1** Initialize the subtitle module by calling HI\_UNF\_SUBT\_Init.
- Step 2** Create a subtitle instance by calling HI\_UNF\_SUBT\_Create.
- Step 3** Register a callback function for obtaining the current PTS by calling HI\_UNF\_SUBT\_RegGetPtsCb.
- Step 4** Inject subtitle data by calling HI\_UNF\_SUBT\_InjectData.
- Step 5** (Optional) Reset the subtitle instance by calling HI\_UNF\_SUBT\_Reset if the program is to be switched.
- Step 6** Destroy the subtitle instance to release system resources by calling HI\_UNF\_SUBT\_Destroy.
- Step 7** Deinitialize the subtitle module by calling HI\_UNF\_SUBT\_DeInit.

----End



## Notes

The data injected by calling HI\_UNF\_SUBT.InjectData must be section data; otherwise, a decoding error may occur.

Subtitles do not need to be switched or updated in the SCTE services, and therefore HI\_UNF\_SUBT\_SwitchContent and HI\_UNF\_SUBT\_Update are not used. If HI\_UNF\_SUBT\_RegGetPtsCb is not called after a subtitle instance is created, some program subtitles cannot be displayed properly, because in the SCTE subtitle standard, if the subtitle PTS is too large to be included in 32 bits, the PTS is reduced for transmission.

HI\_UNF\_SUBT\_RegGetPtsCb is used to synchronize the PTS with the current system PTS.

HI\_UNF\_SUBT\_RegGetPtsCb is used only in the SCTE service but not the DVB service.

Before HI\_UNF\_SUBT.InjectData is called, the type of the DEMUX channel for filtering section data and parsing SCTE subtitles must be HI\_UNF\_DMX\_CHAN\_TYPE\_SEC. The DEMUX filter attribute **au8Mask[0]** must be **0xC6** because the SCTE subtitle table ID is 0xC6.

## Sample

For details, see [\sample\subtitle](#) of the SDK.

### 23.4.4 Selecting a Subtitle Stream

#### Scenario

The DVB subtitles often contain multiple subtitle streams to meet requirements of different countries. HI\_UNF\_SUBT\_SwitchContent is used to select a subtitle stream from multiple subtitle streams. The API is not used for the SCTE service because SCTE subtitles provide only one subtitle language. The subtitle module provides multiple buffers for DVB subtitles, and multiple subtitle streams are stored in the buffers at the same time.

HI\_UNF\_SUBT\_SwitchContent selects the correct buffer data for parsing. The parameters of HI\_UNF\_SUBT\_SwitchContent specify the attributes of the selected subtitle.

#### Working Process

None

## Notes

HI\_UNF\_SUBT\_SwitchContent is designed for multi-language subtitle decoding, therefore it is called when the program has several subtitle streams in different languages and one specific subtitle stream needs to be selected. SCTE subtitles do not involve multi-language design and therefore do not need this API.

You can specify the subtitle stream information (PID, page ID, and ancillary page ID) by calling HI\_UNF\_SUBT\_SwitchContent to output the desired subtitle stream.

## Sample

For details, see [\sample\subtitle](#) of the SDK.



## 23.4.5 Switching the Program

### Scenario

HI\_UNF\_SUBT\_Update updates subtitle information and buffer in the subtitle module when the program is switched. For a program with multiple subtitle streams, the subtitle module provides multiple buffers based on the number of subtitle streams so that the subtitles can be switched quickly. When the program is switched, HI\_UNF\_SUBT\_Update re-allocates the buffers based on the number of subtitles in the program.

### Working Process

None

### Notes

The SCTE service provides only one subtitle stream. During program switching, data in the buffer is cleared, but the buffer does not need to be re-allocated. The program can be switched in the following two ways:

- Call HI\_UNF\_SUBT\_Update: When you switch to a program, call HI\_UNF\_SUBT\_Update to update the subtitle information for the current program without destroying or creating an instance. This method can be used when the streams contain only DVB subtitles.
- Create and destroy instances: Select a subtitle type and create a subtitle instance each time a program is to be played. Destroy the instance first when the program is to be switched. After switching to the new program, select a new subtitle type and create another subtitle instance. This method is used when the streams contain both DVB and SCTE subtitles.

### Sample

For details, see \sample\subtitle of the SDK.

## 23.4.6 Using Callback Functions

### Scenario

The subtitle module only parses subtitle stream data that is injected into the module, but does not draw the subtitles. The callback functions are designed to enable the application to draw subtitles. The subtitle module provides the following two callback functions as required:

- `typedef HI_S32 (*HI_UNF_SUBT_CALLBACK_FN)(HI_U32 u32UserData, HI_UNF_SUBT_DATA_S *pstData)`

This callback function is used to output data decoded by the subtitle module and provide necessary information for the application to draw subtitles. For details about the parameters, see [Table 23-1](#).

- `typedef HI_S32 (*HI_UNF_SUBT_GETPTS_FN)(HI_U32 u32UserData, HI_S64*p64Pts)`

This callback function is used to obtain the current system PTS to adjust the PTS contained in the subtitle streams. For details about the parameters, see [Table 23-2](#).



**Table 23-1** HI\_UNF\_SUBT\_DATA\_S parameters

No.	Parameter	Type	Description
1	enDataType	typedef enum hiUNF_SUBT_TYPE_E { HI_UNF_SUBT_TYPE_BITMAP, HI_UNF_SUBT_TYPE_TEXT, HI_UNF_SUBT_TYPE_BUTT }HI_UNF_SUBT_TYPE_E	Subtitle page type: bitmap or text.
2	enPageState	typedef enum hiUNF_SUBT_PAGE_STATE_E { HI_UNF_SUBT_PAGE_NORMAL_C ASE, HI_UNF_SUBT_PAGE_ACQUISITI ON_POINT, HI_UNF_SUBT_PAGE_MODE_CHA NGE, HI_UNF_SUBT_RESERVED }HI_UNF_SUBT_PAGE_STATE_E	Subtitle status, indicating the display mode: updating some part, updating the entire screen, update the entire subtitle page, or reserved
3	u32x	HI_U32	Horizontal coordinate of the top left pixel of a subtitle page
4	u32y	HI_U32	Vertical coordinate of the top left pixel of a subtitle page
5	u32w	HI_U32	Subtitle page width
6	u32h	HI_U32	Subtitle page height
7	u32BitWidth	HI_U32	Pixel bit width
8	u32PTS	HI_U32	Subtitle PTS
9	u32Duration	HI_U32	Subtitle page display duration (ms)
10	u32PaletteIt em	HI_U32	CLUT (palette) length
11	pvPalette	HI_VOID*	Address for storing CLUT (palette) data
12	u32DataLen	HI_U32	Subtitle data length
13	pu8SubtData	HI_U8*	Address for storing subtitle data



**Table 23-2 HI\_UNF\_SUBT\_GETPTS\_FN parameters**

No.	Parameter	Type	Description
1	u32UserData	HI_U32	User data for the callback
2	p64Pts	HI_S64*	Current system PTS

## Working Process

None

## Notes

When the CLUT is used, the pixel width is 8 bits by default. If the values of **pvPalette** and **u32PaletteItem** are valid, **pu8SubtData** specifies the index value of the CLUT, and the CLUT stores the values of the pixel components. If the pixel width of the subtitle module is 32 bits, the CLUT is not used to describe pixel components. In this case, the values of **pvPalette** and **u32PaletteItem** (0 or NULL) are invalid, and values of pixel components can be obtained in **pu8SubtData**. Each pixel is stored in the sequence of ARGB, occupying four bytes.

## Sample

For details, see \sample\subtitle of the SDK.

## 23.5 FAQs

### 23.5.1 How Are Subtitles Displayed and Synchronized?

#### Question

How are subtitles displayed and synchronized?

#### Answer

The display and synchronization of subtitles require support of the subtitle output (SO) module. The subtitle module provides callback functions to obtain the subtitle information and synchronization information (subtitle PTS), and the SO module displays and synchronizes the subtitles based on the obtained information. Note the following when using the SO module to display subtitles:

- After the subtitle instance successfully decodes data, the callback function registered by **HI\_UNF\_SUBT\_Create** is called to output the decoded subtitle content. The subtitle content is used as the data source for **HI\_UNF\_SO\_SendData** if an SO module instance has been created.



- The SO module continuously obtains the current program PTS and compares it with the subtitle PTS. If the subtitle PTS is greater than or equal to the program PTS, the SO module displays the subtitle content on the OSD. When a display timeout occurs, the SO module stops displaying the subtitle content. For details about the SO module, see the documents related to the SO module.
- These two modules work independently and share no data block.
- The reference sample is as follows:

```
HI_UNF_SO_SUBTITLE_INFO_S stSubtitleOut;
memset(&stSubtitleOut, 0, sizeof(stSubtitleOut));
if (pstData->enPageState == HI_UNF_SUBT_PAGE_NORMAL_CASE)
{
    stSubtitleOut.unSubtitleParam.stGfx.enMsgType =
    HI_UNF_SO_DISP_MSG_NORM;
}
else
{
    stSubtitleOut.unSubtitleParam.stGfx.enMsgType =
    HI_UNF_SO_DISP_MSG_ERASE;
}

stSubtitleOut.unSubtitleParam.stGfx.x = pstData->u32x;
stSubtitleOut.unSubtitleParam.stGfx.y = pstData->u32y;
stSubtitleOut.unSubtitleParam.stGfx.w = pstData->u32w;
stSubtitleOut.unSubtitleParam.stGfx.h = pstData->u32h;

stSubtitleOut.unSubtitleParam.stGfx.s32BitWidth = pstData->u32BitWidth;
if(pstData->pvPalette && pstData->u32PaletteItem)
{
    memcpy(stSubtitleOut.unSubtitleParam.stGfx.stPalette, pstData-
>pvPalette, pstData->u32PaletteItem);
}

stSubtitleOut.unSubtitleParam.stGfx.s64Pts = pstData->u32PTS;
stSubtitleOut.unSubtitleParam.stGfx.u32Duration = pstData->u32Duration;

stSubtitleOut.unSubtitleParam.stGfx.u32Len = pstData->u32DataLen;
stSubtitleOut.unSubtitleParam.stGfx.pu8PixData = pstData->pu8SubtData;
stSubtitleOut.unSubtitleParam.stGfx.u32CanvasWidth = pstData-
>u32DisplayWidth;
stSubtitleOut.unSubtitleParam.stGfx.u32CanvasHeight = pstData-
>u32DisplayHeight;
HI_UNF_SO_SendData(hSubtitleOut, &stSubtitleOut, 1000);
```



## 23.5.2 How Are Subtitles Drawn by Using Callback Functions?

### Question

How are subtitles drawn by using callback functions?

### Answer

The drawing function registered by the SO module is used to draw subtitles in synchronization with the audio and video. The following describes the two scenarios for drawing subtitles by using the HiGo as an example.

- Graphics subtitles

Graphics (bitmap) subtitles are described in pixels. During drawing, the component information of each pixel is filled in the surface data area of the graphics layer and then displayed by the display device. The pixel component information can be described by the CLUT or the decoded data. The application fills in the information based on the scenario. The reference code is as follows:

```
if(BITWIDTH_8_BITS == pstInfo->unSubtitleParam.stGfx.s32BitWidth)
{
    for (i = 0; i < pstInfo->unSubtitleParam.stGfx.h; i++)
    {
        for (j = 0; j < pstInfo->unSubtitleParam.stGfx.w; j++)
        {
            if (i * pstInfo->unSubtitleParam.stGfx.w + j > pstInfo-
>unSubtitleParam.stGfx.u32Len)
            {
                break;
            }
            u32PaletteIdx = pstInfo->unSubtitleParam.stGfx.pu8PixData[i *
pstInfo->unSubtitleParam.stGfx.w + j];
            if (u32PaletteIdx >= HI_UNF_SO_PALETTE_ENTRY)
            {
                break;
            }
            pu8Surface[i * pData[0].Pitch + 4 * j + 3]
                = pstInfo-
>unSubtitleParam.stGfx.stPalette[u32PaletteIdx].u8Alpha;
            pu8Surface[i * pData[0].Pitch + 4 * j + 2]
                = pstInfo-
>unSubtitleParam.stGfx.stPalette[u32PaletteIdx].u8Red;
            pu8Surface[i * pData[0].Pitch + 4 * j + 1]
                = pstInfo-
>unSubtitleParam.stGfx.stPalette[u32PaletteIdx].u8Green;
            pu8Surface[i * pData[0].Pitch + 4 * j + 0]
                = pstInfo-
```



```
>unSubtitleParam.stGfx.stPalette[u32PaletteIdx].u8Blue;
    }
}
}
else if(BITWIDTH_32_BITS == pstInfo->unSubtitleParam.stGfx.s32BitWidth)
{
    for (i = 0; i < pstInfo->unSubtitleParam.stGfx.h; i++)
    {
        for (j = 0; j < pstInfo->unSubtitleParam.stGfx.w; j++)
        {
            if (i * (pstInfo->unSubtitleParam.stGfx.w) + j > (pstInfo-
>unSubtitleParam.stGfx.u32Len))
            {
                break;
            }
            pu8Surface[i * pData[0].Pitch + 4 * j + 3]
                = pstInfo-
>unSubtitleParam.stGfx.pu8PixData[u32Index++];//alpha
            pu8Surface[i * pData[0].Pitch + 4 * j + 2]
                = pstInfo-
>unSubtitleParam.stGfx.pu8PixData[u32Index++];//u8Red
            pu8Surface[i * pData[0].Pitch + 4 * j + 1]
                = pstInfo-
>unSubtitleParam.stGfx.pu8PixData[u32Index++];//u8Green;
            pu8Surface[i * pData[0].Pitch + 4 * j + 0]
                = pstInfo-
>unSubtitleParam.stGfx.pu8PixData[u32Index++];//u8Blue;
        }
    }
}
```

- Text subtitles

Text subtitles are described by character strings. The subtitle data transmitted by the subtitle module indicates a string of characters that need to be drawn. The characters can be directly displayed on the display device by calling the corresponding API. Only the DVB service supports text subtitles. The reference code is as follows:

```
HI_CHAR* pszText = (HI_CHAR*)pstInfo->unSubtitleParam.stText.pu8Data;
(HI_VOID)HI_GO_TextOutEx(s_hFont, s_hSubtSurface, pszText, &rect,
                         HIGO_LAYOUT_WRAP | HIGO_LAYOUT_HCENTER | HIGO_LAYOUT_BOTTOM);
```

For different character sets, you can call the corresponding font libraries when creating the text handle by using the HiGo. The reference code is as follows:

```
s32Ret = HI_GO_CreateText("../higo/res/higo_gb2312.ubf", NULL, &s_hFont);
```

**higo\_gb2312.ubf** can be replaced by another font library.



### 23.5.3 Why Is the Callback Function for Obtaining the System PTS Required When the Subtitle Stream Contains PTS Information?

#### Question

Why is the callback function for obtaining the system PTS required when the subtitle stream contains PTS information?

#### Solution

This callback function is used during decoding of SCTE subtitles but not DVB subtitles.

The SCTE subtitle PTS is a 32-bit 90 kHz time stamp, but the system time stamp is 33-bit. When the data exceeds 32 bits, the PTS in subtitle streams indicates the lower 32 bits of the system time stamp, and the current system PTS must be obtained to adjust the PTS in subtitle streams. Otherwise, the PTS obtained from subtitle streams is far smaller than the current system PTS, and some program subtitles cannot be displayed.



# Contents

---

<b>24 Closed Caption .....</b>	<b>1</b>
24.1 Overview .....	1
24.1.1 Application Architecture .....	1
24.1.2 CC Data Transmission .....	2
24.2 Important Concepts .....	3
24.3 Features .....	4
24.4 Development Guide.....	5
24.4.1 Managing the Module .....	5
24.4.2 Configuring and Obtaining Attributes.....	7
24.4.3 Inputting and Decoding Data .....	9
24.4.4 Using the Callback Function.....	12
24.4.5 Obtaining ARIB CC Information .....	16
24.5 FAQs .....	17
24.5.1 Character Set.....	17
24.5.2 Caption Output.....	18
24.5.3 What Do I Do If the Display Sequence Is Incorrect After the Input CC User Data Is Decoded? .....	19
24.5.4 What Should I Pay Attention to When Obtaining the PTS?.....	20



# Figures

Figure 24-1 Application architecture of the CC module .....	1
Figure 24-2 DTV CC data transmission.....	2
Figure 24-3 ARIB CC data transmission.....	3
Figure 24-4 Description of ARIB CC data in the PMT table .....	3
Figure 24-5 Processing for managing the module.....	6
Figure 24-6 Process for obtaining and setting the CC instance attributes .....	8
Figure 24-7 Process for inputting and decoding user data .....	10
Figure 24-8 ATSC service process of the CC module .....	11
Figure 24-9 ARIB service process of the CC module .....	12
Figure 24-10 Process of obtaining ARIB CC information .....	16
Figure 24-11 Video frame sequencing.....	19



# Tables

Table 24-1 CC module parameters .....	5
Table 24-2 Parameters of the display function .....	13
Table 24-3 Parameters of the function for obtaining the word width .....	14
Table 24-4 Parameters of the function for transferring bit blocks .....	14
Table 24-5 Parameters of the function for outputting VBI data .....	14
Table 24-6 Parameters of the function for outputting XDS data .....	15
Table 24-7 Parameters of the function for obtaining the PTS of the current frame .....	15



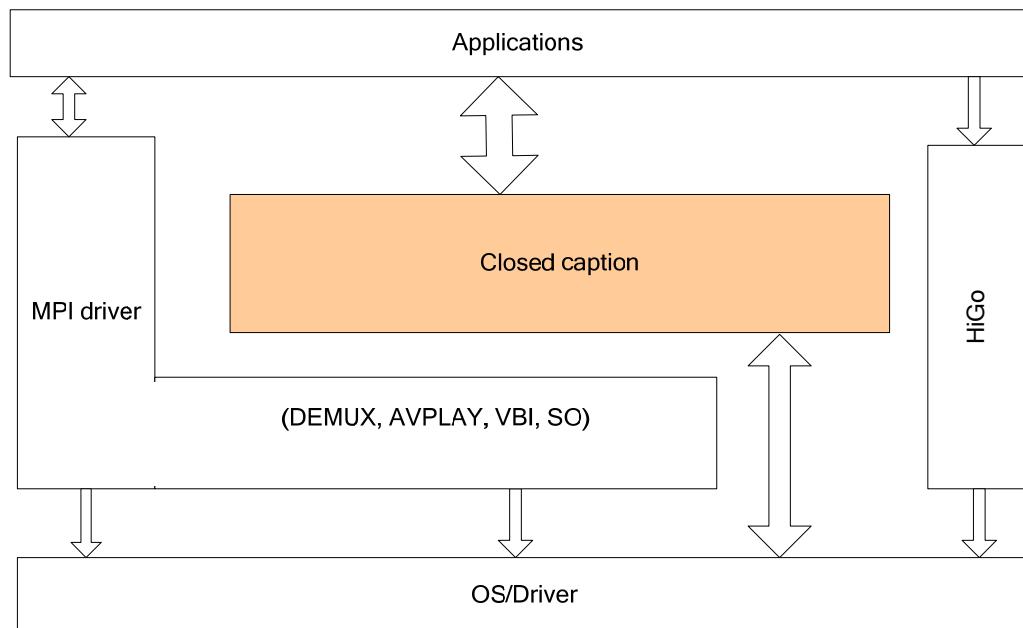
# 24 Closed Caption

## 24.1 Overview

### 24.1.1 Application Architecture

The closed caption (CC) module decodes caption stream data and MPEG-2 graphical user data in programs into captions that can be displayed. Currently CCs in the 608 (EIA-608) and 708 (EIA-708/CEA-708) standards defined by the Advanced Television Systems Committee (ATSC) and CCs in the STB-B24 standard defined by Japanese Association of Radio Industries and Businesses (ARIB) are supported. The Brazil digital television standard SBTVD-T is based on the Japanese ISDB-T standard, therefore the CC standard of the Brazil digital television is also ARIB STD-B24. [Figure 24-1](#) shows the application architecture of the CC module.

**Figure 24-1** Application architecture of the CC module



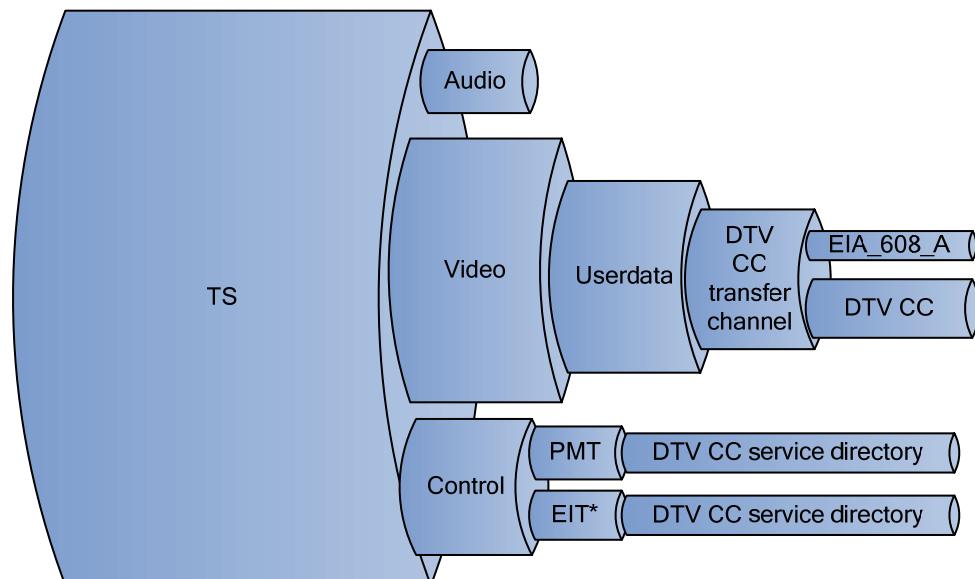


## 24.1.2 CC Data Transmission

Transmission of the ATSC CC data differs a lot from that of the ARIB CC data. Details about the transmission are as follows:

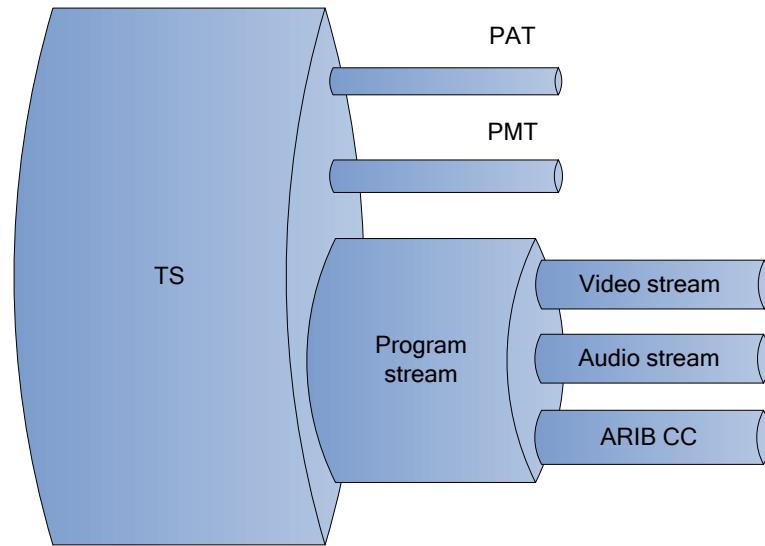
- The DTV CC data transmission is defined in ATSC A/53 and ISO/IEC 13818. The DTV CC data transfer layer transmits the DTV CC data in the encoder to the decoder in the TV receiver.
  - The DTV CC data is transmitted in the TSs as three independent parts: user data, PMT, and EIT. DTV CC service data (caption characters, window commands) is contained in the user data of MPEG-2 streams, and the service directory (service attributes) of the DTV CC caption channel is included in the PMT and EIT (if any).
  - The DTV video streams, PMT, EIT, other control data, and the audio data together compose the transmission signal of the DTV system, as shown in [Figure 24-2](#).
- The ARIB CC data transfer layer transmits CC data as independent ESs in TSs, as shown in [Figure 24-3](#).
  - The CC data is firstly packaged as independent synchronized PES. The stream ID of the PES is 0xBD, and the PES packet header must contain the PTS. Then the PES is packaged as a TS packet and transmitted in TSs.
  - If TSs contain CC data, the CC data is described in the corresponding descriptors in the PMT and EIT of the TSs. The stream type in the PMT for ESs that contain CC data is 0x06. Therefore, the TS packet that contains CC ES can be filtered out from the TSs by searching for the PID corresponding to the ES whose stream type is 0x06 in the PMT table. See [Figure 24-4](#).

**Figure 24-2** DTV CC data transmission

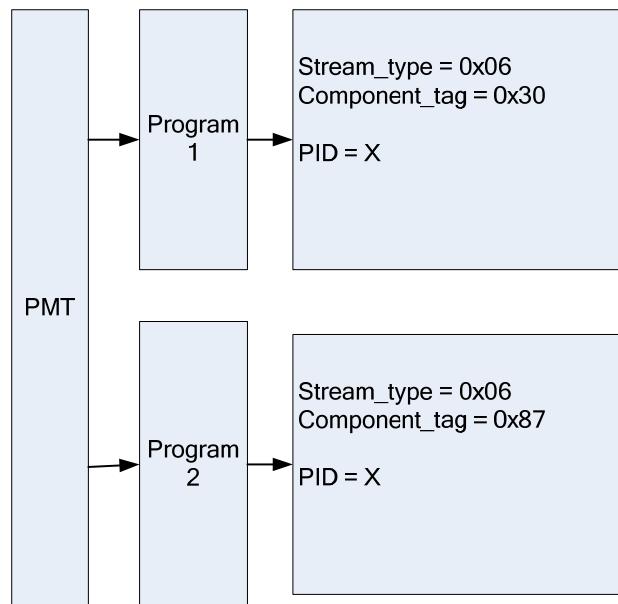




**Figure 24-3** ARIB CC data transmission



**Figure 24-4** Description of ARIB CC data in the PMT table



**NOTE**

The program needs to parse the PMT table to obtain the ARIB CC information in [Figure 24-4](#).

## 24.2 Important Concepts

[Caption channel]



- In the 608 protocol, CC information is allocated based on odd/even fields. CC data can be transmitted through nine channels: four channels for the odd fields (including CC1, CC2, TEXT1, and TEXT2), and five channels for the even fields (including CC3, CC4, TEXT3, TEXT4, and extended data services (XDS)). CC1, CC2, CC3, and CC4 are used to transmit texts in different languages. That is, these four channels support multi-language captions. CC1 is the primary caption language transfer channel. TEXT1, TEXT2, TEXT3 and TEXT4 are used to transmit information such as the weather or news report. The XDS transmits data that is used for V-CHIP (program rating).
- In the 708 standard, caption services are used to distinguish caption channels of different languages. A maximum of 63 services are supported, including 6 standard services and 57 extended services. The channel corresponding to service # 1 is pre-allocated as the main caption language channel, the channel corresponding to service #2 is the second caption language channel, and other service channels are not pre-allocated and can be allocated by program providers. The CC module supports the decoding of six standard services.

#### [Protocol stack]

The DTV CC (708 standard) transfer channels use the layer-based transmission. The data consists of five layers: transfer layer, packet layer, service layer, encoding layer, and interpretation layer. Data in the 608 standard is transmitted at the transfer layer, but data in the 708 standard is encapsulated at the preceding layers.

#### [User data]

The video user data in MPEG-2 streams is encoded into the following three levels:

- Sequence: sequence level
- Group of pictures (GOP) level
- Picture code level

The user data in this document indicates the user data that is encoded into the picture code level. This level is also the level into which CC data is encoded as specified in ATSC.

#### [Roll-up mode]

The roll-up mode is a display mode defined in the CC standards. In this mode, when all caption lines that can be displayed are displayed on the screen and the line break control word is received, the previously displayed line is not erased immediately to allow the audience to read the CC content completely. Only the line on the top is erased, and the rest lines scroll up by one line.

#### [Canvas]

The canvas is a relative coordinate system that specifies the positions to draw captions and fill area and bitmap captions. The start position is coordinates (0,0) at the upper left corner. The canvas needs to be scaled when the aspect ratio of the canvas is inconsistent with that of the system OSD.

## 24.3 Features

The CC module supports the 608 and 708 standards defined by ATSC and the ARIB STD-B24 standard of the Japanese DTV. The CC module has the following features:

- The channels for captions to be displayed can be configured, and the language can be switched.



- The foreground color and background color of texts can be customized.
- The transparency of texts can be customized.
- Text attributes such as underline and italic are supported.
- The font size can be customized.
- Attributes such as background frame and text frame can be configured.
- UTF-16LE encoding is supported. You can choose the font libraries.
- The decoded texts and bitmap captions can be output in OSD mode.
- The VBI data of CC608 can be output in VBI mode.

## 24.4 Development Guide

The CC module is used in the following scenarios:

- Module management
- Attributes configuration and obtaining
- Data input and decoding
- Callback function usage

### 24.4.1 Managing the Module

#### Scenario

Module management involves operations such as initializing and deinitializing, creating and destroying, and starting and stopping.

The following APIs are provided:

- HI\_UNF\_CC\_Init: Initializes the CC module.
- HI\_UNF\_CC\_DeInit: Deinitializes the CC module.
- HI\_UNF\_CC\_Create: Creates a CC instance.
- HI\_UNF\_CC\_Destroy: Destroys a CC instance.
- HI\_UNF\_CC\_Start: Starts a CC instance.
- HI\_UNF\_CC\_Stop: Stops a CC instance.

**Table 24-1** CC module parameters

No.	Parameter	Type	Description
1	stCCAttr	HI_UNF_CC_ATTR_S	CC attribute information. For details, see the header file.
2	pfnCCDisplay	HI_UNF_CC_DISPLAY_CB_FN	Callback function for displaying decoded CCs
3	pfnCCGetTextSize	HI_UNF_CC_GETTEXTSIZE_CB_FN	Callback function for obtaining the width and height of the displayed text

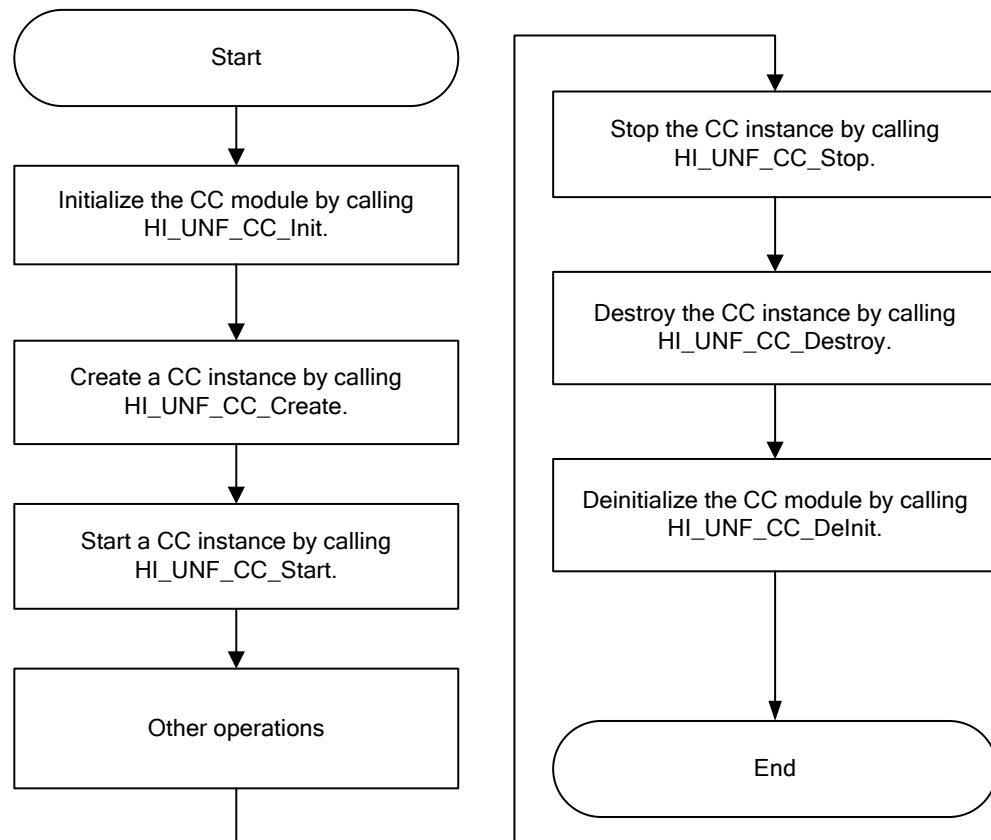


No.	Parameter	Type	Description
4	pfnBlit	HI_UNF_CC_BLIT_CB_FN	Callback function for transferring bit blocks in roll-up mode
5	pfnVBIOutput	HI_UNF_CC_VBI_CB_FN	Callback function for outputting CC608 data in VBI mode
6	pfnXDSOutput	HI_UNF_CC_XDS_CB_FN	Callback function for outputting CC608 extended data (XDS)
7	pfnCCGetPts	HI_UNF_CC_GETPTS_CB_FN	Callback function for obtaining the PTS of the current video
8	u32UserData	HI_U32	User private data transferred by applications

## Working Process

Figure 24-5 shows the process for managing the module.

**Figure 24-5** Processing for managing the module





## Notes

Note the following:

- The CC module cannot be used to initialize the DISPLAY, VO, SOUND, DEMUX, and AVPLAY devices. Therefore, you must initialize these devices before using the CC module-based applications.
- Initialize the CC module by calling `HI_UNF_CC_Init` before using its functions.
- Deinitialize the CC module by calling `HI_UNF_CC_DeInit` when the module is no longer used.
- Call the APIs of the CC module based on the sequence in [Figure 24-5](#).

## Sample

For details, see `\sample\cc` of the SDK.

### 24.4.2 Configuring and Obtaining Attributes

#### Scenario

To obtain and set the CC instance attributes, call the following APIs:

- `HI_UNF_CC_GetDefaultAttr`: Obtains default CC attributes. The default attributes of the corresponding type of CC are obtained to create a CC instance.
- `HI_UNF_CC_SetAttr`: Sets the attributes of the CC instance.
- `HI_UNF_CC_GetAttr`: Obtains the attributes of the CC instance.

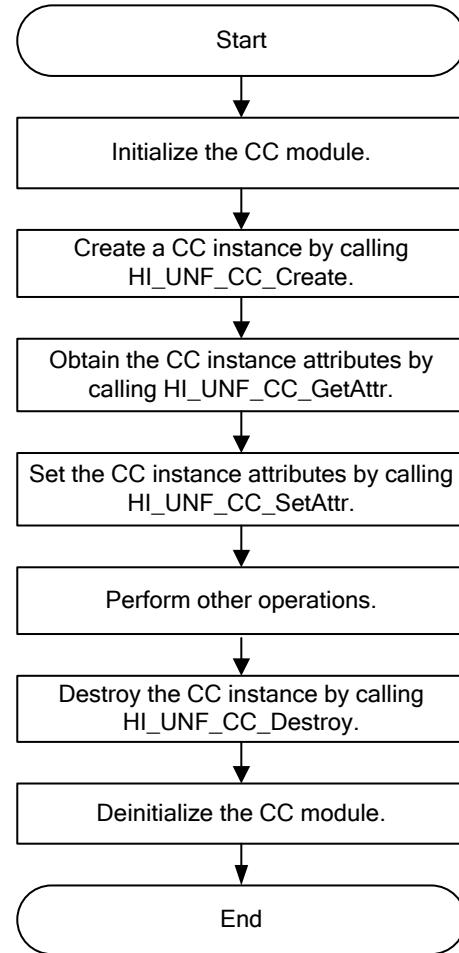
The CC instance can be internally classified into the 608 type, 708 type, and ARIB type, corresponding to the 608 and 708 standards defined by the ATSC and the STD-B24 standard defined by the ARIB. The methods of setting and obtaining the (default) attributes of the three types are different. The attributes of the 608 type are described in **stCC608ConfigParam**, the attributes of the 708 type are described in **stCC708ConfigParam**, and the attributes of the ARIB type are described in **stCCARIBConfigParam**.

#### Working Process

[Figure 24-6](#) shows the process for obtaining and setting the CC instance attributes.



Figure 24-6 Process for obtaining and setting the CC instance attributes



## Notes

The 608 part indicates the sub module which decodes the CC data defined in EIA-608. The 708 part indicates the sub module which decodes the CC data defined in EIA-708/CEA-708. Note the following:

- The CC data type in the attribute information configured by calling `HI_UNF_CC_Create` and `HI_UNF_CC_GetDefaultAttr` must be **`HI_UNF_CC_DATA_TYPE_608`**, **`HI_UNF_CC_DATA_TYPE_708`**, or **`HI_UNF_CC_DATA_TYPE_ARIB`**. Other types are not supported.
- To prevent from setting inappropriate attribute parameter values when `HI_UNF_CC_Create` is called, you are advised to obtain the default attributes by calling `HI_UNF_CC_GetDefaultAttr`, and then modify the attributes as required.
- When `HI_UNF_CC_GetAttr` is called, and the obtained **`enCCDataType`** is **`HI_UNF_CC_DATA_TYPE_608`**, the attributes are described in **`stCC608ConfigParam`** of **`unCCConfig`**. If the obtained **`enCCDataType`** is **`HI_UNF_CC_DATA_TYPE_708`**, the attributes are described in **`stCC708ConfigParam`** of **`unCCConfig`**. If the obtained **`enCCDataType`** is **`HI_UNF_CC_DATA_TYPE_ARIB`**, the attributes are described in **`stCCARIBConfigParam`** of **`unCCConfig`**.



- When HI\_UNF\_CC\_SetAttr is called, you must set **enCCDataType** to specify a data enumeration type first. The attribute configuration must be based on the data type specified in **enCCDataType**.
- Ensure that a CC instance has been created before calling HI\_UNF\_CC\_GetAttr and HI\_UNF\_CC\_SetAttr.
- After HI\_UNF\_CC\_Create is called, you can obtain the set the attributes any time by calling HI\_UNF\_CC\_GetAttr and HI\_UNF\_CC\_SetAttr before calling HI\_UNF\_CC\_Destroy.

## Sample

For details, see \sample\cc of the SDK.

### 24.4.3 Inputting and Decoding Data

#### Scenario

Data is input into the CC module and then decoded. This is the most important process of the CC module. The operations are performed after the CC module is initialized and a CC instance is created and started. The related APIs are as follows:

- HI\_UNF\_CC.InjectUserData
- HI\_UNF\_CC.InjectPESData

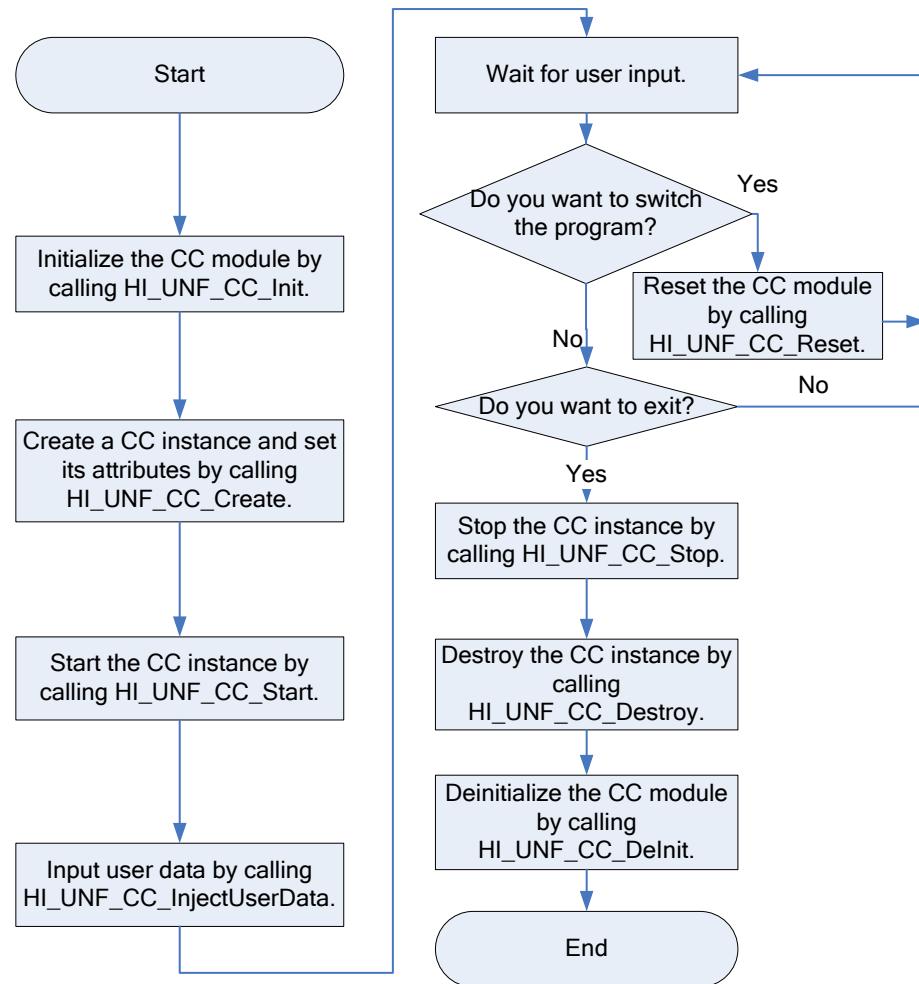
HI\_UNF\_CC.InjectUserData can be used to decode the user data in the video (CC data in the ATSC standard), and HI\_UNF\_CC.InjectPESData can be used to decode caption PES data in the program (CC data in the ARIB STD-B24 standard).

#### Working Process

Figure 24-7 shows the process for inputting and decoding user data.



**Figure 24-7** Process for inputting and decoding user data



Perform the following steps:

- Step 1** Initialize the CC module by calling HI\_UNF\_CC\_Init.
- Step 2** Obtain default CC attributes by calling HI\_UNF\_CC\_GetDefaultAttr.
- Step 3** Create a CC instance by calling HI\_UNF\_CC\_Create.
- Step 4** Start the CC instance by calling HI\_UNF\_CC\_Start.
- Step 5** Input user data into the CC instance by calling HI\_UNF\_CC.InjectUserData, or inject PES data into the CC instance by calling HI\_UNF\_CC.InjectPESData.
- Step 6** Wait for user input. If you want to exit, go to step 6; if you want to switch the program, reset the CC module by calling HI\_UNF\_CC\_Reset.
- Step 7** Stop the CC instance by calling HI\_UNF\_CC\_Stop.
- Step 8** Destroy the CC instance by calling HI\_UNF\_CC\_Destroy.
- Step 9** Deinitialize the CC module by calling HI\_UNF\_CC\_Deinit.

----End



Figure 24-8 and Figure 24-9 show the service process of the CC module.

**Figure 24-8** ATSC service process of the CC module

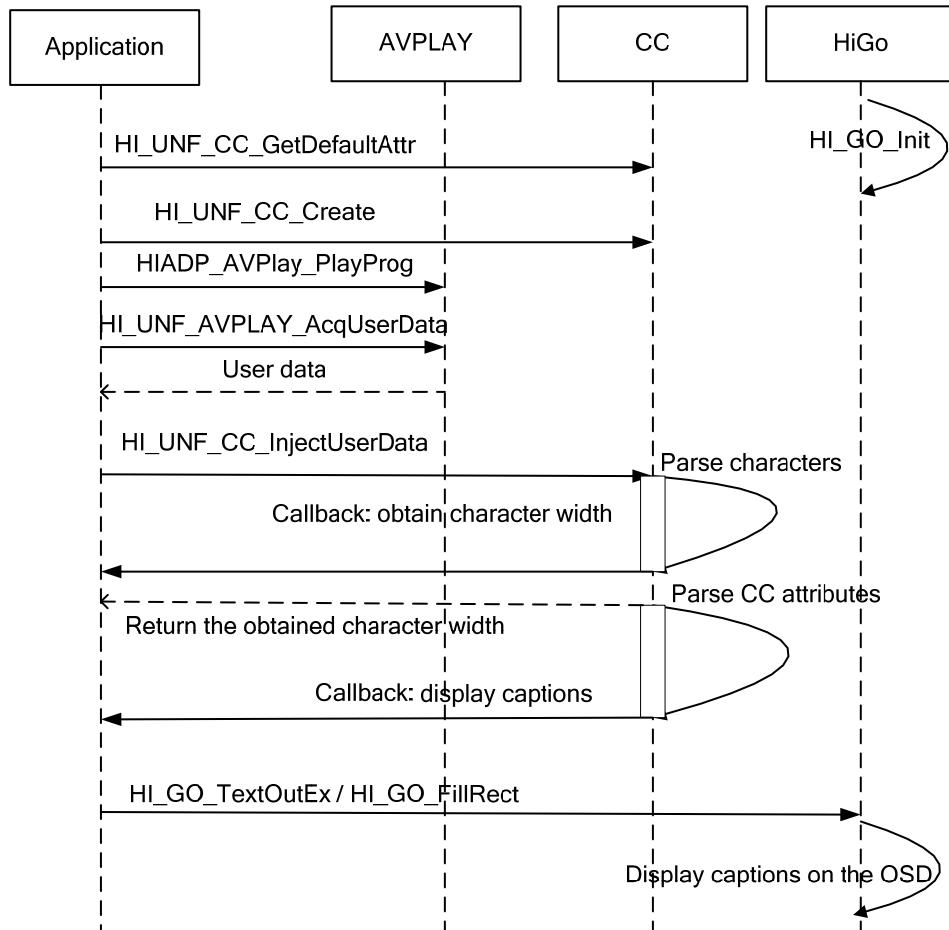
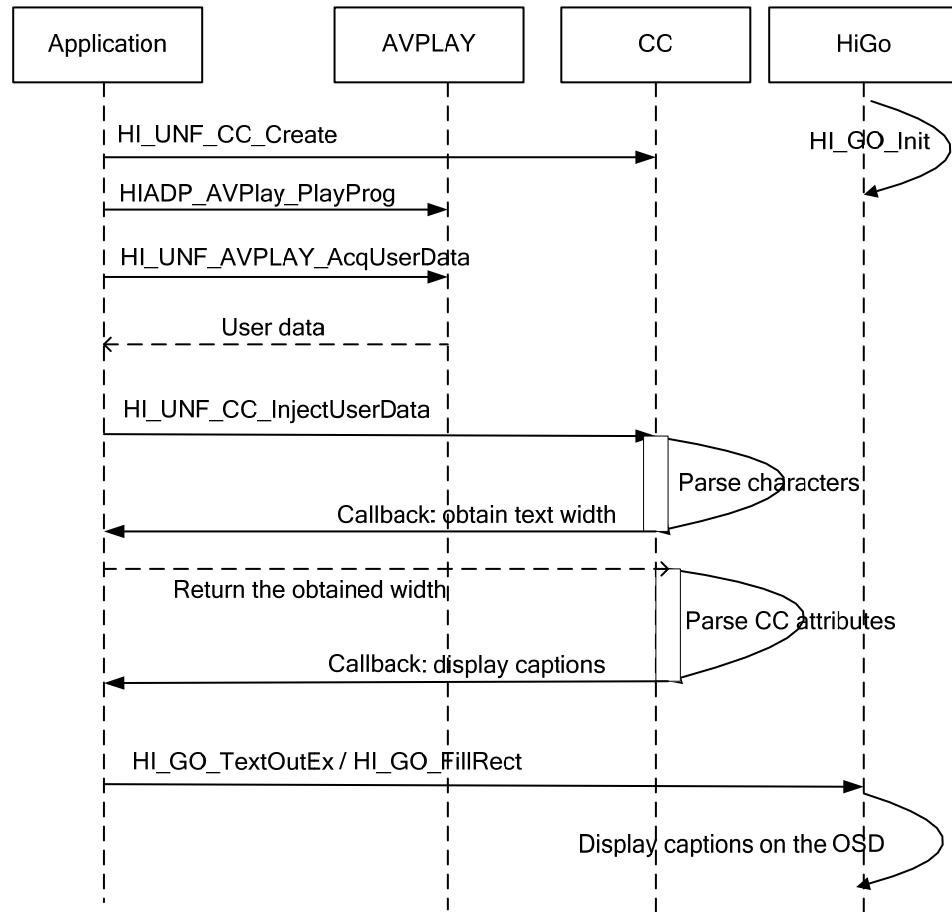




Figure 24-9 ARIB service process of the CC module



## Notes

The parameters of `HI_UNF_CC_InjectUserData` in step 4 must be set based on the display sequence of video frames. For details, see section [24.5.3 "What Do I Do If the Display Sequence Is Incorrect After the Input CC User Data Is Decoded?"](#)

If the type of the CC instance to be created is `HI_UNF_CC_DATA_TYPE_608` or `HI_UNF_CC_DATA_TYPE_708`, you can call `HI_UNF_CC_InjectUserData` to inject data; if the type of the created CC instance is `HI_UNF_CC_DATA_TYPE_ARIB`, you can call `HI_UNF_CC_InjectPESData` to inject data.

## Sample

For details, see `\sample\cc` of the SDK.

### 24.4.4 Using the Callback Function

## Scenario

Six callback functions are used in the CC module, which are described as follows:



- Display function: `typedef HI_S32 (*HI_UNF_CC_DISPLAY_CB_FN)(HI_U32 u32Userdata, HI_UNF_CC_DISPLAY_PARAM_S *pstCCdispalyParm)`. For details about the parameters, see [0](#).
- Function for obtaining the word width: `typedef HI_S32 (*HI_UNF_CC_GETTEXTSIZE_CB_FN)(HI_U32 u32Userdata, HI_U16 *u16Str, HI_S32 s32StrNum, HI_S32 *ps32Width, HI_S32 *ps32Heighth);`  
The start position for CC character drawing is obtained from streams. The purpose of obtaining the word width is to determine the positions of other characters after the first character. For details about the parameters, see [Table 24-3](#).
- Function for transferring bit blocks: `typedef HI_S32 (*HI_UNF_CC_BLIT_CB_FN)(HI_U32 u32UserData, HI_UNF_CC_RECT_S *pstSrcRect, HI_UNF_CC_RECT_S *pstDstRect);`  
This function is used to transfer captions on the OSD in roll-up mode in the 608 standard. It is used only in roll-up mode. For details about the parameters, see [Table 24-4](#).
- Function for outputting VBI data: `typedef HI_S32 (*HI_UNF_CC_VBI_CB_FN)(HI_U32 u32UserData, HI_UNF_CC_VBI_DADA_S *pstVBIOddDataField1, HI_UNF_CC_VBI_DADA_S *pstVBIEvenDataField2);`  
This function is used to output CC608 VBI data. For details about the parameters, see [Table 24-5](#).
- Function for outputting XDS data: `typedef HI_S32 (*HI_UNF_CC_XDS_CB_FN)(HI_U8 u8XDSClass, HI_U8 u8XDSPacketType, HI_U8 *pu8Data, HI_U8 u8DataLen)`  
This function is used to output CC608 XDS data. For details about the parameters, see [Table 24-6](#).
- Function for obtaining the PTS of the current frame: `typedef HI_S32 (*HI_UNF_CC_GETPTS_CB_FN)(HI_U32 u32UserData, HI_S64 *ps64CurrentPts)`  
This function is used to synchronize the ARIB CC output. For details about the parameters, see [Table 24-7](#).

**Table 24-2** Parameters of the display function

No	Parameter	Description
1	u32Userdata	User private data transmitted from the application to the callback function, transferred by <code>HI_UNF_CC_Create</code>
2	pstCCDispalyParam	<code>enOpt</code> : operation type during display. Texts, filling areas, and bitmaps can be displayed. <code>u32DisplayWidth</code> : display canvas width <code>u32DisplayHeight</code> : display canvas height <code>stRect</code> : position of the displayed content <code>unDispParam</code> : display parameter (text parameter, filling area parameter, or bitmap parameter), corresponding to the three values of <code>enOpt</code>



**Table 24-3** Parameters of the function for obtaining the word width

No	Parameter	Type	Description
1	u32Userdata	HI_U32	User private data transmitted from the application to the callback function, transferred by HI_UNF_CC_Create
2	pu16Str	HI_U16 *	Character string of which the word width is to be obtained in the CC module
3	s32StrNum	HI_S32	Length of the character string
4	ps32Width	HI_S32 *	Output width of the specified character string
5	ps32Heighth	HI_S32 *	Output height of the specified character string

**Table 24-4** Parameters of the function for transferring bit blocks

No	Parameter	Type	Description
1	u32UserData	HI_U32	User private data transmitted from the application to the callback function, transferred by HI_UNF_CC_Create
2	pstSrcRect	HI_UNF_CC_RECT_S *	Source position for the bit block transfer
3	pstDstRect	HI_UNF_CC_RECT_S *	Target position for the bit block transfer

**Table 24-5** Parameters of the function for outputting VBI data

No.	Parameter	Type	Description
1	u32UserData	HI_U32	User private data transmitted from the application to the callback function, transferred by HI_UNF_CC_Create
2	pstVBIOddDataField1	HI_UNF_CC_VBI_DADA_S *	VBI data structure, odd field data u8FieldParity: odd/even field information of the VBI data u8Data1: first byte in the field data byte pair u8Data2: second byte in the field data byte pair



No.	Parameter	Type	Description
3	pstVBIEvenDataField2	HI_UNF_CC_VBI_DADA_S *	VBI data structure, even field data u8FieldParity: odd/even field information of the VBI data u8Data1: first byte in the field data byte pair u8Data2: second byte in the field data byte pair

**Table 24-6** Parameters of the function for outputting XDS data

No.	Parameter	Type	Description
1	u8XDSClass	HI_U8	XDS class
2	u8XDSPacketType	HI_U8	XDS packet type
3	pu8Data	HI_U8 *	XDS data
4	u8DataLen	HI_U8	XDS data length

**Table 24-7** Parameters of the function for obtaining the PTS of the current frame

No.	Parameter	Type	Description
1	u32UserData	HI_U32	User private data transmitted from the application to the callback function, transferred by HI_UNF_CC_Create
2	ps64CurrentPts	HI_S64 *	PTS of the current video frame

## Working Process

None

## Notes

None

## Sample

For details about the sample, see [/sample/cc](#) of the SDK. For details about the decoding of XDS packets, see [/sample/cc/sample\\_cc\\_xds.c](#) of the SDK.



## 24.4.5 Obtaining ARIB CC Information

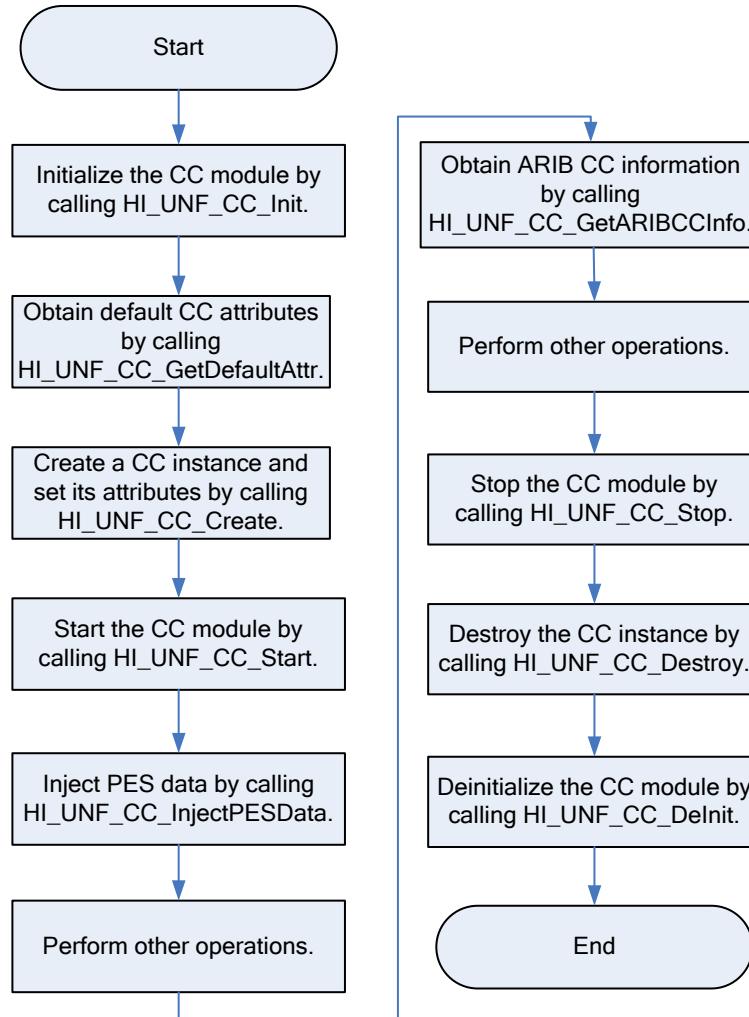
### Scenario

To obtain the ARIB CC information, you need to initialize the CC module, create and start an instance, and inject ARIB CC data. The related API is as follows:

`HI_UNF_CC_GetARIBCCInfo`

### Working Process

**Figure 24-10** Process of obtaining ARIB CC information



### Notes

None



## Sample

For details, see `/sample/cc` in the SDK.

## 24.5 FAQs

### 24.5.1 Character Set

#### 24.5.1.1 Does the CC Module Support Multiple Languages?

##### Problem Description

Does the CC module support multiple languages?

##### Solution

The basic ATSC CC character set includes not only the ASCII characters but also the ISO 8859-1 Latin character set. It supports multiple languages such as Spanish, French, Portuguese, German, Danish, and Korean. The ARIB CC character set supports languages such as Japanese and Brazilian Portuguese. To support multiple languages, the CC module encodes the characters in the CC character set by using UTF-16. The display requires cooperation with applications. For example, a text output object can be created by using the Korean library to display Korean characters. Applications can create text output objects by using different font libraries. Take the HiGo as an example. The reference code is as follows:

```
if(HI_SUCCESS != HI_GO_CreateText( "DroidSansMono.ttf ",NULL,  
&s_hFontOut ))  
{  
    printf( "failed to create the font: DroidSansMono.ttf!,ret  
=0x%x\n",s32Ret );  
    goto RET;  
  
}  
}
```



#### CAUTION

If the SDK supports vector font libraries, **DroidSansMono.ttf** can be used, which is a fixed-width font library. If the SDK does not support vector font libraries, the dot-matrix font library **higo\_gb2312.ubf** can be used, which is a variable-width font library.

**DroidSansMono.ttf** is better than **higo\_gb2312.ubf** in terms of display effect. Note that the preceding font libraries are single-byte font libraries and do not support the ISO 8859-1 Latin character set. Therefore, some conversion is required to support these libraries in the sample. In the following sample, you can convert the UTF-16 encoding into UTF-8 encoding. You can also use your own font libraries to support Unicode characters.

```
if (c < 0x80) {first = 0; len = 1;}  
else if (c < 0x800) {first = 0xc0;len = 2;}
```



```
else if (c < 0x10000) {first = 0xe0;len = 3;}
else if (c < 0x200000) {first = 0xf0;len = 4;}
else if (c < 0x4000000) {first = 0xf8;len = 5;}
else {first = 0xfc;len = 6;}
for (i = len - 1; i > 0; --i)
{
    outbuf[i] = (c & 0x3f) | 0x80;
    c >>= 6;
}
outbuf[0] = c | first;
```

### 24.5.1.2 How Do I Choose Font Libraries?

#### Problem Description

How do I choose font libraries?

#### Solution

There are eight fonts specified in CEA-708-B. However, similar fonts can be used to replace them. In the CC regulations, the position of a character is determined by specifying the row and column. As the character width required for each font library differs, it is recommended that fixed-width font libraries be used to achieve better display effect.

### 24.5.2 Caption Output

#### 24.5.2.1 How Do I Output Decoded CCs to the OSD?

#### Problem Description

How do I output decoded CCs to the OSD?

#### Solution

The drawing of CC characters is implemented by using the callback functions. There are three situations: one is to draw a character or a group of specific characters, one is to fill a blank region, and the last one is to draw a bitmap. **enOpt** in the display parameters of the callback function is used to distinguish these situations.

The information required during drawing, such as the position, character string, character attributes, and background color, is specified by the parameters of the callback function. The application displays the captions on the OSD.

For details about the reference code, see **SDK\sample\cc\sample\_cc\_out.c**.

#### 24.5.2.2 Why Is ATSC CC Data Synchronization Not Required?

#### Problem Description

Why is ATSC CC data synchronization not required?



## Solution

The user data obtained from the MPEG-2 images does not contain the PTS value. Therefore the CC data cannot be synchronized based on the PTS. Actually in most cases ATSC CC providers insert DTV CCs into the streams in real time. That is, captions are transmitted before the display, and the CC controlling command information is used to draw the characters.

### 24.5.3 What Do I Do If the Display Sequence Is Incorrect After the Input CC User Data Is Decoded?

#### Problem Description

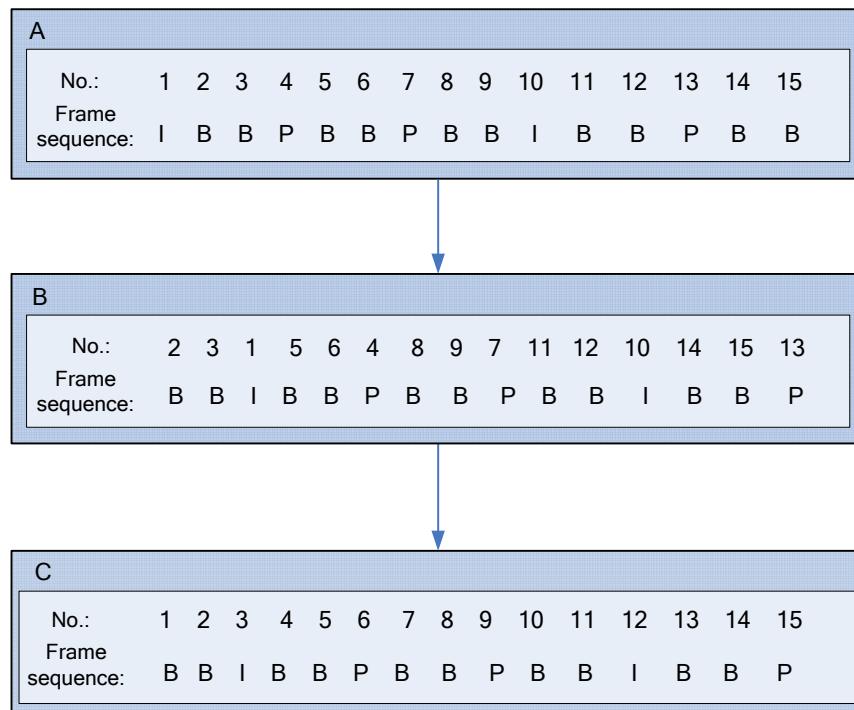
After the CC data is input by calling HI\_UNF\_CC.InjectUserData, the display of the decoded data is disordered.

#### Cause Analysis

According to the ATSC standards, when CC data is encoded into the user data streams of the graphics user level, fixed bandwidth is requested in each image frame for the transfer channel of DTV CCs. CC program providers insert CC data into each image frame based on the bandwidth of the CC channel. Note that data must be input into the CC module based on the display sequence but not the decoding sequence. Otherwise, the CC decoding is abnormal.

Take the MPEG-2 video streams as an example to illustrate the data sequencing, as shown in [Figure 24-11](#).

**Figure 24-11** Video frame sequencing





The letter numbers in [Figure 24-11](#) are described as follows:

- A: Indicates the initial frame sequence of the video, that is, the decoding sequence.
- B: Indicates the video frame display output sequence arranged based on the frame type of frames in A. The number corresponding to each frame indicates the position of the frame in the decoding sequence.
- C: Indicates the sequence of frames to be output for display, that is, the display sequence. The number corresponding to each frame indicates the position of the frame in the display sequence.

 **NOTE**

I, B, and P in [Figure 24-11](#) indicate the I frame (key frame), P frame (predicted differential frame), and B frame (bidirectional differential frame) respectively.

## Solution

When injecting data into the CC module by calling HI\_UNF\_CC.InjectUserData, ensure that the data is injected in the display sequence of video frames. The user data injected by calling HI\_UNF\_AVPLAY.AcqUserData of the HiSilicon AVPLAY is output in display sequence, whereas the user data injected by calling HI\_UNF\_AVPLAY.Invoke needs to be adjusted as the display sequence and then output to the CC module. For details about this API, see [\SDK\sample\cc\sample\\_cc.c](#).

### 24.5.4 What Should I Pay Attention to When Obtaining the PTS?

#### Problem Description

What should I pay attention to when obtaining the PTS?

## Solution

Obtaining the PTS is to ensure synchronous output of ARIB CCs because the ARIB CC data is transmitted by PES. Note that the PTS transferred by the application must be in the unit of kHz.



# Contents

---

<b>25 HiPlayer .....</b>	<b>1</b>
25.1 Overview .....	1
25.2 Important Concepts .....	3
25.3 Features .....	4
25.4 Development Guide.....	6
25.4.1 Basic Playback Process.....	6
25.4.2 Outputting Subtitles .....	21
25.4.3 Setting Attributes .....	25
25.4.4 Implementing the File Parser/Protocol.....	29
25.4.5 HTTP Playback .....	36
25.4.6 HTTPS Bidirectional Certification .....	37
25.4.7 HLS Playback .....	39
25.4.8 Seamless M3U9 Playback.....	43
25.4.9 RTSP Playback.....	44
25.4.10 MMS Playback.....	45
25.4.11 RTMP Playback.....	46
25.4.12 Mounting ISO Files in UDF Format .....	46
25.4.13 Playback of 3D Media Sources .....	51
25.4.14 HiDrmEngine .....	52



# Figures

<b>Figure 25-1</b> Application architecture of the HiPlayer.....	2
<b>Figure 25-2</b> Position of protocols in the HiPlayer data streams .....	4
<b>Figure 25-3</b> HiPlayer playback process.....	7
<b>Figure 25-4</b> Creating a HiPlayer using an external AVPLAY .....	12
<b>Figure 25-5</b> HiPlayer local file list .....	13
<b>Figure 25-6</b> Asking the user whether to play from the bookmark .....	13
<b>Figure 25-7</b> Rolling lyrics .....	24
<b>Figure 25-8</b> Process for setting the attributes .....	25
<b>Figure 25-9</b> Implementing the file parser .....	29
<b>Figure 25-10</b> Selecting the HiPlayer parser.....	35
<b>Figure 25-11</b> HLS media source.....	40
<b>Figure 25-12</b> HLS file index.m3u8 .....	40
<b>Figure 25-13</b> HLS file 01.m3u8 .....	41
<b>Figure 25-14</b> Running <b>make menuconfig ARCH=arm</b> .....	48
<b>Figure 25-15</b> File systems .....	49
<b>Figure 25-16</b> CD-ROM/DVD Filesystems.....	50
<b>Figure 25-17</b> Playback process for 3D media sources.....	52
<b>Figure 25-18</b> HiDrmEngine application architecture .....	53
<b>Figure 25-19</b> HiDrmEngine working process.....	54
<b>Figure 25-20</b> HiDrmEngine DRM request process .....	56



## Tables

<b>Table 25-1</b> HiPlayer version features.....	2
<b>Table 25-2</b> HiPlayer playback control .....	11
<b>Table 25-3</b> Relationship among the MMS URL prefix, transmission protocol, and HiPlayer operation.....	45
<b>Table 25-4</b> Relationship among the RTMP URL prefix, transmission protocol, and encryption mode.....	46



# 25 HiPlayer

## 25.1 Overview

The HiPlayer is a component developed based on the SDK. [Figure 25-1](#) shows the application architecture of the HiPlayer.

- The file parser parses media files.
- The SDK implements the decoding, synchronization, and output of audio and video data, and subtitle synchronization.
- The HiPlayer calls the parser to parse audio and video frames and subtitle data.

The HiPlayer schedules media data, controls playback of media data, and interacts with applications. It has the following functions:

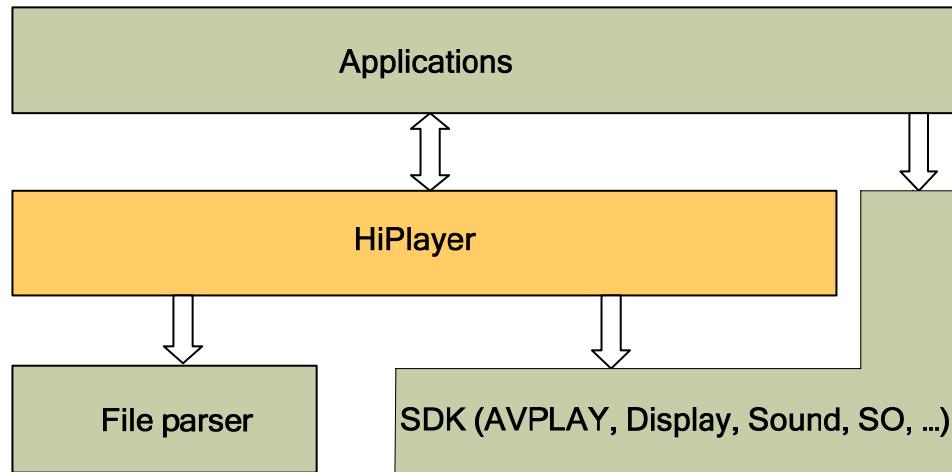
- Plays local files. The file formats supported by the HiPlayer are determined by the file parser that is registered with the HiPlayer by applications (applications developed based on the HiPlayer).
- Plays online media. Network protocols such as Hypertext Transfer Protocol (HTTP), Real Time Streaming Protocol (RTSP), and Multimedia Messaging Service (MMS) are supported. The supported network protocols are determined by the file parser that is registered with the HiPlayer by applications.
- Supports the operations such as play, fast-forward, rewind, pause, stop, and seek (seek to a specified time point).
- Loads a file parser.

### NOTE

Applications call the HiPlayer APIs to implement playback control operations, and call SDK UNF APIs to set attributes such as the output volume and position of the video window.



**Figure 25-1** Application architecture of the HiPlayer



The HiPlayer has three versions: basic version, normal version, and full version. The features supported by each version differ. Generally, features supported by the basic version are a subset of that supported by the normal version, and the features supported by the normal version are a subset of that supported by the full version. See [Table 25-1](#). For details about each feature, see the following sections.

**Table 25-1** HiPlayer version features

HiPlayer Feature	SDK Version				
	HiSTBLinux V100R001			HiSTBLinux V100R002	HiSTBAndroid V500R001
	Basic	Normal	Full	Full	Default
File format	Mainstream formats	Most formats	Most formats	Most formats	Most formats
FFmpeg audio software decoding	No	No	Yes	Yes	Yes
FFmpeg video software decoding	No	No	No	Yes	Yes
HTTP	No	Yes	Yes	Yes	Yes
HTTPS	No	No	Yes	Yes	Yes
HLS	No	Yes	Yes	Yes	Yes
M3U9	No	No	Yes	Yes	Yes



HiPlayer Feature	SDK Version				
	HiSTBLinux V100R001			HiSTBLinux V100R002	HiSTBAndroid V500R001
	Basic	Normal	Full	Full	Default
RTSP	No	No	Yes	Yes	Yes
MMS	No	No	Yes	Yes	Yes
RTMP	No	No	Yes	Yes	Yes
Bluray	No	No	No	Yes	Yes
3D file	No	No	No	Yes	Yes
HiDrmEngine	No	No	Yes	Yes	Yes

#### NOTE

The HiPlayer basic version supports only local playback. The basic and normal versions apply to low-cost solutions that impose serious limitations on the memory and flash memory.

To check the current HiPlayer version on Linux, perform the following steps:

**Step 1** Run **make menuconfig** in the SDK root directory.

**Step 2** Select **Component**.

**Step 3** Select **HiPlayer Support**.

Then you can view the version information. For example, **HiPlayer Full Support** indicates the full version, and **HiPlayer Normal Support** indicates the normal version.

----End

## 25.2 Important Concepts

[File parser]

A media file may contain multiple audio/video streams and subtitle streams. The file parser parses media files and sorts the audio/video frames and subtitle data. Then the HiPlayer transmits audio/video streams to the AVPLAY for decoding and playback, and subtitle data to the SO module for synchronous output.

The file parser is a module that parses media file formats (such as .avi and .mp4) and sorts audio/video frames and subtitle data. Applications can implement their own file parsers. For details, see section [25.4.3 "Setting Attributes."](#)

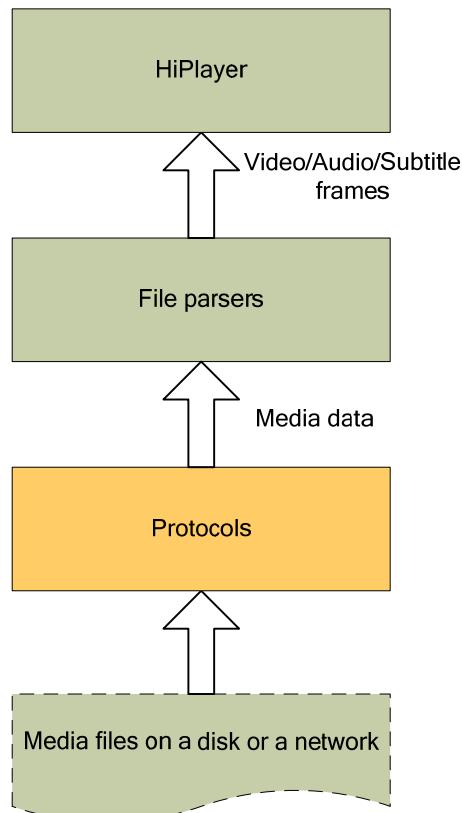
[Protocol]

The file parser must read data while parsing a file. For example, the AVI parser is used to parse media data when a local AVI file and an AVI file on the HTTP server are played, but the



used protocols are different. When a local file is played, the File protocol (protocol that operates files by using standard open/read/close interfaces) is used to read data from the disk. When a file on the HTTP server is played, the HTTP protocol is used to download data from the network. Modules similar to the File protocol or HTTP protocol that implement the data read function in the HiPlayer are called protocols. Common protocols include the File, HTTP, HTTPS, RTMP, and MMS protocols. Application also can implement their own proprietary protocols. For details, see section [25.4.3 "Setting Attributes."](#)

**Figure 25-2** Position of protocols in the HiPlayer data streams



## 25.3 Features

The HiPlayer provides the following functions:

- Controls the playback of local or network media files. The following APIs are provided:
  - HI\_SVR\_PLAYER\_Init: Initializes the HiPlayer.
  - HI\_SVR\_PLAYER\_RegisterDynamic: Registers a file parser or protocol.
  - HI\_SVR\_PLAYER\_Create: Creates a HiPlayer instance.
  - HI\_SVR\_PLAYER\_RegCallback: Registers a HiPlayer event callback function.
  - HI\_SVR\_PLAYER\_SetMedia: Sets the address for the media file to be played.
  - HI\_SVR\_PLAYER\_GetFileInfo: Obtains media file information.
  - HI\_UNF\_SO\_RegOnDrawCb: Sets the callback function for outputting subtitles.
  - HI\_SVR\_PLAYER\_Play: Starts playing.



- HI\_SVR\_PLAYER\_Pause: Pauses the playback.
- HI\_SVR\_PLAYER\_Resume: Resumes the playback.
- HI\_SVR\_PLAYER\_TPlay: Fast forwards/Rewinds a file.
- HI\_SVR\_PLAYER\_Seek: Seeks a position to play.
- HI\_SVR\_PLAYER\_Stop: Stops playback.
- HI\_SVR\_PLAYER\_Invoke: Implements various functions by specifying **InvokeID** (interface parameter). For details, see the instances in the following sections.
  - [Obtaining MetaData](#)
  - [MetaData is stored by key value pairs. For details about how to obtain the data, see the implementation of HI\\_SVR\\_META\\_PRINT.](#)
  - [Setting the Network Buffer Size and Thresholds by Byte](#)
  - [Setting the Buffer Size and Thresholds Based on the Playback Duration](#)
  - [Setting the HiPlayer Operation When the Network Buffer Underloads](#)
  - [Obtaining Network Bandwidth During Network Playback](#)
  - [Setting Cookie Information](#)
  - [Playing HLS Streams](#)

When this API is called, the HiPlayer transparently transmits it to the current file parser.

- HI\_SVR\_PLAYER\_Destroy: Destroys a HiPlayer instance.
- HI\_SVR\_PLAYER\_UnRegisterDynamic: Deregisters the file parser.
- HI\_SVR\_PLAYER\_Deinit: Deinitializes the HiPlayer.
- Outputs subtitles. The following APIs are provided:
  - HI\_SVR\_PLAYER\_GetParam: Obtains the AVPLAY and SO module handles used by the HiPlayer.
  - HI\_UNF\_SO\_RegOnDrawCb: Registers the callback function for outputting subtitles.
- Sets the HiPlayer attributes. The HiPlayer attributes, such as the playback volume, video display position, and mute status, are not encapsulated. Users can obtain the AVPLAY, SO, and window device handles by using the HiPlayer APIs and then set the attributes by calling corresponding UNF APIs. The following APIs are provided:
  - HI\_SVR\_PLAYER\_GetParam: Obtains the sound track handle used by the HiPlayer.
  - HI\_UNF SND\_SetTrackWeight: Sets the volume of the HiPlayer.
  - HI\_UNF SND\_SetMute: Sets the mute state of the HiPlayer.
  - HI\_UNF VO\_GetWindowAttr: Obtains the window attributes.
  - HI\_UNF VO\_SetWindowAttr: Sets the window attributes to specify the video display region.
- Mounts the parser or protocol. The following APIs are provided:
  - HI\_SVR\_PLAYER\_RegisterDynamic: Registers a file parser.
  - HI\_SVR\_PLAYER\_UnRegisterDynamic: Deregisters the file parser.

Other functions are described in the **Sample** section of each chapter or [sample/localplay/sample\\_localplay.c](#).



## 25.4 Development Guide

After the SDK is decompressed, the relative path for the HiPlayer sample program is **sample/localplay**, and that for the HiPlayer component is **source/component/hiplayer**. After the SDK is compiled, the HiPlayer header files (with hi-svr as the prefix) are copied to the **pub/include** directory, the HiPlayer library files are copied to the **pub/lib/** directory. The sample (**sample\_localplay**) is also compiled. After modifying the HiPlayer sample, you can recompile it by running the following command:

```
make sample_clean M=localplay; make sample M=localplay;
```

### 25.4.1 Basic Playback Process

#### Scenario

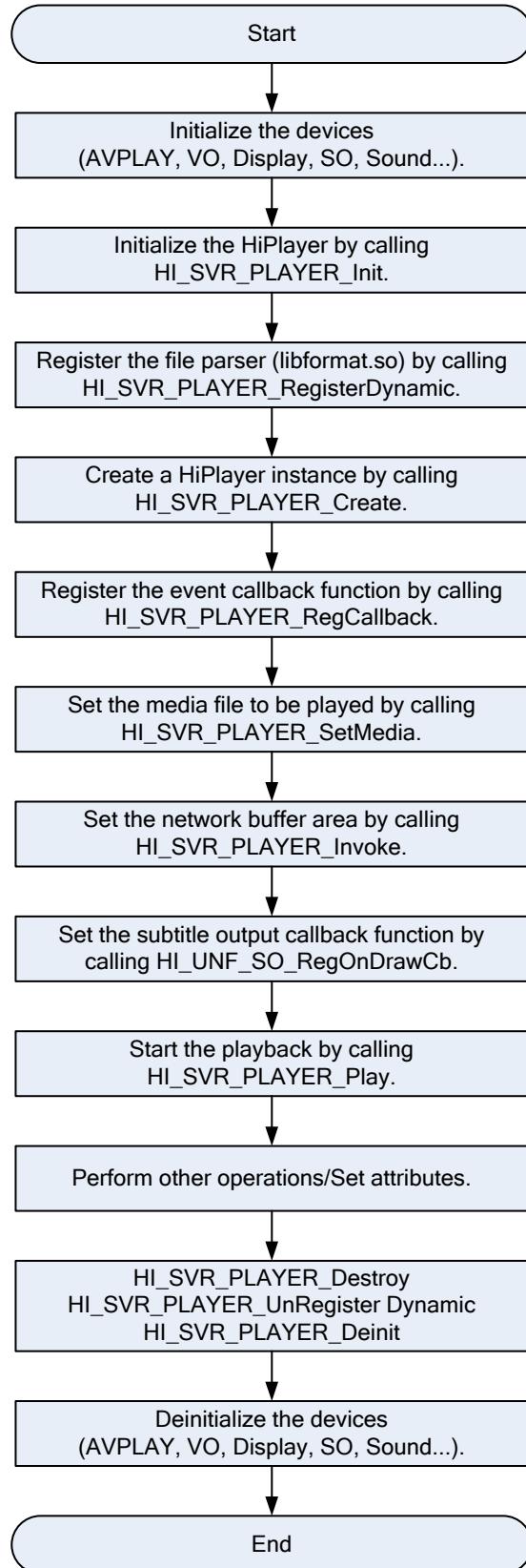
The HiPlayer is used to play a local or network media file.

#### Working Process

[Figure 25-3](#) shows the HiPlayer playback process.



Figure 25-3 HiPlayer playback process





The process details are as follows:

- Initialize the HiPlayer and required devices.
  - Initialize playback-related devices including the AVPLAY, Display, VO, SO, and Sound devices. The HiPlayer creates and uses, but does not initialize these device resources. See the implementation of the function `deviceInit` in [sample/localplay/sample\\_localplay.c](#).
  - Initialize the HiPlayer. After initializing the HiPlayer, you can call the HiPlayer APIs to create a player and specify the media file for playing.
- Register a file parser.

The application registers file parsers with the HiPlayer by calling the API provided by the HiPlayer. When opening a file, the HiPlayer traverses all registered file parsers and finds the one that can parse the file to be played. **libformat.so** provided in the SDK is a dynamic library for parsers. It implements parsing of subtitle files in formats such as lrc and srt and encapsulating of file protocols.
- Register an event callback function.

After the HiPlayer is initialized, create a player instance by calling `HI_SVR_PLAYER_Create`, and register the playback event callback function with the player. The HiPlayer has the following callback events:

  - `HI_SVR_PLAYER_EVENT_STATE_CHANGED`: HiPlayer status change event. The parameter of this event is the current player status. The HiPlayer supports the play, pause, fast forward, rewind, and stop operations. The APIs for these operations are implemented in asynchronous mode, and the user cannot obtain the internal playback status based on the return values of the APIs. When the status changes, the HiPlayer notifies the application by using this callback function.
  - `HI_SVR_PLAYER_EVENT_SOFTWARE`: Event indicating that the start or end of the file is reached. The HiPlayer calls this function when the start or end of the file is reached and tells the application whether to start playback or rewind to the start of the file by using the event parameters.
  - `HI_SVR_PLAYER_EVENT_EOF`: Event indicating that end of the file is reached. When the application receives this event, the playback of the current file is complete, and frame data stored in the hardware buffer is output. The application can play the next file.
  - `HI_SVR_PLAYER_EVENT_PROGRESS`: Playback progress event. During playback, the HiPlayer notifies the application of the percentage of the played part of a file by using this event, and the application implements the progress bar by using this event. The HiPlayer also provides the API `HI_SVR_PLAYER_GetPlayerInfo` for the application to obtain the playback status information.
  - `HI_SVR_PLAYER_EVENT_DOWNLOAD_PROGRESS`: During network playback, the HiPlayer notifies the application of the downloading progress of the current media file by using this event, and the application displays the downloading progress by using this event.
  - `HI_SVR_PLAYER_EVENT_DOWNLOAD_FINISH`: During network playback, the HiPlayer notifies the application of downloading completion of the current media file by using this event.
  - `HI_SVR_PLAYER_EVENT_STREAMID_CHANGED`: Stream change event. For the files with multiple audio or subtitle streams, the application can dynamically switch audio or subtitle streams during playback. The API provided by the HiPlayer is asynchronous. After the streams are successfully switched, the HiPlayer notifies the application of the new stream ID by using this event.



- HI\_SVR\_PLAYER\_EVENT\_SEEK\_FINISHED: Seek completion event. The seek operation is asynchronous during playback. You cannot check whether the seek operation is complete after the seek API is called. The HiPlayer notifies the application of seek completion by using this event.
- HI\_SVR\_PLAYER\_EVENT\_BUFFER\_STATE: Network playback buffer status event. The HiPlayer reports the buffer starting event and notifies the application of the event when no data can be played during network playback due to the network status. Then the application can stop the current playback and wait for the player to buffer data. When the buffered data is sufficient, the HiPlayer reports the buffer sufficiency event. The application resumes playback when it receives this event. The event improves the playback effect when the network status is abnormal. See the implementation in **sample/localplay/sample\_localplay.c**.

The HiPlayer also provides the auto-pause mode when the buffer underload occurs. When the auto-pause mode is enabled, the application does not need to perform the pause or resume operation when receiving a buffer event. See the example of "[Setting the HiPlayer Operation When the Network Buffer Underloads](#)". You can choose either of the preceding ways.

- HI\_SVR\_PLAYER\_EVENT\_FIRST\_FRAME\_TIME: Event indicating the time interval from opening the file to displaying the first frame, that is, time interval from calling SetMedia to displaying the first frame. This callback event is not supported currently.
- HI\_SVR\_PLAYER\_EVENT\_ERROR: Player error event, such as the video channel cannot start and the video fails to be played. Currently, the supported error type is HI\_SVR\_PLAYER\_ERROR\_E.
- HI\_SVR\_PLAYER\_EVNET\_NETWORK\_INFO: Network exception event, indicating that the network connection or playback process is abnormal. The HiPlayer reports the network error type to the application by using this event. The parameter type is HI\_FORMAT\_NET\_STATUS\_S.
- Specify the address for the local or network media file to be played.  
The application calls the API to specify the media file to be played and the subtitle file. In addition, the HiPlayer searches for the subtitle file with the same name as the media file under the directory in which the media file is located (the search operation is implemented in SetMedia). The application can obtain information about the media file by calling HI\_SVR\_PLAYER\_GetFileInfo and display the corresponding program information on the screen. After the media file to be played is successfully specified, obtain the SO handle by calling HI\_SVR\_PLAYER\_GetParam, and register the functions for outputting and clearing subtitles with the SO module. If the played media file has subtitles, the SO module synchronizes the subtitles and then outputs subtitles by calling the corresponding function.
- (Optional) Configure the buffer for network playback. This operation can be dynamically performed during playback.

When the network bandwidth is insufficient, intermittence constantly occurs because the downloaded audio/video data is insufficient. To solve this problem, the HiPlayer defines four thresholds for the buffer based on the amount of audio/video data in the internal buffer. When the data amount in the buffer reaches a threshold, the HiPlayer notifies the application by using the callback event (HI\_SVR\_PLAYER\_EVENT\_BUFFER\_STATE). For details, see the procedure for registering the event callback function in [Figure 25-3](#). The four thresholds (in ascending order) are defined by the following enumerations (the enumeration type is HI\_SVR\_PLAYER\_BUFFER\_E):

- HI\_SVR\_PLAYER\_BUFFER\_EMPTY: There is no data in the buffer, and the threshold cannot be configured. This issue occurs when the playback starts or after the application calls HI\_SVR\_PLAYER\_Seek. At this time, the HiPlayer notifies the



application by using the callback event **HI\_SVR\_PLAYER\_EVENT\_BUFFER\_STATE** (the event parameter is **HI\_SVR\_PLAYER\_BUFFER\_EMPTY**), and the application pauses the playback and waits for the HiPlayer to download data after it receives the event.

- **HI\_SVR\_PLAYER\_BUFFER\_START**: Data amount in the buffer is small. The corresponding threshold can be configured. It must be greater than 0 but less than the threshold corresponding to **HI\_SVR\_PLAYER\_BUFFER\_ENOUGH**. When the data amount in the buffer is less than this threshold, the HiPlayer notifies the application by using the callback event **HI\_SVR\_PLAYER\_EVENT\_BUFFER\_STATE** (the event parameter is **HI\_SVR\_PLAYER\_BUFFER\_START**), and the application can pause the playback and wait for data after it receives the event.
- **HI\_SVR\_PLAYER\_BUFFER\_ENOUGH**: Data in the buffer is sufficient. The corresponding threshold can be configured but must be less than the threshold corresponding to **HI\_SVR\_PLAYER\_BUFFER\_FULL**. When the data amount in the buffer is greater than this threshold but less than the threshold corresponding to **HI\_SVR\_PLAYER\_BUFFER\_FULL**, the HiPlayer notifies the application by using the callback event **HI\_SVR\_PLAYER\_EVENT\_BUFFER\_STATE** (the event parameter is **HI\_SVR\_PLAYER\_BUFFER\_ENOUGH**). If the playback is paused, the application can resume the playback.
- **HI\_SVR\_PLAYER\_BUFFER\_FULL**: The buffer is full, and the HiPlayer stops downloading data. The corresponding threshold is the same as the maximum buffer size. When the data amount in the buffer reaches the maximum, the HiPlayer notifies the application by using the callback event **HI\_SVR\_PLAYER\_EVENT\_BUFFER\_STATE** (the event parameter is **HI\_SVR\_PLAYER\_BUFFER\_FULL**), and the application can continue the playback.

The buffer can be configured by byte (see "[Setting the Network Buffer Size and Thresholds by Byte](#)") or playback duration (in ms, see "[Setting the Buffer Size and Thresholds Based on the Playback Duration](#)"). The HiPlayer configures the buffer by byte by default. You are advised to configure the buffer by playback duration.

- The playback effect significantly varies for the media files with different bit rates when the buffer is configured by byte. For example, if the threshold corresponding to **HI\_SVR\_PLAYER\_BUFFER\_ENOUGH** is set to 4 MB, the buffered data can be played for only a few seconds before the **HI\_SVR\_PLAYER\_BUFFER\_ENOUGH** event is reported for a file with a high bit rate. However, the buffered data can be played for tens of minutes before the **HI\_SVR\_PLAYER\_BUFFER\_ENOUGH** event is reported for a file with a low bit rate.
  - When the buffer is configured by the playback duration of data in the buffer, the required buffer for a media file with a high bit rate may be large and the configured duration cannot be reached. Therefore, the HiPlayer provides an API for limiting the maximum size of the buffer that can be allocated in this mode. For details, see "[Setting the Buffer Size and Thresholds Based on the Playback Duration](#)".
- Control the playback.
    - Destroy the player
    - Fast-forward
    - Rewind
    - Pause
    - Stop
- Operations that can be performed on the player vary according to the playback status. [Figure 25-3](#) shows the details. The value **1** indicates supported, and the value **0** indicates unsupported. For example, in the playing status, the following operations are supported:
- Destroy the player
  - Fast-forward
  - Rewind
  - Pause
  - Stop



- Seek

**Table 25-2** HiPlayer playback control

Operation	Playing	Fast-forwarding	Rewinding	Paused	Stopped
Destroy the player	1	1	1	1	1
Set media	0	0	0	0	1
Play	0	0	0	0	1
Fast-forward	1	1	1	1	0
Rewind	1	1	1	1	0
Pause	1	0	0	0	0
Resume	0	1	1	1	0
Stop	1	1	1	1	0
Seek	1	0	0	1	1

- Exit the playback.

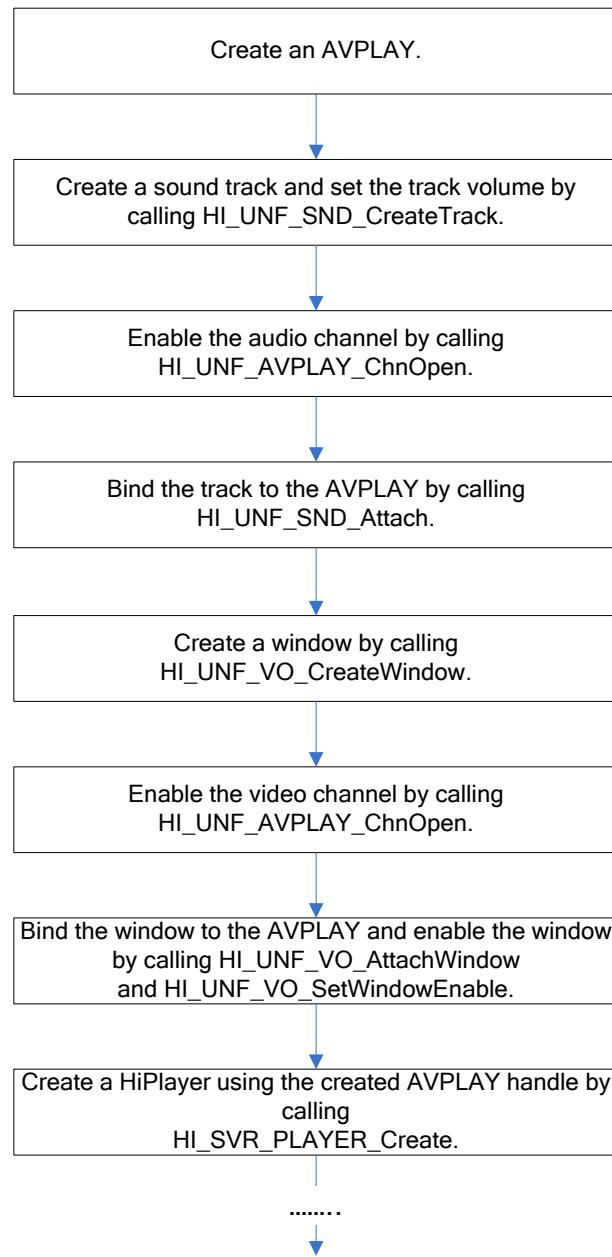
Stop the playback, destroy the player by calling `HI_SVR_PLAYER_Destroy`, unload the parser by calling `HI_SVR_PLAYER_UnRegisterDynamic`, and deinitialize the HiPlayer by calling `HI_SVR_PLAYER_Deinit`.

## Notes

- The playback functions rely on the AVPLAY. You can specify an AVPLAY handle externally or create an AVPLAY in the HiPlayer. [Figure 25-4](#) shows the process for creating a HiPlayer by using an external AVPLAY. After the HiPlayer is destroyed, the audio/video channel of the AVPLAY is in stop state. If the external AVPLAY is not used any more, the application needs to perform the follow-up operations (unbinding and destroying the track and window, disabling the audio/video channel, and destroying the AVPLAY). For details about the reference code of creating an external AVPLAY, see [sample/localplay/sample\\_localplay.c](#).
  - You can set the master/slave attribute when creating a track. The master track differs from other tracks because it supports transparent transmission. Each SOUND device has only one master track.
  - When the track mixing weight is configured by calling `HI_UNF_SND_SetTrackWeight`, only the volume of the current track is affected. When the volume is set by calling `HI_UNF_SND_SetVolume`, the volume of all tracks on the SOUND device is affected.
  - After a window is successfully created, you can set the window display size and position by calling `HI_UNF_VO_SetWindowAttr`.



**Figure 25-4** Creating a HiPlayer using an external AVPLAY



- You need to re-set the subtitle output function when re-setting the media file to be played by calling `HI_SVR_PLAYER_SetMedia`. Setting the media file to be played is a block operation, which cannot be performed in the main task of the application. Otherwise, the main task of the application may fail to respond to user operations in time.

## Browsing Files in the List

On the local file playback screen, all the media files in the disk are presented in a list, as shown in [Figure 25-5](#).



Figure 25-5 HiPlayer local file list

Path of media files C:/  Pirates of Caribbean.avi  Dances with Wolves.avi  Home of the Brave.avi  ...	File: Home of the Brave.avi Duration: 02:30:00 Title: ***** Video encoding: H264 Audio encoding: AAC ...
Media file poster	

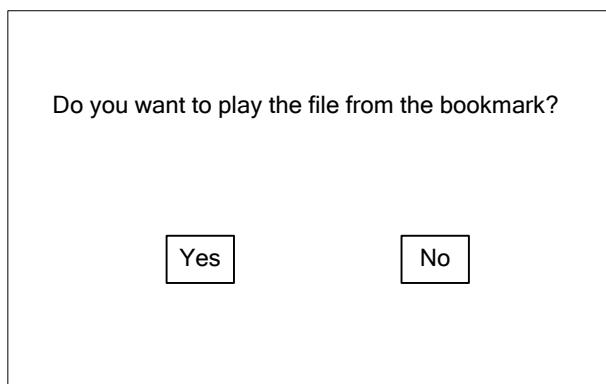
The user can view information about corresponding files by selecting the file. For example, when the user moves the cursor over **Home of the Brave.avi**, the HiPlayer SetMedia interface is called to set the media file to be played, and then the GetFileInfo interface is called to obtain and display the file information.

The SetMedia operation cannot be performed in the user operation main task, because the SetMedia operation is a block operation. SetMedia calls the file parser to parse files, and the parsing time is determined by the file complexity. Typically, it takes 2 seconds. If SetMedia is processed in the main task, the main task cannot respond to user operations in time, which affects user experience. For example, the user presses a key on the remote control to move the cursor, but the cursor does not move. In this case, the application can perform the SetMedia operation by starting a timer. The main task only switches the cursor and starts the timer.

## Implementing the Bookmark Function

If a user exits the film during playback, the application records the current playback position of the film. When the user watches the same film next time, the application asks the user whether to watch the film from the position they exited last time. This function is called bookmark. See Figure 25-6.

Figure 25-6 Asking the user whether to play from the bookmark





The bookmark function is implemented by performing the seek operation after the SetMedia interface is called and before the film is played. The reference code is as follows:

```
.....  
  
s32Ret = HI_SVR_PLAYER_SetMedia(hPlayer, HI_SVR_PLAYER_MEDIA_STREAMFILE,  
&stMedia);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("ERR: HI_SVR_PLAYER_SetMedia err, ret = 0x%x! \n", s32Ret);  
    return;  
}  
  
/* The subtitle callback function needs to be re-set after the SetMedia  
interface is called. */  
s32Ret |= HI_SVR_PLAYER_SetParam(hPlayer, HI_SVR_PLAYER_ATTR_SO_HDL,  
&hSo);  
s32Ret |= HI_SVR_PLAYER_SetParam(hPlayer, HI_SVR_PLAYER_ATTR_AVPLAYER_HDL,  
&hAVPlay);  
s32Ret |= HI_UNF_SO_RegOnDrawCb(hSo, subOndraw, subOnClear, hAVPlay);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("ERR: set subtitle draw function fail! \n");  
    return;  
}  
  
/* The user exited the film after half an hour. The recorded bookmark is  
1800000 ms, that is, 30 minutes (1800000/1000/60). The played time can be  
obtained by calling HI_SVR_PLAYER_GetPlayerInfo during playback.  
s32Ret = HI_SVR_PLAYER_Seek(hPlayer, 1800000);  
s32Ret = HI_SVR_PLAYER_Play(hPlayer);  
  
.....
```

## Supporting FFmpeg Audio/Video Software Decoding

Files in some audio encoding formats must be decoded by using the FFmpeg software. If the HiPlayer version provides the audio software decoding adaptation library **libHA.AUDIO.FFMPEG\_ADEC.decode.so**, it supports FFmpeg audio software decoding. Common audio encoding formats that require software decoding include the following: ADPCM, FLAC, Monkey's Audio (APE), Vorbis, and WMAPro.

Some video encoding formats also require FFmpeg software decoding. If the HiPlayer version provides the video software decoding adaptation library **libHV.VIDEO.FFMPEG\_VDEC.decode.so**, it supports FFmpeg video software decoding. Common video encoding formats that require software decoding include the following: WMV1, WMV2, RV10, RV20, Cinepak, Indeo 4, H.263, Sorenson 1, and Sorenson 3. The software decoding performance is typically restricted by the processor. Therefore, if the



resolution or bit rate of the video is high (also related to the video encoding type), the decoding frame rate is insufficient and the playback is not smooth.

To support the FFmpeg audio/video software decoding, you need to register the audio/video software decoding adaptation library provided by the HiPlayer during system initialization.

```
.....
HI_SYS_Init();

.....
/*Register the audio software decoding library.*/
s32Ret =
    HI_UNF_AVPLAY_RegisterAcodecLib("libHA.AUDIO.FFMPEG_ADEC.decode.so");
if (HI_SUCCESS != s32Ret)
{
    printf("register ffmpeg adep lib failed\n");
}

/*Register the video software decoding library.*/
s32Ret =
    HI_UNF_AVPLAY_RegisterVcodecLib("libHV.VIDEO.FFMPEG_VDEC.decode.so");
if (HI_SUCCESS != s32Ret)
{
    printf("register ffmpeg vdec lib failed\n");
}

.....
s32Ret = HI_SVR_PLAYER_Init();
.....
```

## Obtaining MetaData

After the media information is successfully set by calling HI\_SVR\_PLAYER\_SetMedia, you can obtain audio/video information about the current media file by calling HI\_SVR\_PLAYER\_GetFileInfo. For some files, MetaData contains additional information, such as the title, artist, album, and style. The application can obtain the MetaData by calling HI\_SVR\_PLAYER\_Invoke.

```
HI_HANDLE hPlayer = (HI_HANDLE)NULL,
HI_SVR_PLAYER_METADATA_S stMetaData;
...
/*Create a HiPlayer.*/
s32Ret = HI_SVR_PLAYER_Create(..., &hPlayer);
if (HI_SUCCESS != s32Ret)
{
    return HI_FAILURE;
}
...
s32Ret = HI_SVR_PLAYER_SetMedia(hPlayer, ...);
if (HI_SUCCESS != s32Ret)
{
```



```
    return HI_FAILURE;
}
...
memset(&stMetaData, 0, sizeof(stMetaData));
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer, HI_FORMAT_INVOKE_GET_METADATA,
&stMetaData);
if (HI_SUCCESS == s32Ret)
{
    /*Display MetaData information.*/
    HI_SVR_META_PRINT(&stMetaData);
}

...

```

MetaData is stored by key value pairs. For details about how to obtain the data, see the implementation of HI\_SVR\_META\_PRINT.

## Obtaining Video File Thumbnails

After opening a directory for storing video files on a PC, you can see a thumbnail for each file, which presents a video frame at a time point of the file. The HiPlayer also supports thumbnails. After the media file is configured by calling HI\_SVR\_PLAYER\_SetMedia, you can obtain a video frame as the thumbnail by calling HI\_SVR\_PLAYER\_Invoke. The pixel format and size of the obtained image can be specified.

### NOTE

The HiPlayer basic version does not support the function of obtaining video file thumbnails.

```
HI_HANDLE hPlayer = (HI_HANDLE) NULL,
HI_FORMAT_THUMBNAIL_PARAM_S stThumbnailParam;
HI_S32 i;

...
/*Create a HiPlayer.*/
s32Ret = HI_SVR_PLAYER_Create(..., &hPlayer);
if (HI_SUCCESS != s32Ret)
{
    return HI_FAILURE;
}
...
s32Ret = HI_SVR_PLAYER_SetMedia(hPlayer, ...);
if (HI_SUCCESS != s32Ret)
{
    return HI_FAILURE;
}
...
memset(&stThumbnailParam, 0, sizeof(stThumbnailParam));
stThumbnailParam.thumbPixelFormat = HI_FORMAT_PIX_FMT_YUV420P; /*Specify
```



```
the pixel format of the image to be obtained.*/
stThumbnailParam.thumbnailSize = 120; /*Specify the size of the image to
be obtained.*/
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer, HI_FORMAT_INVOKE_GET_THUMBNAIL,
(HI_VOID *)&stThumbnailParam);
printf("get thumbnail ret %d\n", s32Ret);
if (HI_SUCCESS == s32Ret)
{
    printf("thumbnail width:%d,height:%d\n", stThumbnailParam.width,
stThumbnailParam.height);
    for (i = 0; i < HI_FORMAT_THUMBNAIL_PLANAR; i++)
    {
        if (stThumbnailParam.frameData[i] != NULL)
        {
            /*Release the memory.*/
            free(stThumbnailParam.frameData[i]);
            stThumbnailParam.frameData[i] = NULL;
        }
    }
}
...
...
```

The specified size (**stThumbnailParam.thumbnailSize**) of the thumbnail here indicates the larger one between the width and height of the thumbnail. If the width of the original video is greater than the height, the specified size is the obtained thumbnail width; if the width of the original video is less than the height, the specified size is the obtained thumbnail height. Note that the application needs to release the thumbnail frame data (**stThumbnailParam.frameData**) after using it. For details, see the related HiPlayer header files.

## Setting the Network Buffer Size and Thresholds by Byte

```
HI_HANDLE hPlayer = (HI_HANDLE)NULL,
HI_FORMAT_BUFFER_CONFIG_S stBufConfig;
HI_S64 s64BufMaxSize;
...
/*Create a HiPlayer.*/
s32Ret = HI_SVR_PLAYER_Create(..., &hPlayer);
if (HI_SUCCESS != s32Ret)
{
    return HI_FAILURE;
}
...
s32Ret = HI_SVR_PLAYER_SetMedia(hPlayer, ...);
if (HI_SUCCESS != s32Ret)
{
```



```
        return HI_FAILURE;
    }

/*Specify the maximum value of the network buffer size.*/
s64BufMaxSize = 5*1024*1024; /*Bytes*/
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer,
HI_FORMAT_INVOKE_SET_BUFFER_MAX_SIZE, &s64BufMaxSize);
if (HI_SUCCESS != s32Ret)
{
    HI_SVR_PLAYER_Invoke(hPlayer, HI_FORMAT_INVOKE_GET_BUFFER_MAX_SIZE,
&s64BufMaxSize);
}

memset(&stBufConfig, 0, sizeof(HI_FORMAT_BUFFER_CONFIG_S));
stBufConfig.eType = HI_FORMAT_BUFFER_CONFIG_SIZE;
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer, HI_FORMAT_INVOKE_GET_BUFFER_CONFIG,
&stBufConfig);
if (HI_SUCCESS == s32Ret)
{
    stBufConfig.eType = HI_FORMAT_BUFFER_CONFIG_SIZE;
    /*The value of stBufConfig.s64Total cannot be greater than that of
s64BufMaxSize.*/
    stBufConfig.s64Total = s64BufMaxSize; /*Bytes*/
    stBufConfig.s64EventStart = (256 * 1024); /*Bytes*/
    stBufConfig.s64EventEnough = (3 * 1024 * 1024); /*Bytes*/
    stBufConfig.s64TimeOut = 120000; /*ms*/
    s32Ret = HI_SVR_PLAYER_Invoke(hPlayer,
HI_FORMAT_INVOKE_SET_BUFFER_CONFIG, &stBufConfig);
}

if (HI_SUCCESS != s32Ret)
{
    printf("Config buffer failed\n");
}

...

```

After the buffer is configured based on the preceding codes, the buffer size is 5 MB, and the `HI_SVR_PLAYER_EVENT_BUFFER_STATE` event is reported when data in the buffer is 0, less than 256 KB, greater than or equal to 3 MB, or greater than or equal to 5 MB. The event carries different parameters to identify different buffer states. `s64EventStart` indicates the threshold corresponding to `HI_SVR_PLAYER_BUFFER_START`, and `s64EventEnough` indicates the threshold corresponding to `HI_SVR_PLAYER_BUFFER_ENOUGH`. For details about how to parse the `HI_SVR_PLAYER_EVENT_BUFFER_STATE` event parameters, see the implementation of the HiPlayer event callback function (registered by calling `HI_SVR_PLAYER_RegCallback`) in `sample/localplay/sample_localplay.c`.



## Setting the Buffer Size and Thresholds Based on the Playback Duration

```
HI_HANDLE hPlayer = (HI_HANDLE)NULL,
HI_FORMAT_BUFFER_CONFIG_S stBufConfig;
HI_S64 s64BufMaxSize = 0;

...
/*Create a HiPlayer.*/
s32Ret = HI_SVR_PLAYER_Create(..., &hPlayer);
if (HI_SUCCESS != s32Ret)
{
    return HI_FAILURE;
}
...

s32Ret = HI_SVR_PLAYER_SetMedia(hPlayer, ...);
if (HI_SUCCESS != s32Ret)
{
    return HI_FAILURE;
}

s32Ret = HI_SVR_PLAYER_Invoke(hPlayer,
HI_FORMAT_INVOKE_GET_BUFFER_MAX_SIZE, &s64BufMaxSize);
if (HI_SUCCESS == s32Ret)
{
    printf("current buffer max size is :%lld bytes\n", s64BufMaxSize);
}

/*Set the maximum size of the network buffer to 10 MB.*/
s64BufMaxSize = (10 * 1024 * 1024);
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer,
HI_FORMAT_INVOKE_SET_BUFFER_MAX_SIZE, &s64BufMaxSize);
if (HI_SUCCESS != s32Ret)
{
    printf("set buffer max size to %lld bytes failed\n", s64BufMaxSize);
}

memset(&stBufConfig, 0, sizeof(HI_FORMAT_BUFFER_CONFIG_S));
stBufConfig.eType = HI_FORMAT_BUFFER_CONFIG_TIME;
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer, HI_FORMAT_INVOKE_GET_BUFFER_CONFIG,
&stBufConfig);
if (HI_SUCCESS == s32Ret)
{
    stBufConfig.eType = HI_FORMAT_BUFFER_CONFIG_TIME;
    stBufConfig.s64Total = 30000; /*ms*/
    stBufConfig.s64EventStart = 500; /*ms*/
    stBufConfig.s64EventEnough = 10000; /*ms*/
```



```
    stBufConfig.s64TimeOut = 120000; /*ms*/
    s32Ret = HI_SVR_PLAYER_Invoke(hPlayer,
        HI_FORMAT_INVOKE_SET_BUFFER_CONFIG, &stBufConfig);
}

if (HI_SUCCESS != s32Ret)
{
    printf("Config buffer failed\n");
}

...
```

After the buffer is configured based on the preceding codes, the buffer size is the size of data that can be played for 30 seconds. However, if the bit rate of the media file is high, the data size may exceed the maximum buffer size (10 MB) configured by `HI_FORMAT_INVOKE_SET_BUFFER_MAX_SIZE`. In this case, the parser limits the data amount in the buffer to less than 10 MB, and therefore the actual playback duration may be less than 30 seconds. The `HI_SVR_PLAYER_EVENT_BUFFER_STATE` event is reported when there is no data in the buffer, or when the playback duration of data in the buffer is less than 500 ms, greater than or equal to 10 seconds, or greater than or equal to 30 seconds (or when 10 MB is reached). The event carries different event parameters to identify different buffer states. `s64EventStart` indicates the threshold corresponding to `HI_SVR_PLAYER_BUFFER_START`, and `s64EventEnough` indicates the threshold corresponding to `HI_SVR_PLAYER_BUFFER_ENOUGH`. For details about how to parse the `HI_SVR_PLAYER_EVENT_BUFFER_STATE` event parameters, see the implementation of the HiPlayer event callback function (registered by calling `HI_SVR_PLAYER_RegCallback`) in `sample/localplay/sample_localplay.c`.

## Setting the HiPlayer Operation When the Network Buffer Underloads

```
HI_HANDLE hPlayer = (HI_HANDLE)NULL,
HI_BOOL bUnderRunPause = HI_TRUE;
...
/*Create a HiPlayer.*/
s32Ret = HI_SVR_PLAYER_Create(..., &hPlayer);
if (HI_SUCCESS != s32Ret)
{
    return HI_FAILURE;
}
...

s32Ret = HI_SVR_PLAYER_Invoke(hPlayer,
    HI_FORMAT_INVOKE_SET_BUFFER_UNDERRUN, &bUnderRunPause);
if (HI_SUCCESS != s32Ret)
{
    printf("set buffer underrun pause failed\n");
}
```



...

This configuration can be implemented after the HiPlayer is created or during the playback process. After the configuration, when there is no data in the network buffer (HI\_SVR\_PLAYER\_BUFFER\_EMPTY) or when data in the network buffer reaches the threshold corresponding to HI\_SVR\_PLAYER\_BUFFER\_START, the HiPlayer automatically pauses the playback. When data in the network buffer reaches the threshold corresponding to HI\_SVR\_PLAYER\_BUFFER\_ENOUGH or the threshold corresponding to HI\_SVR\_PLAYER\_BUFFER\_FULL, the HiPlayer plays the file continuously. In this case, the network buffer events are also reported.

## Obtaining Network Bandwidth During Network Playback

During network playback, you can obtain the current network bandwidth before data downloading is complete by using the following method. You can check whether data downloading is complete by using the downloading progress event and downloading completion event reported by the HiPlayer. For details about how to register the event callback function, see [Figure 25-3](#).

```
HI_HANDLE hPlayer = (HI_HANDLE)NULL,  
HI_S64 s64BandWidth = 0;  
...  
/*Create a HiPlayer.*/  
s32Ret = HI_SVR_PLAYER_Create(..., &hPlayer);  
if (HI_SUCCESS != s32Ret)  
{  
    return HI_FAILURE;  
}  
...  
  
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer, HI_FORMAT_INVOKE_GET_BANDWIDTH,  
&s64BandWidth);  
if (HI_SUCCESS == s32Ret)  
{  
    printf("bandwidth is %lld bps\n", s64BandWidth);  
}  
...
```

### 25.4.2 Outputting Subtitles

#### Scenario

Subtitles can be classified into text subtitles and graphical subtitles. Common text subtitles include LRC, ASS, SRT, and SMI, and common graphical subtitles include PGS, and IDX+SUB. Displaying text subtitles on the screen requires font libraries, and possibly transcoding. The HiPlayer calls the file parser to parse subtitle files, and transmits the parsed subtitles to the SO module. The SO module synchronizes the subtitle data and outputs the subtitles by calling the subtitle output function registered by the application (by calling HI\_UNF\_SO\_RegOnDrawCb). If the subtitles are text subtitles and need to be transcoded,



the file parser calls the transcoding module. The library **libhi\_charsetxx** and header file **hi\_svr\_charsetxx.h** provided by the HiPlayer belong to the transcoding module.

 **NOTE**

The HiPlayer basic version does not support the CHAESSET transcoding module.

## Working Process

To output subtitles, the application needs to perform the following operations:

- Initializes the transcoding module CHARSET before calling HI\_SVR\_PLAYER\_Create.
  - Initializes the CHARSET module by calling HI\_CHARSET\_Init.
  - Registers the transcoding library **libhi\_charset.so** by calling HI\_CHARSET\_RegisterDynamic.
- Sets the CHARSET management API for the HiPlayer after the HiPlayer is created by calling HI\_SVR\_PLAYER\_Create and before the URL is set by calling HI\_SVR\_PLAYER\_SetMedia. The HiPlayer then sets the API for the corresponding file parser.  
Sets the CHARSET management function by calling HI\_SVR\_PLAYER\_Invoke(hPlayer, HI\_FORMAT\_INVOKE\_SET\_CHARSETMGR\_FUNC, &g\_stCharsetMgr\_s).
- Sets the target subtitle transcoding type after the HiPlayer is created by calling HI\_SVR\_PLAYER\_Create. Currently only UTF-8 is supported. The target transcoding type can be dynamically modified during playback.  
Sets the target transcoding type to UTF-8 by calling HI\_SVR\_PLAYER\_Invoke(hPlayer, HI\_FORMAT\_INVOKE\_SET\_DEST\_CODETYPE, &s32DestCodeType).
- Registers the callback function for outputting subtitles.
  - Obtains the SO handle by calling HI\_SVR\_PLAYER\_GetParam.
  - Obtains the AVPLAY handle by calling HI\_SVR\_PLAYER\_GetParam.
  - Registers the function for outputting and clearing subtitles by calling HI\_UNF\_SO\_RegOnDrawCb.
- Outputs subtitles. The subtitle output and clear functions need to be implemented by applications. For details about the implementation, see subOndraw and subOnclear in **sample\_localplay.c**.
  - The subOndraw function implements the display of subtitles. After the SO module completes subtitle synchronization, it calls this function to output subtitles. **sample\_localplay.c** in **sample\localplay** is a sample that implements subtitle display by using the HiGo interface. The user data can be attached when the subtitle output function is registered. When the SO module calls the subtitle output or clearing functions, it transparently transmits the user data to the user.
  - The subOnclear function clears subtitles. For example, for srt subtitles, if the start and end time for displaying the subtitles is specified, the SO module clears the displayed subtitles by calling this function when the end time is reached.

For details about CHARSET-related operations, see the code isolated by using the macro **CHARSET\_SUPPORT** in **sample/localplay/sample\_localplay.c**.

 **NOTE**

Ignore the related procedure if you do not need the CHARSET transcoding function.



## Notes

- The CHARSET module automatically identifies the source encoding type of subtitles by default. If the displayed text subtitles are garbled, the source encoding type may be incorrectly identified. In this case, the application can forcibly set the source encoding type by calling HI\_SVR\_PLAYER\_Invoke(hPlayer, HI\_FORMAT\_INVOKE\_SET\_SOURCE\_CODETYPE, &s32SrcCodeType).
- You must call the SetMedia interface before registering the subtitle output function by calling HI\_UNF\_SO\_RegOnDrawCb, and you must register the subtitle output function again after re-calling the SetMedia interface.

## Registering the Subtitle Output Function

```
...
HI_HANDLE hSo = (HI_HANDLE)NULL, hAVPlay = (HI_HANDLE)NULL;

/* Obtain the SO handle by calling the HiPlayer API.*/
s32Ret = HI_SVR_PLAYER_GetParam(hPlayer, HI_SVR_PLAYER_ATTR_SO_HDL, &hSo);
s32Ret |= HI_SVR_PLAYER_GetParam(hPlayer, HI_SVR_PLAYER_ATTR_AVPLAYER_HDL,
&hAVPlay);

/* Register the subtitle output and clear function by calling the UNF API.
*/
s32Ret |= HI_UNF_SO_RegOnDrawCb(hSo, subOndraw, subOnClear, hAVPlay);
...
```

## Rolling Lyrics of MP3 Files

When an MP3 file is being played, the lyrics can be rolled on the screen, as shown in [Figure 25-7](#).

### NOTE

The HiPlayer basic version does not support the lrc subtitle format.



**Figure 25-7** Rolling lyrics

Song: Burning  
Singer: Maria Arre  
Album: Burning CDs

passion is sweet  
love makes weak  
you said you cherished freedom so  
you refuse to let it go  
follow your fate  
love and hate  
never fail to seize the day  
but don't give yourself away  
oh when the night falls  
and your all alone  
in your deepest sleep  
what are you dreaming of  
my skin's still burning from your touch  
oh i just can't get enough i  
said i wouldn't ask for much  
but your eyes are dangerous  
oh the thought keep spinning in my head  
can we drop this masquerade  
i can't predict where it ends  
if your the rock i'll crush against  
trapped in a crowd  
the music is loud  
i said i love my freedom to  
now i'm not sure i do  
all eyes on you  
rings so true  
better quit while you're ahead  
now i'm not so sure i am  
my soul my heart  
if you're near if you're far  
my life my love  
you can have it all....ooohaaaah.  
ooh if your the rock i'll crush against.

Currently, the SO outputs subtitles only sentence by sentence. The application can separately parse the lrc subtitles. During the MP3 file playback, the SO module notifies the application of the MP3 lyric information to be displayed by using the callback function. The lyric information contains the PTS. The application compares the obtained PTS with the PTS of parsed lyrics in the callback function and highlights the lyrics if the PTS is the same as the obtained PTS in rolling mode.

```
static HI_S32 subOnDraw(HI_U32 u32UserData, const
HI_UNF_SO_SUBTITLE_INFO_S *pstInfo, HI_VOID *pArg)
{
    HI_RECT rc = {160, 550, 460, 30};

    switch (pstInfo->eType)
    {
        case HI_UNF_SUBTITLE_TEXT:
            if (NULL == pstInfo->unSubtitleParam.stText.pu8Data)
            {
                return HI_FAILURE;
            }
    }
}
```



```
        printf("OUTPUT: pts %lld, %s \n", pstInfo->unSubtitleParam.stText.s64Pts,
               pstInfo->unSubtitleParam.stText.pu8Data);

        /* Search for the parsed lrc subtitles based on lyric PTS to roll
         * the lyrics. */
        break;
    default:
        break;
    }

    return HI_SUCCESS;
}
```

### 25.4.3 Setting Attributes

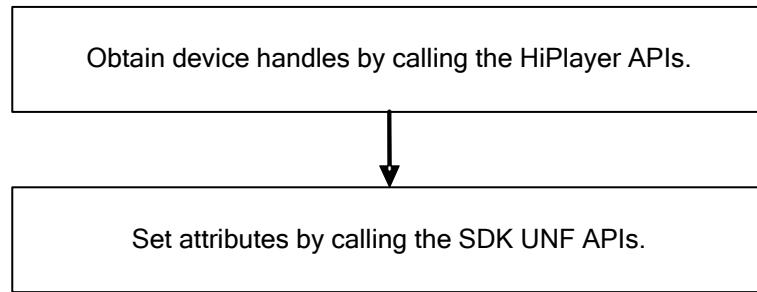
#### Scenario

The HiPlayer does not encapsulate the settings of attributes including the playback volume, video display region, and mute mode. It only provides APIs for obtaining the AVPLAY, SO, and window device handles.

#### Working Process

Figure 25-8 shows the process for setting the attributes.

**Figure 25-8** Process for setting the attributes



#### Notes

None

#### Setting the Video Display Region

```
HI_HANDLE hdl= 0;
HI_UNF_WINDOW_ATTR_S stWinAttr;

/* Obtain the window handle by calling the HiPlayer API. */
s32Ret = HI_SVR_PLAYER_GetParam(hPlayer, HI_SVR_PLAYER_ATTR_WINDOW_HDL,
&hdl);
```



```
s32Ret |= HI_UNF_VO_GetWindowAttr(hdl, &stWinAttr);

if (HI_SUCCESS != s32Ret)
{
    printf("ERR: get win attr fail! \n");
    return HI_FAILURE;
}

stWinAttr.stOutputRect.s32X = 100;
stWinAttr.stOutputRect.s32Y = 100;
stWinAttr.stOutputRect.s32Width = 300;
stWinAttr.stOutputRect.s32Height = 300;

/* Set the window display region by calling the UNF API. */
s32Ret = HI_UNF_VO_SetWindowAttr(hdl, &stWinAttr);
...
```

## Fine-Tuning Audio, Video, and Subtitle Synchronization

When the audio, video, and subtitle are not synchronous, the application can fine-tune the synchronization by calling the corresponding UNF interface. To restore to the initial state, set each member of stSyncAttr to 0.

```
...
HI_HANDLE hdl= 0;
HI_UNF_SYNC_ATTR_S stSyncAttr;

/* Obtain the AVPLAY handle by calling the HiPlayer API. */
s32Ret = HI_SVR_PLAYER_GetParam(hPlayer, HI_SVR_PLAYER_ATTR_AVPLAYER_HDL,
&hdl);
s32Ret |= HI_UNF_AVPLAY_GetAttr(hdl, HI_UNF_AVPLAY_ATTR_ID_SYNC,
(HI_VOID*)&stSyncAttr);

if (HI_SUCCESS != s32Ret)
{
    printf("ERR: get sync attr fail! \n");
    return HI_FAILURE;
}

stSyncAttr.s32VidPtsAdjust = 50; /*The video leads the audio. Add 50 ms to
the PTS of each video frame.*/

/* Set the AVPLAY synchronization attribute by calling the UNF API. */
s32Ret = HI_UNF_AVPLAY_SetAttr(hdl, HI_UNF_AVPLAY_ATTR_ID_SYNC,
```



```
(HI_VOID*) &stSyncAttr);  
...  
  
Setting the Playback Volume  
...  
HI_UNF_SND_GAIN_ATTR_S stGain;  
  
s32Ret = HI_SVR_PLAYER_GetParam(hPlayer, HI_SVR_PLAYER_ATTR_AUDTRACK_HDL  
if (HI_SUCCESS != s32Ret)  
    printf("ERR: get sound track hdl fail! \n");  
    return HI_FAILURE;  
}  
/* Set the current playback volume by calling the UNF API.*/  
  
stGain.bLinearMode = HI_TRUE;  
stGain.s32Gain = 100;  
s32Ret = HI_UNF_SND_SetTrackWeight(hdl, &stGain);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("ERR: set volume failed! \n");  
    return HI_FAILURE;  
}  
...  
...
```

## Setting the ID of the Stream to Be Played

A media file may contain multiple audio streams. You can select different streams for output during playback. After the file with multiple audio streams is parsed, the file information can be obtained by calling HI\_SVR\_PLAYER\_GetFileInfo.

File name: Pirates of Caribbean.1.avi

Stream type: ES

File size: 321441792 bytes

Start time: 0:0:0

Duration: 1:8:35

bps: 8973 kb/s

Program 0:

video info:

stream idx: 0

format: H264

w \* h: 1920 \* 1088

fps: 23.976

bps: 0 kb/s

audio 0 info:



```
stream idx:      1
format:         DTS
samplerate:     48000 Hz
bitpersample:   0
channels:       5
bps:            754 kb/s
lang:
audio 1 info:
stream idx:      2
format:         AC3
samplerate:     48000 Hz
bitpersample:   0
channels:       2
bps:            192 kb/s
lang:
audio 2 info:
stream idx:      3
format:         AC3
samplerate:     48000 Hz
bitpersample:   0
channels:       6
bps:            448 kb/s
lang:
```

The preceding contents are the file information obtained after **Pirates of Caribbean.1.avi** is opened. This file contains one video stream and three audio streams. If the ID of the stream to be played is not specified before playback, the player plays stream 0 by default. If you want to output audio stream 1 during playback, do as follows:

```
...
s32Ret = HI_SVR_PLAYER_GetParam(hPlayer, HI_SVR_PLAYER_ATTR_STREAMID,
(HI_VOID*)&stStreamId;

stStreamId.u16AudStreamId = 1;

if (HI_SUCCESS == s32Ret)
{
    s32Ret = HI_SVR_PLAYER_SetParam(hPlayer, HI_SVR_PLAYER_ATTR_STREAMID,
(HI_VOID*)&stStreamId);
}
...
```



## 25.4.4 Implementing the File Parser/Protocol

### Scenario

For details about the concepts of the file parser and protocol, see section [25.2 "Important Concepts."](#)

The file formats supported by the HiPlayer are determined by the file parser registered by the application with the HiPlayer. The file parser is a dynamic library stored in `\usr\lib` of the file system, such as `libformat.so`.

### Working Process

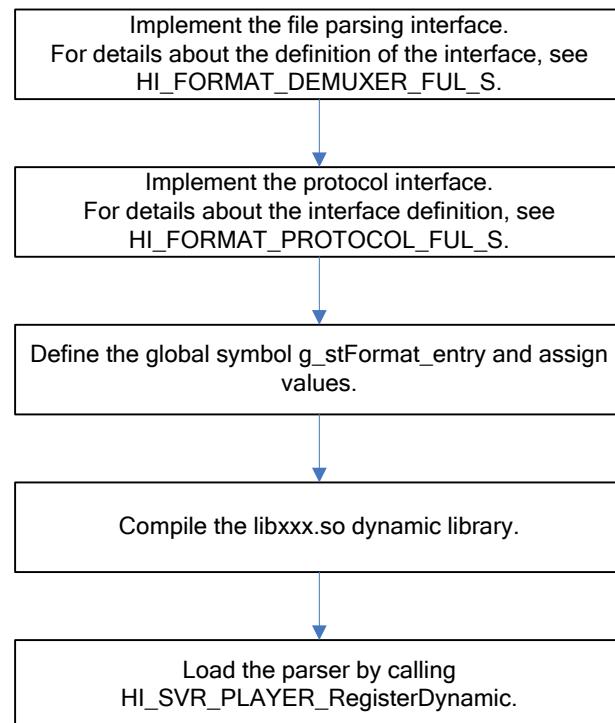
[Figure 25-9](#) shows the process for implementing the file parser. To implement the file parser, you must implement the file parsing interface, but not necessarily the protocol interface (only if the protocol interface, such as `fopen` and `fread`, of one parser is implemented). After the parsing interface is implemented, the `g_stFormat_entry` symbol must be defined and the symbol attributes, such as the supported file formats, protocols, and parser priority, must be specified. The symbol name must be `g_stFormat_entry`.

Similarly, to implement a proprietary protocol, you must implement the protocol interface but not necessarily the file parser interface. For details, see [sample/localplay/test\\_protocol.c](#).

#### NOTE

The HiPlayer provides the File protocol only for external parsers. To implement a proprietary parser for network playback, the HiPlayer must implement the proprietary network protocol.

**Figure 25-9** Implementing the file parser





The following is an example of parser/protocol global symbol definitions. The parser and protocol interfaces in the right of the equal signs are taken as examples to describe the precautions to be taken before using these interfaces.

```
HI_FORMAT_S g_stFormat_entry = {  
    .pszDllName          = (const HI_PCHAR )"libmydemuxer.so",  
    .pszFormatName       = "srt,mp3,ts",  
    .pszProtocolName     = "myprotocol",  
    .stLibVersion        = {1, 0, 0, 0},  
    .pszFmtDesc          = (const HI_PCHAR )"My demuxer",  
    .u32Merit            = 0xFFFFFFFF, /*Priority*/  
    .stDemuxerFun.fmt_find      = MY_DEMUXER_Find,  
    .stDemuxerFun.fmt_open       = MY_DEMUXER_Open,  
    .stDemuxerFun.fmt_find_stream= MY_DEMUXER_FindStream,  
    .stDemuxerFun.fmt_getinfo   = MY_DEMUXER_GetInfo,  
    .stDemuxerFun.fmt_read      = MY_DEMUXER_Read,  
    .stDemuxerFun.fmt_free      = MY_DEMUXER_Free,  
    .stDemuxerFun.fmt_invoke    = MY_DEMUXER_Invoke,  
    .stDemuxerFun.fmt_close     = MY_DEMUXER_Close,  
    .stDemuxerFun.fmt_seek_pts  = MY_DEMUXER_SeekPts,  
    .stDemuxerFun.fmt_seek_pos  = MY_DEMUXER_SeekPos,  
  
    .stProtocolFun.url_find    = MY_URL_Find,  
    .stProtocolFun.url_open     = MY_URL_Open,  
    .stProtocolFun.url_read    = MY_URL_Read,  
    .stProtocolFun.url_seek    = MY_URL_Seek,  
    .stProtocolFun.url_close   = MY_URL_Close,  
    .stProtocolFun.url_read_seek= MY_URL_ReadSeek,  
    .stProtocolFun.url_invoke  = MY_URL_Invoke,  
}
```

- Implement the file parser interfaces. The member **stDemuxerFun** of the global symbol **g\_stFormat\_entry** defines the parser interfaces to be implemented.
  - **MY\_DEMUXER\_Find**: When the application sets the playback URL by calling **HI\_SVR\_PLAYER\_SetMedia**, the HiPlayer calls the find interface of each parser in sequence based on the priority of the registered parsers. The first parameter is the playback URL, and the second parameter is the protocol interface pointer. Because the parser can determine whether it supports the corresponding media file only after it has analyzed some of the media data, the parser returns **OK** if it cannot determine whether it supports the media file at this time.
  - **MY\_DEMUXER\_Open**: After the specific parser is found by calling **MY\_DEMUXER\_Find**, the HiPlayer calls the open interface of the parser. The second parameter of the open interface is the protocol interface pointer (**HI\_FORMAT\_PROTOCOL\_FUN\_S \***). The parser calls the find interface of the protocol to check whether the protocol supports the URL specified in the parameter. If the protocol does not support the URL or the find interface is not implemented, a failure code is returned. (If the parser needs to use other proprietary protocol, the protocol must support the URL.) The parser allocates the parser handle and saves the used protocol interface pointer in the open interface, opens the file or connects to the



network by calling the open interface of the protocol, and reads sufficient data for parsing by calling the read interface of the protocol to check whether itself can parse the media file of the URL. If the parser does not support the media file of the URL, a failure code is returned.

- **MY\_DEMUXER\_FindStream:** After opening the media file by calling the open interface, the HiPlayer calls the FindStream interface of the parser. In this interface, the parser calls the read interface of the protocol to continue to read file data and parse file information, such as the encoding type of the audio/video stream, resolution and frame rate of the video stream, sampling rate of the audio stream, number of channels, and type of the subtitle stream. For details about the file information parsed by the parser, see the definition `HI_FORMAT_FILE_INFO_S`.
- **MY\_DEMUXER\_GetInfo:** This interface is used to obtain file information. After the parser parses file information by calling the FindStream interface, the HiPlayer obtains the file information by calling `MY_DEMUXER_GetInfo` so that the audio/video decoder can be correctly configured when the playback starts. Actually, the file information that the application obtains from the HiPlayer by calling `HI_SVR_PLAYER_GetFileInfo` is the information that the HiPlayer obtains from the parser by calling `MY_DEMUXER_GetInfo`.
- **MY\_DEMUXER\_Read:** This interface is used to read data. During playback, the HiPlayer reads audio/video frames and subtitle data by constantly calling `MY_DEMUXER_Read`, and the parser reads data, separates audio, video, and subtitle data, and implements framing by calling the read interface of the protocol. The parser needs to allocate the frame buffer itself. For details about the frame structure, see the definition of `HI_FORMAT_FRAME_S`. If the read interface of the protocol returns an error code, the parser needs to return the error code `HI_FORMAT_ERRNO_ENDOFFILE`, indicating that the end of file (EOF) is reached.
- **MY\_DEMUXER\_Free:** This interface is used to release the frame buffer. The HiPlayer calls this interface after it obtains frames by using the read interface, and the parser needs to release the frame buffer in this interface.
- **MY\_DEMUXER\_SeekPts:** When the application calls `HI_SVR_PLAYER_Seek` or `HI_SVR_PLAYER_TPlay`, the HiPlayer calls `MY_DEMUXER_SeekPts` to perform the seek operation on the file by PTS (in ms). The parser implements the seek operation by calling the seek interface of the protocol in this interface. The parameter in the seek interface of the protocol indicates the offset value (in byte) of the target seek point of the file. Therefore, the parser needs to convert the PTS into a byte offset value. After this interface is returned, the parser needs to make sure that the first frame read by calling the read interface is the key frame that starts from the new position. In addition, the implementation of the seek operation by the parser varies according to the value of the last parameter **SeekFlag** in this interface (the parameter type is `HI_FORMAT_SEEK_FLAG_E`).
  - `HI_FORMAT_AVSEEK_FLAG_NORMAL`: Seek the first key frame after the PTS. If the PTS just corresponds to a key frame, seek the key frame.
  - `HI_FORMAT_AVSEEK_FLAG_BACKWARD`: Seek the nearest key frame before the PTS. If the PTS just corresponds to a key frame, seek the key frame.
  - `HI_FORMAT_AVSEEK_FLAG_BYTE`: Seek by position (in byte). (In this case, the PTS parameter value in the interface indicates the offset value in byte of the target seek point in the file.) Seek the first key frame after the offset value. If the seek point just corresponds to a key frame, seek the key frame.
  - `HI_FORMAT_AVSEEK_FLAG_ANY`: Seek the key frame nearest to the PTS (before or after the frame corresponding to the PTS).



- MY\_DEMUXER\_SeekPos: If the last parameter **SeekFlag** of the SeekPts interface is **HI\_FORMAT\_AVSEEK\_FLAG\_BYTE**, the function that needs to be implemented by the parser is similar to that of the SeekPts interface.
  - MY\_DEMUXER\_Invoke: This is an interface for extending various functions defined by the second parameter **InvokeID** (the type is **HI\_FORMAT\_INVOKE\_ID\_E**). The **InvokeID** can be set to the following values:
    - HI\_FORMAT\_INVOKE\_USER\_OPE: When the application calls the following playback control interfaces of the HiPlayer:
      - ✧ HI\_SVR\_PLAYER\_Play
      - ✧ HI\_SVR\_PLAYER\_Stop
      - ✧ HI\_SVR\_PLAYER\_Pause
      - ✧ HI\_SVR\_PLAYER\_Resume
      - ✧ HI\_SVR\_PLAYER\_TPlay
      - ✧ HI\_SVR\_PLAYER\_Seek
- The HiPlayer transparently transmits the operation to the parser by using the Invoke interface of the parser. The second parameter **InvokeID** is **HI\_FORMAT\_INVOKE\_USER\_OPE**, and the third parameter can be set to any of the following (corresponding to the preceding playback control interfaces):
- ✧ HI\_FORMAT\_CMD\_STOP: After this operation is performed, calling the read interface of the parser returns a failure code. For network playback, the parser also needs to request the server to stop sending data.
  - ✧ HI\_FORMAT\_CMD\_PLAY: After this operation is performed, calling the read interface of the parser returns data from the first key frame of the media file. For network playback, the parser also needs to request data from the server.
  - ✧ HI\_FORMAT\_CMD\_PAUSE: For network playback, if the parser has its own network buffer and the buffer is full, the parser needs to request the server to pause data transmission.
  - ✧ HI\_FORMAT\_CMD\_BACKWARD/HI\_FORMAT\_CMD\_FORWARD (rewind/fast-forward): The parser does not need to implement the rewind/fast-forward function because the HiPlayer implements this function by calling the SeekPts interface of the parser intermittently. However, the parser needs to ensure the data read speed when this function is implemented.
  - ✧ HI\_FORMAT\_CMD\_RESUME (resume from the pausing/rewinding/fast-forwarding state): For network playback, if the parser requests the server to pause data transmission before, it needs to request the server to continue to transmit data again.
  - HI\_FORMAT\_INVOKE\_GET\_METADATA: This interface is used to obtain media file metadata. When the application calls the HiPlayer interface **HI\_SVR\_PLAYER\_Invoke** (**InvokeID** is **HI\_FORMAT\_INVOKE\_GET\_METADATA**), the HiPlayer transparently transmits the interface to the parser.
  - HI\_FORMAT\_INVOKE\_GET\_THUMBNAIL: This interface is used to obtain the video thumbnail. When the application calls the HiPlayer interface **HI\_SVR\_PLAYER\_Invoke** (**InvokeID** is **HI\_FORMAT\_INVOKE\_GET\_THUMBNAIL**), the HiPlayer transparently transmits the interface to the parser. If the parser supports the video thumbnail function, it needs to implement decoding and scaling of a video frame (any



frame in the video stream) in the interface. For details about how to obtain the video thumbnail, see "[Obtaining Video File Thumbnails](#)."

- HI\_FORMAT\_INVOKE\_SET\_REFERER
  - HI\_FORMAT\_INVOKE\_SET\_USERAGENT
  - HI\_FORMAT\_INVOKE\_SET\_COOKIE: This interface is used to set the Referer/UserAgent/Cookie information address (HTTP protocol). If the proprietary HTTP protocol is implemented, the parser needs to save the Referer/UserAgent/Cookie information in the interface and transmit the information to the HTTP protocol. For details about Referer/UserAgent/Cookie, see section [25.4.4 "Implementing the File Parser/Protocol"](#).
  - HI\_FORMAT\_INVOKE\_GET\_BANDWIDTH: If a proprietary network protocol is implemented and the function of obtaining bandwidth is required, the parser (or proprietary protocol) needs to implement the bandwidth statistic function. For details about how to obtain the bandwidth information, see "[Obtaining Network Bandwidth During Network Playback](#)."
  - HI\_FORMAT\_INVOKE\_GET\_BUFFER\_MAX\_SIZE/HI\_FORMAT\_INVOKE\_SET\_BUFFER\_MAX\_SIZE/HI\_FORMAT\_INVOKE\_GET\_BUFFER\_CONFIG/HI\_FORMAT\_INVOKE\_SET\_BUFFER\_CONFIG
- These four interfaces are related to network buffer configuration. For details, see section [25.4.1 "Basic Playback Process"](#). If the proprietary network protocol is implemented, and the parser has its own network buffer, the parser must implement the preceding four functions.
- ✧ When the network buffer is configured by playback duration, the parser can calculate the playback duration of data in the buffer based on the difference of the PTSs of the first frame and the last frame in the network buffer.
  - ✧ HI\_FORMAT\_INVOKE\_GET\_BUFFER\_MAX\_SIZE/HI\_FORMAT\_INVOKE\_SET\_BUFFER\_MAX\_SIZE is used to obtain/set the maximum size of the network buffer (byte). For details, see "[Setting the Network Buffer Size and Thresholds by Byte](#)" and "[Setting the Buffer Size and Thresholds Based on the Playback Duration](#)".
- HI\_FORMAT\_INVOKE\_GET\_BUFFER\_LAST PTS: This interface is used to obtain the PTS (in ms) of the last frame in the network buffer. If a proprietary network protocol is implemented and the parser has its own network buffer, the PTS of the last frame in the buffer (maximum PTS value in the buffer) is returned. Otherwise, HI\_FORMAT\_NO PTS is returned.
  - HI\_FORMAT\_INVOKE\_GET\_BUFFER\_STATUS: This interface is used to obtain the data amount (in byte) in the network buffer and the playback duration (in ms) of data in the buffer. If a proprietary network protocol is implemented and the parser has its own network buffer, this interface must be implemented. Otherwise, 0 is returned.
  - HI\_FORMAT\_INVOKE\_GET\_MSG\_EVENT: This interface is used to obtain the parser event. The data structure type is HI\_FORMAT\_MSG\_S, and the event type is HI\_FORMAT\_MSG\_TYPE\_E. If a proprietary network protocol is implemented, the parser needs to report at least two events: HI\_FORMAT\_MSG\_DOWNLOAD\_FINISH (data downloading completion event) and HI\_FORMAT\_MSG\_NETWORK (network error event). If there is no event, a failure code is returned.
- MY\_DEMUXER\_Close: This interface is used to close the parser. The parser needs to call the close interface of the protocol to close all opened files, disconnect from the network server, and release all resources in the interface.



- Implement the protocol interfaces. The member **stProtocolFun** of the global symbol **g\_stFormat\_entry** defines the protocol interfaces to be implemented. See the sample program **sample/localplay/test\_protocol.c**.
  - **MY\_URL\_Find**: This interface is similar to the find interface of the parser. The parameter is the playback URL. If the protocol supports the media file of the URL, OK is returned. Otherwise, a failure code is returned.
  - **MY\_URL\_Open**: The protocol needs to allocate the protocol handle in this interface and open a local file or connect to the server.
  - **MY\_URL\_Read**: This interface is used to read data by byte. The data buffer is allocated by the parser, and the protocol only needs to read data of the corresponding byte. If the data is less than the number of bytes specified in the read interface parameter, the number of actually read bytes is returned. If the EOF is reached and there is no data to be read, an error code is returned.
  - **MY\_URL\_Seek**: This interface is used to implement the seek operation based on the byte offset.
  - **MY\_URL\_ReadSeek**: This interface is not supported, and a failure code is returned when it is called.
  - **MY\_URL\_Invoke**: This interface is used to extend functions that cannot be implemented by other protocol interfaces. For example, during network playback, when the network buffer is full, the parser needs to call this protocol interface to request the server to pause data transmission.
  - **MY\_URL\_Close**: The protocol needs to close the opened files, disconnect from the server, and release all resources in this interface.
- Compile the dynamic library **libxxx.so** (for example, **libmydemuxer.so**). Copy the created .c file (for example, **mydemuxer.c**) to **sample/localplay/**, and modify **sample/localplay/Makefile** (for details, see **libprivateprotocol.so**).

```
LIB_FILE := ./libprivateprotocol.so
MY_DEMUX_LIB := ./libmydemuxer.so

all: $(IMG_FILE) $(LIB_FILE) $(MY_DEMUXER_LIB)

$(LIB_FILE): test_protocol.c
@$(CC) $(CFLAGS) -shared -fPIC -o $@ $^

$(MY_DEMUX_LIB): mydemuxer.c
@$(CC) $(CFLAGS) -shared -fPIC -o $@ $^

clean:
-@rm -f $(IMG_FILE)
-@rm -f $(LIB_FILE)
-@rm -f $(MY_DEMUX_LIB)
```

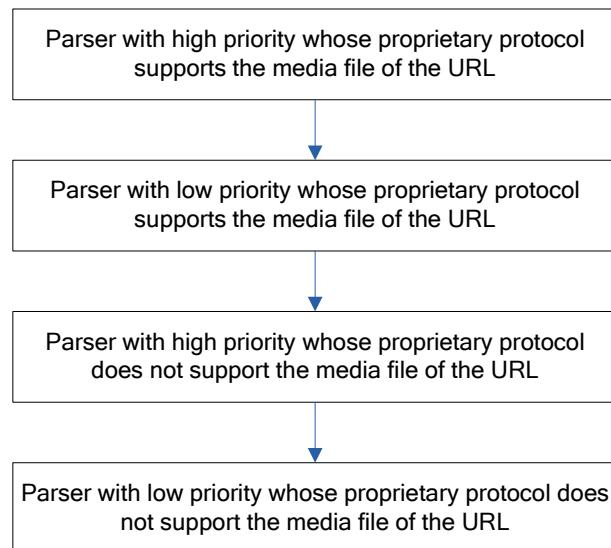
- Copy the compiled library file **libmydemuxer.so** to the board **/usr/lib/** directory. The application then can call **HI\_SVR\_PLAYER\_RegisterDynamic** in its own code to mount the parser.



## Notes

- The priority (`g_stFormat_entry.u32Merit`) of the file parser/protocol ranges from 0x0 to 0xFFFFFFFF. The larger the value, the higher the priority. Among the file parsers for which a success code is returned when the HiPlayer calls the parser find/open/FindStream interface, the file parser with a higher priority is selected. The HiPlayer transmits the protocol interface address (`&g_stFormat_entry.stProtocolFun`) of the parser to the parser in priority by using the parser find/open interface parameter. Therefore, the protocol is searched based on the protocol priority only when all the proprietary protocols used by parsers do not support the media file of the URL. For details, see [Figure 25-10](#).

**Figure 25-10** Selecting the HiPlayer parser



- Before deinitializing the HiPlayer, unload the parser dynamic library by calling `HI_SVR_PLAYER_UnRegisterDynamic`. (If you call this API and set the unmounting type to `HI_SVR_PLAYER_DLL_PARSER`, all dynamic libraries registered by calling `HI_SVR_PLAYER_RegisterDynamic` are unmounted.)

## Loading the File Parser/Protocol

```
...
/* Register the parsers provided by HiSilicon */
s32Ret = HI_SVR_PLAYER_RegisterDynamic(HI_SVR_PLAYER_DLL_PARSER,
"libformat.so");
/* Register other parsers */
s32Ret = HI_SVR_PLAYER_RegisterDynamic(HI_SVR_PLAYER_DLL_PARSER,
"libmydemuxer.so");
...
```



## 25.4.5 HTTP Playback

### Scenario

The HiPlayer supports HTTP 1.1. You can install an HTTP server on the PC, save the media file to be played in the corresponding directory, and then create a HiPlayer to play the media file by using the HTTP protocol.

### Working Process

The HTTP playback process complies with the HiPlayer basic playback process (see section [25.4.1 "Basic Playback Process."](#)).

The following functions are also supported by using the HI\_SVR\_PLAYER\_Invoke interface:

- Sets cookie information.
- Sets the UserAgent. Some servers require specific UserAgent. In this case, the application can set the UserAgent by calling the Invoke interface. If the UserAgent is not set, the HiPlayer uses the default UserAgent.
- Sets the Referer. The Referer is used to specify the source of the current VOD address. For details, see the RFC 2068 Referer chapter.

The preceding information must be set before the VOD address is set by calling HI\_SVR\_PLAYER\_SetMedia and cannot be dynamically set during playback.



The HTTPS protocol also supports the preceding functions.

### Notes

After the corresponding Invoke interface for setting the UserAgent or Referer is returned, the application can release the memory for the corresponding character string. However, for the Invoke interface for setting the cookie information, the corresponding memory can be released only when the HI\_SVR\_PLAYER\_SetMedia interface which is called later is returned.

### Setting Cookie Information

```
HI_HANDLE hPlayer = (HI_HANDLE)NULL,  
HI_CHAR *pCookies = NULL;  
...  
/*Create a HiPlayer.*/  
HI_SVR_PLAYER_Create(..., &hPlayer);  
...  
pCookies = malloc(1024);  
snprintf(pCookies, 1024, "%s\r\n", "Cookie:set-cookie: uid = zhangsan;  
Max-Age=3600; Domain=.xxx.org; Path=/bbs; Version=1");  
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer, HI_FORMAT_INVOKE_SET_COOKIE,  
pCookies);  
if (HI_SUCCESS != s32Ret)  
{
```



```
    printf("set cookies failed\n");
}
...
HI_SVR_PLAYER_SetMedia(hPlayer, ...);
/*Release the memory after SetMedia.*/
free(pCookies);
pCookies = NULL;
...
```

## 25.4.6 HTTPS Bidirectional Certification

### Scenario

The HiPlayer integrates the OpenSSL and supports the HTTPS protocol.

The dependent relationship when the HiPlayer plays audio/video from HTTPS links is as follows:



**libssl.so** and **libcrypto.so** are from OpenSSL.

### Working Process

To play HTTPS links by using the HiPlayer, do as follows:

- (Optional) Modify the OpenSSL adaptation layer libtlsadp as required.

The code of the OpenSSL adaptation layer is open to users to facilitate user extension. The compilation of libtlsadp depends on the OpenSSL header file and the library files **libssl.so** and **libcrypt.so** generated after OpenSSL compilation. To compile the libtlsadp, perform the following steps:

**Step 1** Configure the OpenSSL by running the following commands in OpenSSL.

```
AR="arm-hisiv200-linux-ar" NM=arm-hisiv200-linux-nm RANLIB=arm-hisiv200-
linux-ranlib ./Configure os/compiler:arm-hisiv200-linux-gcc shared -
march=armv7-a -mcpu=cortex-a9 -mfloating-abi=softfp -mfpu=vfpv3-d16 -
DOPENSSL_NO_STATIC_ENGINE no-cipher no-sse2 no-krb5 zlib-dynamic no-idea
no-mdc2 no-rc5
```

**Step 2** Modify the **Makefile** of OpenSSL.

1. Change **SHLIB\_EXT=** to **SHLIB\_EXT=.so.\$(SHLIB\_MAJOR).\$(SHLIB\_MINOR)**.



2. Change **SHLIB\_TARGET**= to **SHLIB\_TARGET=linux-shared**.
3. Add **-fPIC -DOPENSSL\_PIC** to **CFLAG**.
4. Change **EX\_LIBS**= to **EX\_LIBS= -ldl**.
5. Change **SHARED\_LIBS**= to **SHARED\_LIBS=\$(SHARED\_CRYPTO) \$(SHARED\_SSL)**.
6. Change **SHARED\_LIBS\_LINK\_EXTS**= to **SHARED\_LIBS\_LINK\_EXTS=.so.\$(SHLIB\_MAJOR).so**.
7. Change **SHARED\_LDFLAGS**= to **SHARED\_LDFLAGS=-rpatch-link=libssl:libcrypto**.

**Step 3** Copy the zlib header files **zconf.h** and **zlib.h** in the **pub/include** directory of the SDK to the OpenSSL root directory.

**Step 4** Compile OpenSSL.

Run the make clean and make commands. Note that you cannot use multiple threads to compile. For example, make -j60.

**Step 5** Compile libtsadp.

Modify the path for the OpenSSL header file and library files in Makefile of the libtsadp directory, and compile libtsadp.

**Step 6** Copy **libssl.so.1.0.0**, **libcrypto.so.1.0.0**, and **libtsadp.so** to **usr/lib** in the directory of the player to replace the original files.

----End

Store the certificates.

The licenses for the HiPlayer released versions are stored in **/system/license**. The licenses are named as follows:

- **ca.crt**: CA certificate
- **client1.crt**: client certificate
- **client1.key**: client key

If the client CA certificate consists of multiple certificates, package the certificates into a file named **ca.crt** by using the following scripts.

```
***** begin *****  
#!/bin/sh  
rm ca.crt  
for i in ca-client.crt ca-server.crt; do  
    openssl x509 -in $i -text >> ca.crt  
done  
chmod 777 ca.crt
```

- Set the date and time for the development board/player.
- Play the audio/video from the HTTPS link by calling the HiPlayer interface.

This step is the same as the basic playback process of the HiPlayer. For details, see section [25.4.1 "Basic Playback Process."](#)



## Notes

If the network environment and certificate storage are normal, but the HTTPS link cannot be played, the date and time of the client may be incorrect.

## Playing HTTPS Links by Using Localplay

After the preceding preparations, connect the serial port of the development board/player and the network cable, copy **sample\_localplay** to the development board/player, and run **sample\_localplay** on the client:

```
/sample_localplay https://10.67.225.43/mkv/[OpfansMaplesnow].mkv
```



The path for localplay is **sample/localplay/**.

## 25.4.7 HLS Playback

### Scenario

The HTTP Live Streaming (HLS) is a media streaming protocol that enables adaptive bit rates implemented by Apple Inc. An HLS media source contains one or multiple M3U8 files and many TS chunk files, and even files with encrypted information. A media source typically contains multiple HLS streams with the same audio/video content but different bit rates. By using the HLS technology, streams with a low bit rate can be played when the bandwidth is low, and streams with a high bit rate can be played when the bandwidth is high. The HiPlayer support HLS version 2 and the playback of AES-128 encrypted streams.

### Working Process

The HLS playback process complies with the HiPlayer basic playback process (see section [25.4.1 "Basic Playback Process."](#)). The HiPlayer can play HLS files locally or by using the HTTP protocol. When the HLS file is played by using the HTTP protocol, the cookie/UserAgent/Referer information can also be set. For details, see section [25.4.4 "Implementing the File Parser/Protocol."](#) The HLS media source directory typically contains an **index.m3u8** file and multiple **N.m3u8** files (*N* indicates 01, 02, ...). **index.m3u8** contains the index information of all HLS streams, and **N.m3u8** indicates the HLS streams that require different bandwidth. The URL for the HLS media source is in the format of <http://Server IP address+Path/index.m3u8> or *Local HLS path/index.m3u8*.



Figure 25-11 HLS media source

04.m3u8	14 KB
03.m3u8	14 KB
02.m3u8	14 KB
01.m3u8	14 KB
00_index.m3u8	1 KB
04-612.ts	150 KB
04-611.ts	248 KB
04-610.ts	247 KB
04-609.ts	247 KB
04-608.ts	248 KB
04-607.ts	247 KB
04-606.ts	247 KB
04-605.ts	248 KB
04-604.ts	247 KB
04-603.ts	248 KB
04-602.ts	225 KB
04-601.ts	218 KB
04-600.ts	241 KB
04-599.ts	306 KB
04-598.ts	248 KB
04-597.ts	248 KB

Figure 25-12 HLS file index.m3u8

```
#EXTM3U
#EXT-X-VERSION:1
#EXT-X-STREAM-INF:PROGRAM-ID=15,BANDWIDTH=494816
01.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=15,BANDWIDTH=612128
02.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=15,BANDWIDTH=818176
03.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=15,BANDWIDTH=1021216
04.m3u8
```



**Figure 25-13** HLS file 01.m3u8

```
#EXTM3U
#EXT-X-TARGETDURATION:2
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:2,
01-0.ts
#EXTINF:2,
01-1.ts
#EXTINF:2,
01-2.ts
#EXTINF:2,
01-3.ts
#EXTINF:2,
01-4.ts
#EXTINF:2,
01-5.ts
#EXTINF:2,
01-6.ts
#EXTINF:2,
01-7.ts
```

Before the HLS playback URL is set by calling HI\_SVR\_PLAYER\_SetMedia, you can set the fastboot mode by calling the corresponding HI\_SVR\_PLAYER\_Invoke interface (**InvokeID** is **HI\_FORMAT\_INVOKE\_SET\_HLS\_START\_MODE**). After the fastboot mode is set, the HiPlayer plays the HLS streams starting from the one that requires the lowest bandwidth.

After the HLS playback URL is set by calling HI\_SVR\_PLAYER\_SetMedia, you can specify the HLS stream to be played by calling the corresponding HI\_SVR\_PLAYER\_Invoke interface (**InvokeID** is **HI\_FORMAT\_INVOKE\_SET\_HLS\_STREAM**). If the specified stream index is -1, the HiPlayer automatically plays the most appropriate stream based on the current bandwidth. Otherwise, the specified HLS stream is played. For details, see the following sample.

## Notes

If a specific HLS stream is specified in the URL transferred to the HiPlayer by using the HI\_SVR\_PLAYER\_SetMedia interface, for example, <http://Server IP address+Path/N.m3u8> or *Local HLS path/N.m3u8*, the HiPlayer plays only this HLS stream and cannot dynamically select the most appropriate stream based on the current bandwidth even the application sets the stream index to -1 by using the HI\_SVR\_PLAYER\_Invoke interface.

## Playing HLS Streams

```
HI_HANDLE hPlayer = (HI_HANDLE)NULL,  
HI_SVR_PLAYER_MEDIA_S stMedia;  
HI_S32 i;  
HI_S32 s32HlsStreamNum = 0;  
HI_S32 s32HlsTrack = -1;  
HI_FORMAT_HLS_STREAM_INFO_S stStreamInfo;  
...  
/*Create a HiPlayer.*/
```



```
HI_SVR_PLAYER_Create(..., &hPlayer);
...
/*Set the fastboot mode.*/
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer,
HI_FORMAT_INVOKE_SET_HLS_START_MODE, (HI_VOID*)HI_FORMAT_HLS_MODE_FAST);
if (HI_SUCCESS != s32Ret)
{
    printf("set hls fast start mode failed\n");
}
...
memset(&(stMedia.aszUrl), 0, sizeof(stMedia.aszUrl));
sprintf(stMedia.aszUrl, "%s",
"http://10.67.212.135/panda4stream/index.m3u8");
HI_SVR_PLAYER_SetMedia(hPlayer, HI_SVR_PLAYER_MEDIA_STREAMFILE, &stMedia);
if (HI_SUCCESS != s32Ret)
{
    printf("set media failed\n");
    return HI_FAILURE;
}
...
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer,
HI_FORMAT_INVOKE_GET_HLS_STREAM_NUM, &s32HlsStreamNum);
if (HI_SUCCESS != s32Ret)
{
    printf("set media failed\n");
    return HI_FAILURE;
}
/*Obtain HLS stream information.*/
for (i = 0; i < s32HlsStreamNum; i++)
{
    stStreamInfo.stream_nb = i;
    s32Ret = HI_SVR_PLAYER_Invoke(hPlayer,
HI_FORMAT_INVOKE_GET_HLS_STREAM_INFO, &stStreamInfo);
    if (HI_SUCCESS != s32Ret)
    {
        continue;
    }
    printf("\nHls stream number is: %d \n", stStreamInfo.stream_nb);
    printf("URL: %s \n", stStreamInfo.url);
    printf("BandWidth: %lld \n", stStreamInfo.bandwidth);
    printf("SegmentNum: %d \n", stStreamInfo.hls_segment_nb);
}
/*Specify the HLS stream to be played. If the specified index is -1, the
HiPlayer automatically plays the most appropriate stream based on the
```



```
bandwidth.*/  
s32HlsTrack = 0;  
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer, HI_FORMAT_INVOKE_SET_HLS_STREAM,  
(HI_VOID*) s32HlsTrack);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("set hls track to %d failed\n", s32HlsTrack);  
}  
...  
...
```

## 25.4.8 Seamless M3U9 Playback

### Scenario

M3U9 is a simple proprietary protocol provided by the HiPlayer. The application creates a list of playback addresses (HTTP links or local file paths) based on the protocol rules. The HiPlayer automatically plays the media files in the playback addresses of the list consecutively without interruption or intermittence. However, there are some restrictions:

- The formats of media files (container formats) in the same list must be the same. The container formats can be .flv, .mp4, .ts, .m2ts, or .vob.
- The numbers of programs, audio streams, video streams, and subtitle streams in the files of the same list must be consistent, and the encoding formats must also be the same. That is, the slices are divided from the same file. However, the video resolution of each slice can be different.

The postfix of the list file is .m3u9. You can create an M3U9 list file based on the following example.

```
#HISIPLAY  
#HISIPLAY_START_SEAMLESS  
#HISIPLAY_STREAM:2.0  
http://10.157.187.1/panda4stream/01.ts  
#HISIPLAY_STREAM:2.0  
http://10.157.187.1/panda4stream/02.ts  
#HISIPLAY_STREAM:2.0  
http://10.157.187.1/panda4stream/03.ts  
#HISIPLAY_ENDLIST
```

The M3U9 list file format is described as follows:

- "#HISIPLAY" indicates a list for seamless playback.
- "#HISIPLAY\_START\_SEAMLESS" indicates the start of the list.
- "#HISIPLAY\_ENDLIST" indicates the end of the list.
- "#HISIPLAY\_STREAM" indicates that a file URL is displayed in the next row. The file URL can be a local path or HTTP link. "#HISIPLAY\_STREAM" is followed by a number that indicates the playback duration of the file in the URL.



## Working Process

The seamless playback process complies with the HiPlayer basic playback process (see section [25.4.1 "Basic Playback Process."](#)).

The HiPlayer can play files in the M3U9 list locally or by using the HTTP protocol. The playback URL format is <http://Server IP address+Path/01.m3u9> or <Local HLS path/01.m3u9>. When the files are played by using the HTTP protocol, the **cookie/UserAgent/Referer** information can also be set. For details, see section [25.4.4 "Implementing the File Parser/Protocol."](#)

## Notes

None

## 25.4.9 RTSP Playback

### Scenario

The RTSP protocol transmits control information by using the TCP connection, and transmits media data by using the TCP or UDP connection.

## Working Process

The RTSP playback process complies with the HiPlayer basic playback process (see section [25.4.1 "Basic Playback Process."](#)).

## Notes

- By default, the HiPlayer automatically negotiates with the server and selects the media data transmission protocol supported by the server. If **?tcp** or **?udp** is added after the RTSP URL (for example, `rtsp://10.67.216.235/asf/bringiton.asf?tcp`) when `HI_SVR_PLAYER_SetMedia` is called, the HiPlayer negotiates with the server by using the specified protocol. If the server does not support media data transmission by using the protocol, a failure code is returned.
- The RTSP protocol requires that the client proactively send heartbeat packets to the server. When the server does not receive heartbeat packets from the client for a time period (typically one minute), it disconnects from the client. When the HiPlayer is stopped, it does not send heartbeat packets to the server. Therefore, if the file needs to be replayed after the playback is complete, or `HI_SVR_PLAYER_Play` is called to play the file after the playback is stopped by calling `HI_SVR_PLAYER_Stop`, the RTSP URL must be reconfigured by calling `HI_SVR_PLAYER_SetMedia`. Then the HiPlayer reconnects to the server. After `HI_SVR_PLAYER_SetMedia` is called, you need to reconfigure the subtitle output function by calling `HI_UNF_SO_RegOnDrawCb`.
- The HiPlayer does not implement reconnection when the network is disconnected for the RTSP playback. When the network is disconnected, the HiPlayer reports the `HI_SVR_PLAYER_EVENT_NETWORK_INFO` event by using the event callback function (registered by calling `HI_SVR_PLAYER_RegCallback`, see [Figure 25-3](#)). The parameter contains the event type `HI_FORMAT_MSG_NETWORK_ERROR_DISCONNECT`. When the application receives the event, it implements reconnection by calling `HI_SVR_PLAYER_SetMedia`. After `HI_SVR_PLAYER_SetMedia` is called, the subtitle output function needs to be reconfigured by calling `HI_UNF_SO_RegOnDrawCb`. For details about the implementation, see [sample/localplay/sample\\_localplay.c](#).



## 25.4.10 MMS Playback

### Scenario

MMS is a media streaming protocol developed by Microsoft. The MMS protocol transmits control information by using the HTTP or TCP, and transmits media data by using the HTTP, TCP, or UDP.

### Working Process

The MMS playback process complies with the HiPlayer basic playback process (see section [25.4.1 "Basic Playback Process"](#)). **Table 25-3** describes the relationship among the MMS URL prefix, transmission protocol, and HiPlayer operation when the MMS URL is set by calling `HI_SVR_PLAYER_SetMedia`.

**Table 25-3** Relationship among the MMS URL prefix, transmission protocol, and HiPlayer operation

URL Prefix	Media Data Transmission Protocol	Control Information Transmission Protocol	HiPlayer Operation
mmst://	TCP	TCP	Returns an error code when the negotiation with the server fails.
mmsu://	UDP	TCP	Returns an error code when the negotiation with the server fails.
mmsh://	HTTP	HTTP	Returns an error code when the negotiation with the server fails.
mms://	To be determined	To be determined	Tries the MMST, MMSH, MMSU, and RTSP protocols one by one until the operation succeeds, or returns an error code when all attempts fail.

The MMS playback is similar to the HTTP playback, and it also supports the configuration of the UserAgent. For details, see section [25.4.4 "Implementing the File Parser/Protocol."](#)

### Notes

- The MMS protocol requires that the client proactively respond to the heartbeat packets sent by the server. When the server does not receive the response from the client for a time period (typically one minute), it disconnects from the client. When the HiPlayer is stopped, it does not respond to the heartbeat packets sent by the server. Therefore, if the file needs to be replayed after the playback is complete, or `HI_SVR_PLAYER_Play` is called to play the file after the playback is stopped by calling `HI_SVR_PLAYER_Stop`, the RTSP URL must be reconfigured by calling `HI_SVR_PLAYER_SetMedia`. Then the HiPlayer reconnects to the server. After `HI_SVR_PLAYER_SetMedia` is called, you need to reconfigure the subtitle output function by calling `HI_UNF_SO_RegOnDrawCb`.



## 25.4.11 RTMP Playback

### Scenario

RTMP is a media streaming protocol developed by Macromedia that transmits audio/video data and other data between the flash player and the server. It uses the HTTP or TCP protocol as the transmission protocol and has two encryption methods.

### Working Process

The RTMP playback process complies with the HiPlayer basic playback process (see section [25.4.1 "Basic Playback Process"](#)). The RTMP protocol includes the following types: RTMP, RTMPS, RTMPE, RTMPT, and RTMPTE. [Table 25-4](#) describes the relationship among the RTMP URL prefix, transmission protocol, and encryption mode when the RTMP URL is set by calling `HI_SVR_PLAYER_SetMedia`.

**Table 25-4** Relationship among the RTMP URL prefix, transmission protocol, and encryption mode

URL Prefix	Transmission Protocol	Encryption Mode
rtmp://	TCP	Not encrypted
rtmpe://	TCP	Adobe encryption mode RTMPE
rtmpt://	HTTP	Not encrypted
rtmpete://	HTTP	Adobe encryption mode RTMPE
rtmps://	TCP	TLS/SSL encryption

Similar to the HTTP playback, the RTMPT and RTMPTE playback modes also support the configuration of the UserAgent. For details, see section [25.4.4 "Implementing the File Parser/Protocol."](#)

### Notes

If the file needs to be replayed after the playback is complete, or `HI_SVR_PLAYER_Play` is called to play the file after the playback is stopped by calling `HI_SVR_PLAYER_Stop`, the RTMP URL must be reconfigured by calling `HI_SVR_PLAYER_SetMedia`. Then the HiPlayer reconnects to the server. After `HI_SVR_PLAYER_SetMedia` is called, you need to reconfigure the subtitle output function by calling `HI_UNF_SO_RegOnDrawCb`.

## 25.4.12 Mounting ISO Files in UDF Format

### Scenario

Some streams such as blu-ray streams may be stored as ISO files in UDF format. The ISO files must be mounted to a specified directory before being played.



## Working Process

The working process is as follows:

Mount the blu-ray ISO file to the customized directory of the development board/player.

```
mkdir /mnt/iso/;mount -o loop -t udf /mnt/usb/AVATAR.iso /mnt/iso/
```

**Step 1** Play the mounted blu-ray file by calling the HiPlayer interface.

**Step 2** This step is the same as the process of playing local files, except that the "bluray" prefix and "?playlist=-1" postfix must be added to the path of streams to be played, for example, **bluray:/mnt/iso/?playlist=-1**. For details, see section [25.4.1 "Basic Playback Process"](#).



The path for localplay is **sample/localplay/**.

**----End**

## Notes

When blu-ray streams are played, the "bluray" prefix and "?playlist=-1" postfix must be added to the path of streams to be played, and space is not allowed between the prefix and the path and between the postfix and the path.

Mounting ISO files requires support of the kernel. If error information similar to "mount xxxx failed, No such device" is displayed during the mounting process, the kernel does not support the mount operation. In this case, reconfigure and burn the kernel by performing the following steps:

- Search for the macro **CFG\_HI\_KERNEL\_CFG** in the **cfg.mak** file of the SDK root directory. The value of this macro is the name of the current kernel configuration file, which is typically stored in **source/kernel/linux-x.x.x/arch/arm/configs**. (*x.x.x* indicates the kernel version.)
- Switch to the kernel directory (**source/kernel/linux-x.x.x/**), and copy the current kernel configuration file in **arch/arm/configs/** to replace the .config file in the current directory.

**Step 1** Run the **menuconfig** command.

```
make menuconfig ARCH=arm
```



Figure 25-14 Running make menuconfig ARCH=arm

```
Linux/arm 3.4.35 Kernel Configuration
menus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> ex
uilt-in [ ] excluded <M> module <> module capable

General setup --->
[*] Enable loadable module support --->
-*= Enable the block layer --->
System Type --->
[ ] FIQ Mode Serial Debugger
Bus support --->
Kernel Features --->
Boot options --->
CPU Power Management --->
Floating point emulation --->
Userspace binary formats --->
Power management options --->
[*] Networking support --->
Device Drivers --->
File systems --->
Kernel hacking --->
Security options --->
-*= Cryptographic API --->
Library routines --->
---
Load an Alternate Configuration File
v(+)

<Select> < Exit > < Help >
```

**Step 2** Select File systems.



Figure 25-15 File systems

```
File systems
submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <M>
] built-in [ ] excluded <M> module <> module capable

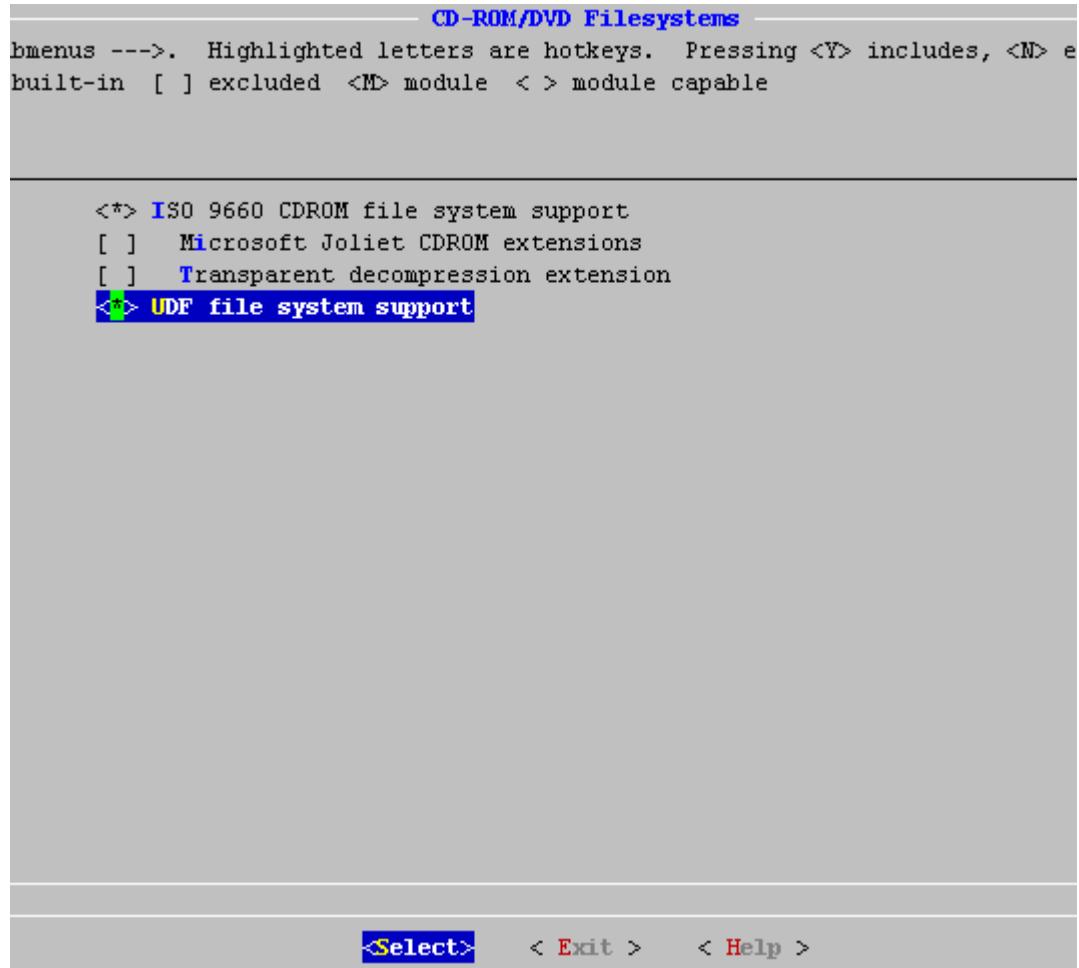
^(-)
[ ] XFS Debugging support (EXPERIMENTAL)
< > GFS2 file system support
< > OCFS2 file system support
< > Btrfs filesystem (EXPERIMENTAL) Unstable disk format
< > NILFS2 file system support (EXPERIMENTAL)
[*] Dnotify support
[*] Inotify support for userspace
[ ] Filesystem wide access notification
[*] Quota support
[ ] Report quota messages through netlink interface
[*] Print quota warnings to console (OBSOLETE)
[ ] Additional quota sanity checks
<M> Old quota format support
<M> Quota format vfstab and vfstab support
<M> Kernel automounter version 4 support (also supports v3)
<*> FUSE (Filesystem in Userspace) support
< > Character device in Userspace support
    Caches --->
    CD-ROM/DVD Filesystems --->
        DOS/FAT/NT Filesystems --->
            Pseudo filesystems --->
v(+)

Select < Exit > < Help >
```

**Step 3** Select **CD-ROM/DVD Filesystems**, enter a space, and select **UDF file system support** (until \* appears in the angle brackets <> on the left of the option).



Figure 25-16 CD-ROM/DVD Filesystems



**Step 4** Exit and save the configuration. The new configuration is saved in the .config file of the current directory.

**Step 5** Copy the .config file to the **arch/arm/configs/** directory to replace the current kernel configuration file.

**Step 6** Switch to the SDK root directory and recompile the kernel by running **make linux\_clean;make linux\_install**.

**Step 7** Reburn the kernel image.

**Step 8** Restart the development board/player, and mount the ISO file.

```
mkdir /mnt/iso/;mount -o loop -t udf /mnt/usb/AVATAR.iso /mnt/iso/
```

----End

## Playing Blu-ray Streams by Using Localplay

To play blu-ray streams by using Localplay, perform the following steps:

**Step 1** Connect the serial port of the development board/player and the network cable, and copy **sample\_localplay** to the development board/player.



**Step 2** Mount the file if the stream file is the ISO file.

```
mkdir /mnt/iso/;mount -o loop -t udf /mnt/usb/AVATAR.iso /mnt/iso/
```

**Step 3** Play the blu-ray streams by using localplay.

```
./sample_localplay bluray:/mnt/iso/?playlist=-1
```

If the streams to be played are blu-ray streams, for example, if the blu-ray file to be played is in `/mnt/usb/AVATAR/`, play the streams directly and mounting is not required.

```
/sample_locaplay bluray:/mnt/usb/AVATAR/?playlist=-1
```

**NOTE**

The path for localplay is `sample/localplay/`.

----End

## 25.4.13 Playback of 3D Media Sources

### Scenario

The HiPlayer supports the playback of 3D media sources in the format of MVC, SBS, or TAB. Files in MVC format are typically blu-ray files. For details about the playback of blu-ray files, see section [25.4.11 "RTMP Playback."](#)

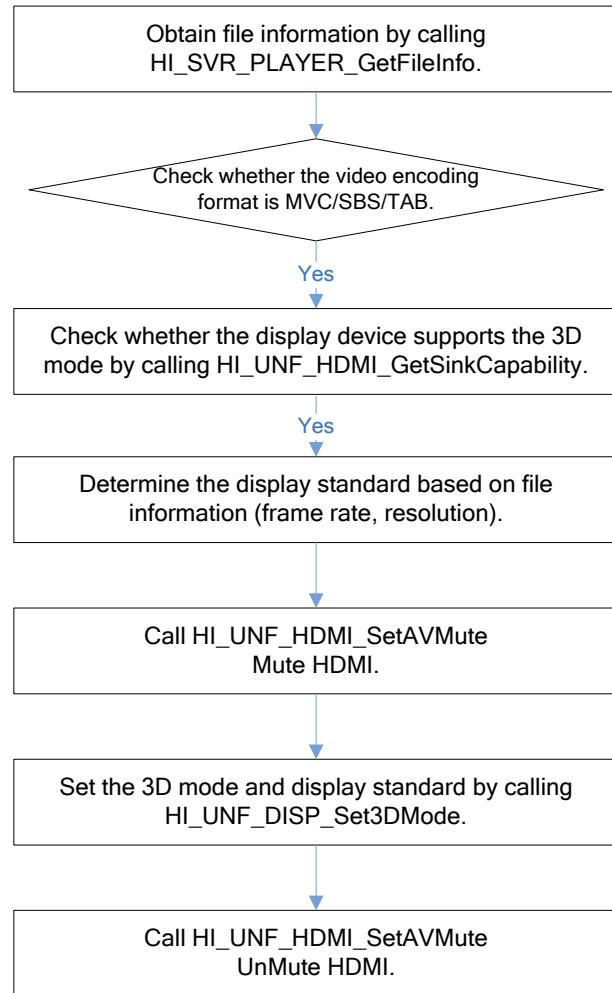
### Working Process

The 3D media playback process complies with the HiPlayer basic playback process (see section [25.4.1 "Basic Playback Process"](#)). In addition, operations shown in [Figure 25-17](#) need to be performed after the playback URL is set by calling `HI_SVR_PLAYER_SetMedia`. These operations can be dynamically performed during playback.

Files whose video encoding format is MVC are 3D media sources. Therefore, if the video encoding format obtained by calling `HI_SVR_PLAYER_GetFileInfo` is MVC, the application can set the 3D mode and display standard by directly calling the related API. However, if the video encoding format is SBS or TAB, you need to observe whether the video is displayed as two images (left/right or top/bottom). If yes, you can then set the corresponding 3D mode and display standard.



**Figure 25-17** Playback process for 3D media sources



## Notes

The 3D modes and display standards to be set for the MVC, SBS, and TAB formats are different. For details about how to set the 3D mode and switch between the 3D mode and 2D mode, see the implementation in **sample/localplay/sample\_localplay.c** (search for **HI\_UNF\_DISP\_Set3DMode** in **sample\_localplay.c**).

## 25.4.14 HiDrmEngine

### Scenario

Digital rights management (DRM) is a digital copyright encryption protection technology. Media files encrypted by using the DRM can be played properly only after being decrypted using the DRM. The HiPlayer integrates the PlayReady technology of Microsoft.

As a DRM framework, the HiDrmEngine provides unified DRM interfaces for applications and encapsulates and manages the WMDRM and PlayReady as plug-ins. See [Figure 25-18](#).

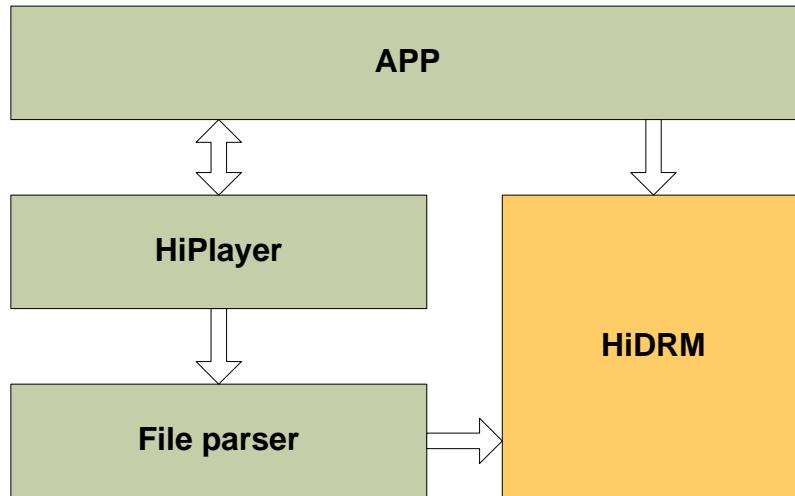
- The file parser parses media files and decrypts audio and video data by calling the HiDrmEngine interface in the HiPlayer.



- The HiPlayer controls the decoding, synchronization, output, and subtitle synchronization of audio and video data. When a player is being created, the HiPlayer transmits the HiDrmEngine to the file parser in pass-through mode.

Applications can perform operations such as configuring the DRM component by calling the HiDrmEngine interfaces.

**Figure 25-18** HiDrmEngine application architecture

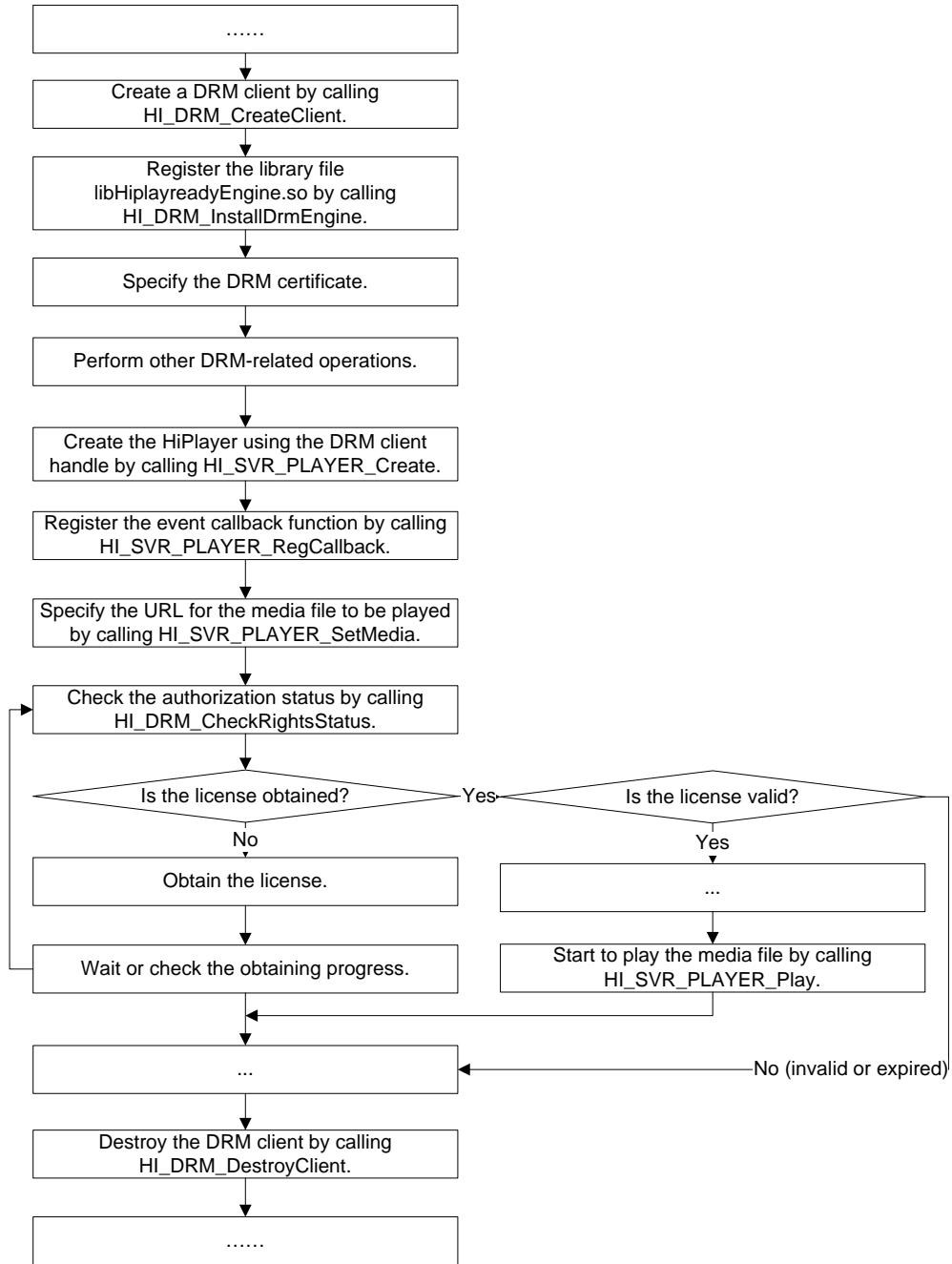


## Working Process

[Figure 25-19](#) shows the working process of the HiDrmEngine. For details, see the DRM-related code in `sample/localplay/sample_localplay.c`.



Figure 25-19 HiDrmEngine working process



The HiDrmEngine works as follows:

**Step 1** Create a DRM client.

Perform this step before creating the HiPlayer.

**Step 2** Register the DRMEngine library files.

Skip this step if the library file (for example, **libHiplayreadyEngine.so**) is stored in **/system/lib/drm/** because the HiPlayer will register it automatically. Otherwise, you need to register the library files by calling an API.



**Step 3** Specify the DRM certificate.

You can use a certificate file or external certificate. Only PlayReady supports external certificates. [Figure 25-20](#) shows the configuration process.

- Skip this step if the DRM certificate is stored in `/system/bin/prpd/` because the system will automatically read the certificate. Otherwise, you need to specify the path of the certificate. For details, see [Setting the PlayReady Certificate Path](#).
- For details about how to use the external certificate (store the DRM certificate data in the memory), see [Using the PlayReady External Certificate](#).

**Step 4** Perform other DRM-related operations.

The DRM requests based on PlayReady are as follows: set the domain certificate and metering certificate, send the metering report after decryption and playback, and set the envelop file to be decrypted and the output file after decryption. For details about the operation process, see [Figure 25-20](#). For details about the samples, see [Setting the PlayReady Domain Certificate](#) and [Decrypting the PlayReady Envelop File](#).

**Step 5** Create the HiPlayer by using the DRM client handle.

The HiPlayer transmits the DRM client handle to the parser in pass-through mode, and the parser calls the corresponding interface of the DRM client for decryption when the DRM-encrypted streams are being played.

**Step 6** Obtain the current DRM authorization status.

The authorization status may include the following: the authorization information is not obtained; the authorization is valid; the authorization is invalid; the authorization has expired; other errors.

**Step 7** Perform operations based on the DRM authorization status.

- If the authorization information is not obtained, obtain the authorization information first. This is an asynchronous operation. You can check the obtaining progress periodically or wait until the information is obtained. For details, see [Figure 25-20](#) and [Obtaining DRM Authorization Information](#).
- If the authorization is valid, the playback is started.
- If the authorization is invalid, has expired, or another error occurs, the playback cannot be started.

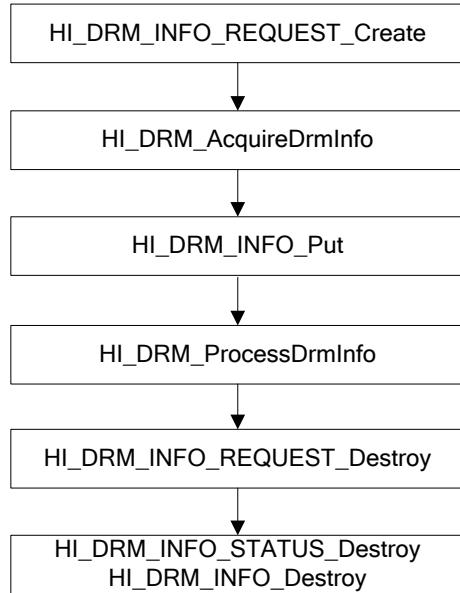
**Step 8** Destroy the DRM client and exit the playback process.

----End

[Figure 25-20](#) shows the common request process of the HiDrmEngine.



Figure 25-20 HiDrmEngine DRM request process



The procedure is as follows:

**Step 1** Create a DRM info request.

You need to specify the DRM mime type and DRM info type (request type).

**Step 2** Obtain DRM info.

**Step 3** Put the key-value that represents the DRM request to the DRM info.

Both key and value are character string types. Each key is declared in `hi_drm_common.h` as a pointer variable. Do not directly use the constant character strings to which these pointer variables point if possible. The following describes some frequently used keys:

- `HI_KEY_PDY_DrmPath`: Sets the PlayReady certificate path. The value of this key is the save path of the PlayReady certificate. The path must end with "/", for example, `/system/bin/prpd/`, but not `/system/bin/prpd`.
- `HI_KEY_PDY_LocalStoreFile`: Sets the PlayReady local file. The value of this key is the name of the DRM local file with full path.
- `HI_KEY_PDY_InitiatorFile`: Sets the domain or metering certificate of the PlayReady. The value of this key is the full path file name of the domain or metering certificate, for example, `/mnt/usb/JoinDomain.cms`, `/mnt/usb/LeaveDomain.cms`, or `/mnt/usb/Metering.cms`.
- `HI_KEY_PDY_EnvelopFile`: Sets the envelop file for decryption. The value of this key is the full path file name of the envelop file, for example, `/mnt/usb/xxx.pye`.
- `HI_KEY_PDY_EnvelopOutputFile`: Sets the output file after decryption. The value of this key is a full path file name, for example, `/mnt/usb/xxx.asf`.

**Step 4** Process DRM info.

If the DRM info request is to obtain the DRM authorization information, it is an asynchronous operation. For details, see [Obtaining DRM Authorization Information](#).



**Step 5** Destroy the DRM info request, DRM info, and DRM info status.

----End

## Notes

- Before using the HiDrmEngine, you need to apply for the PlayReady commercial device license from Microsoft. You cannot use the HiDrmEngine if the PlayReady device license is not obtained.
- Test files related to PlayReady can be obtained from the official website of Microsoft PlayReady. The files include audio/video files encrypted by using the PlayReady technology, the corresponding domain certificate, and the metering certificate.

## Setting the PlayReady Certificate Path

```
HI_S32 s32Ret = HI_SUCCESS;
HI_DRM_CLIENT_PPTR hClient = NULL;
HI_DRM_INFO_REQUEST_PPTR hRequest = NULL;
HI_DRM_INFO_PPTR hInfo = NULL;
HI_DRM_INFO_STATUS_PTR hInfoStatus = NULL;

s32Ret = HI_DRM_CreateClient(&hClient);
if (HI_SUCCESS != s32Ret)
{
    printf("create drm client return %d.\n", s32Ret);
    return s32Ret;
}

s32Ret = HI_DRM_INFO_REQUEST_Create(&hRequest, HI_DRM_REGISTRATION_INFO,
"application/vnd.ms-playready");
if (HI_SUCCESS != s32Ret)
{
    printf("create drm info request return %d.\n", s32Ret);
    goto fail;
}

hInfo = HI_DRM_AcquireDrmInfo(hClient, hRequest);
if (HI_SUCCESS != s32Ret)
{
    printf("acquire drm info return %d.\n", s32Ret);
    goto fail;
}

s32Ret = HI_DRM_INFO_Put(hInfo, HI_KEY_PDY_DrmPath, "./prpd/");
if (HI_SUCCESS != s32Ret)
{
    printf("put info key-value return %d.\n", s32Ret);
```



```
        goto fail;
    }

    s32Ret = HI_DRM_INFO_Put(hInfo, HI_KEY_PDY_LocalStoreFile,
    "./data.localstore");
    if (HI_SUCCESS != s32Ret)
    {
        printf("put info key-value return %d.\n", s32Ret);
        goto fail;
    }

    hInfoStatus = HI_DRM_ProcessDrmInfo(hClient, hInfo);
    if (NULL == hInfoStatus || hInfoStatus->statusCode != HI_DRM_STATUS_OK)
    {
        s32Ret = HI_FAILURE;
        printf("process drm info fail, status code:%d\n",
        hInfoStatus->statusCode);
        goto fail;
    }

fail:
    if (hRequest != NULL)
    {
        HI_DRM_INFO_REQUEST_Destroy(&hRequest);
    }
    if (hInfoStatus != NULL)
    {
        HI_DRM_INFO_STATUS_Destroy(hInfoStatus);
    }
    if (hInfo != NULL)
    {
        HI_DRM_INFO_Destroy(&hInfo);
    }
    if (hClient != NULL)
    {
        HI_DRM_DestroyClient(&hClient);
    }
}
```

## Using the PlayReady External Certificate

The application sends the address of the buffer for storing the PlayReady certificate to the HiDrmEngine. The path for storing the certificate file does not need to be configured.

```
HI_S32 s32Ret = HI_SUCCESS;
HI_DRM_CLIENT_PPTR hClient = NULL;
HI_DRM_INFO_REQUEST_PPTR hRequest = NULL;
```



```
HI_DRM_INFO_PPTR hInfo = NULL;
HI_DRM_INFO_STATUS_PTR hInfoStatus = NULL;
HI_DRM_BUFFER_S stCertsBuffer;
HI_PLAYREADY_DEVCERT_BUF_S *pstOpenParam = NULL;

s32Ret = HI_DRM_CreateClient(&hClient);
if (HI_SUCCESS != s32Ret)
{
    printf("create drm client return %d.\n", s32Ret);
    return s32Ret;
}

s32Ret = HI_DRM_INFO_REQUEST_Create(&hRequest,
HI_DRM_PLAYERREADY_EXTERN_CERT, "application/vnd.ms-playready");
if (HI_SUCCESS != s32Ret)
{
    printf("create drm info request return %d.\n", s32Ret);
    goto fail;
}

hInfo = HI_DRM_AcquireDrmInfo(hClient, hRequest);
if (HI_SUCCESS != s32Ret)
{
    printf("acquire drm info return %d.\n", s32Ret);
    goto fail;
}

stCertsBuffer = HI_DRM_INFO_GetData(hInfo);
pstOpenParam = (HI_PLAYREADY_DEVCERT_BUF_S *)stCertsBuffer.data;
if (NULL == pstOpenParam)
{
    s32Ret = HI_FAILURE;
    printf("get drm certs buffer fail.\n");
    goto fail;
}

//Set the address and size of the buffer for storing the PlayReady
certificate.
pstOpenParam->pu8Pri          = ;//Address for storing priv.dat
pstOpenParam->s32PriLen       = ;//Data size of priv.dat
pstOpenParam->pu8Zgpriv        = ;//Address for storing zgpriv.dat
pstOpenParam->s32ZgprivLen     = ;//Data size of zgpriv.dat
pstOpenParam->pu8Bgroupcert   = ;//Address for storing
bgroupcert.dat
pstOpenParam->s32BgroupcertLen = ;//Data size of bgroupcert.dat
pstOpenParam->pu8Devcerttemplate = ;//Address for storing
```



```
devcerttemplate.dat
pstOpenParam->s32DevcerttemplateLen = ;//Data size of devcerttemplate.dat
pstOpenParam->pfnDestructor = NULL;//Set the callback function
for releasing the buffer for storing certificate data.
hInfoStatus = HI_DRM_ProcessDrmInfo(hClient, hInfo);
if (NULL == hInfoStatus || hInfoStatus->statusCode != HI_DRM_STATUS_OK)
{
    s32Ret = HI_FAILURE;
    printf("process drm info fail, status code:%d\n",
hInfoStatus->statusCode);
    goto fail;
}

fail:
if (hRequest != NULL)
{
    HI_DRM_INFO_REQUEST_Destroy(&hRequest);
}
if (hInfoStatus != NULL)
{
    HI_DRM_INFO_STATUS_Destroy(hInfoStatus);
}
if (hInfo != NULL)
{
    HI_DRM_INFO_Destroy(&hInfo);
}
if (hClient != NULL)
{
    HI_DRM_DestroyClient(&hClient);
}
```

## Setting the PlayReady Domain Certificate

To play the PlayReady encrypted file with the domain certificate, you need to join the domain before the playback starts and leave the domain after the playback is complete. The following is an instance of setting **JoinDomain.cms**. The method of setting **LeaveDomain.cms** or metering certificate is similar.

```
HI_S32 s32Ret = HI_SUCCESS;
HI_DRM_CLIENT_PPTR hClient = NULL;
HI_DRM_INFO_REQUEST_PPTR hRequest = NULL;
HI_DRM_INFO_PPTR hInfo = NULL;
HI_DRM_INFO_STATUS_PTR hInfoStatus = NULL;

s32Ret = HI_DRM_CreateClient(&hClient);
if (HI_SUCCESS != s32Ret)
```



```
{  
    printf("create drm client return %d.\n", s32Ret);  
    return s32Ret;  
}  
  
s32Ret = HI_DRM_INFO_REQUEST_Create(&hRequest,  
HI_DRM_PLAYREADY_INITIATOR_INFO, "application/vnd.ms-playready");  
if (HI_SUCCESS != s32Ret)  
{  
    printf("create drm info request return %d.\n", s32Ret);  
    goto fail;  
}  
hInfo = HI_DRM_AcquireDrmInfo(hClient, hRequest);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("acquire drm info return %d.\n", s32Ret);  
    goto fail;  
}  
//Set the key-value.  
s32Ret = HI_DRM_INFO_Put(hInfo, HI_KEY_PDY_InitiatorFile,  
"./JoinDomain.cms");  
if (HI_SUCCESS != s32Ret)  
{  
    printf("put info key-value return %d.\n", s32Ret);  
    goto fail;  
}  
hInfoStatus = HI_DRM_ProcessDrmInfo(hClient, hInfo);  
if (NULL == hInfoStatus || hInfoStatus->statusCode != HI_DRM_STATUS_OK)  
{  
    s32Ret = HI_FAILURE;  
    printf("process drm info fail, status code:%d\n",  
hInfoStatus->statusCode);  
    goto fail;  
}  
  
fail:  
if (hRequest != NULL)  
{  
    HI_DRM_INFO_REQUEST_Destroy(&hRequest);  
}  
if (hInfoStatus != NULL)  
{  
    HI_DRM_INFO_STATUS_Destroy(hInfoStatus);  
}
```



```
if (hInfo != NULL)
{
    HI_DRM_INFO_Destroy(&hInfo);
}

if (hClient != NULL)
{
    HI_DRM_DestroyClient(&hClient);
}
```

## Decrypting the PlayReady Envelop File

You need to set the full path file names of the envelop file and the output file.

```
HI_S32 s32Ret = HI_SUCCESS;
HI_DRM_CLIENT_PPTR hClient = NULL;
HI_DRM_INFO_REQUEST_PPTR hRequest = NULL;
HI_DRM_INFO_PPTR hInfo = NULL;
HI_DRM_INFO_STATUS_PTR hInfoStatus = NULL;

s32Ret = HI_DRM_CreateClient(&hClient);
if (HI_SUCCESS != s32Ret)
{
    printf("create drm client return %d.\n", s32Ret);
    return s32Ret;
}

s32Ret = HI_DRM_INFO_REQUEST_Create(&hRequest,
HI_DRM_PLAYREADY_ENVELOP_INFO, "application/vnd.ms-playready");
if (HI_SUCCESS != s32Ret)
{
    printf("create drm info request return %d.\n", s32Ret);
    goto fail;
}

hInfo = HI_DRM_AcquireDrmInfo(hClient, hRequest);
if (HI_SUCCESS != s32Ret)
{
    printf("acquire drm info return %d.\n", s32Ret);
    goto fail;
}

s32Ret = HI_DRM_INFO_Put(hInfo, HI_KEY_PDY_EnvelopFile, "./bear.pye");
if (HI_SUCCESS != s32Ret)
{
    printf("put info key-value return %d.\n", s32Ret);
    goto fail;
}
```



```
s32Ret = HI_DRM_INFO_Put(hInfo, HI_KEY_PDY_EnvelopOutputFile,
    "./bear.asf");
if (HI_SUCCESS != s32Ret)
{
    printf("put info key-value return %d.\n", s32Ret);
    goto fail;
}
hInfoStatus = HI_DRM_ProcessDrmInfo(hClient, hInfo);
if (NULL == hInfoStatus || hInfoStatus->statusCode != HI_DRM_STATUS_OK)
{
    s32Ret = HI_FAILURE;
    printf("process drm info fail, status code:%d\n",
        hInfoStatus->statusCode);
    goto fail;
}

fail:
if (hRequest != NULL)
{
    HI_DRM_INFO_REQUEST_Destroy(&hRequest);
}
if (hInfoStatus != NULL)
{
    HI_DRM_INFO_STATUS_Destroy(hInfoStatus);
}
if (hInfo != NULL)
{
    HI_DRM_INFO_Destroy(&hInfo);
}
if (hClient != NULL)
{
    HI_DRM_DestroyClient(&hClient);
}
```

## Obtaining DRM Authorization Information

The DRM authorization information can be obtained only after the DRM certificate path is configured, the DRMEngine libraries are registered, and the domain or metering certificate is set (if required).

```
HI_S32 s32Ret = HI_SUCCESS;
HI_HANDLE hPlayer = (HI_HANDLE) NULL;
HI_DRM_CLIENT_PPTR hClient = NULL;
HI_DRM_INFO_REQUEST_PPTR hRequest = NULL;
HI_DRM_INFO_PPTR hInfo = NULL;
```



```
HI_DRM_INFO_STATUS_PTR hInfoStatus = NULL;
HI_SVR_PLAYER_METADATA_S stMetaData;
HI_SVR_PLAYER_MEDIA_S stMedia;
HI_CHAR* pszDrmMimeType = NULL;
HI_DRM_RIGHTS_STATUS_E eStatus;

s32Ret = HI_DRM_CreateClient(&hClient);
if (HI_SUCCESS != s32Ret)
{
    printf("create drm client return %d.\n", s32Ret);
    return s32Ret;
}

s32Ret = HI_SVR_PLAYER_Create(..., &hPlayer)
if (HI_SUCCESS != s32Ret)
{
    printf("create hiplayer return %d.\n", s32Ret);
    goto fail;
}
...
//Set the URL for the media file to be played.
memset(&stMedia, 0, sizeof(stMedia));
snprintf(stMedia.aszUrl, sizeof(stMedia.aszUrl), "%s",
"/mnt/usb/bear.pyv");
s32Ret = HI_SVR_PLAYER_SetMedia(hPlayer, HI_SVR_PLAYER_MEDIA_STREAMFILE,
&stMedia);
if (HI_SUCCESS != s32Ret)
{
    printf("set media return %d.\n", s32Ret);
    goto fail;
}
//Obtain metadata.
memset(&stMetaData, 0, sizeof(HI_SVR_PLAYER_METADATA_S));
s32Ret = HI_SVR_PLAYER_Invoke(hPlayer, HI_FORMAT_INVOKE_GET_METADATA,
&stMetaData);
if (HI_SUCCESS != s32Ret)
{
    printf("get metadata return %d.\n", s32Ret);
    goto fail;
}
for (i = 0; i < stMetaData.u32KvpUsedNum; i++)
{
    if (!strcasecmp(stMetaData.pstKvp[i].pszKey, "DRM_MIME_TYPE"))
    {
```



```
    pszDrmMimeType = (HI_CHAR*) stMetaData.pstKvp[i].unValue.pszValue;
    break;
}
}

//If DRM authorization is required for playing this file
if (pszDrmMimeType)
{
    s32Ret = HI_DRM_INFO_REQUEST_Create(&hRequest,
HI_DRM_RIGHTS_ACQUISITION_INFO, pszDrmMimeType);
    if (HI_SUCCESS != s32Ret)
    {
        printf("create drm info request return %d.\n", s32Ret);
        goto fail;
    }
    hInfo = HI_DRM_AcquireDrmInfo(hClient, hRequest);
    if (HI_SUCCESS != s32Ret)
    {
        printf("acquire drm info return %d.\n", s32Ret);
        goto fail;
    }
    hInfoStatus = HI_DRM_ProcessDrmInfo(hClient, hInfo);
    if (NULL == hInfoStatus || hInfoStatus->statusCode !=
HI_DRM_STATUS_OK)
    {
        s32Ret = HI_FAILURE;
        printf("process drm info fail, status code:%d\n", hInfoStatus-
>statusCode);
        goto fail;
    }
    eStatus = HI_DRM_RIGHTS_NOT_ACQUIRED;
    while(eStatus == HI_DRM_RIGHTS_NOT_ACQUIRED)
    {
        eStatus = HI_DRM_CheckRightsStatus(hClient, stMedia.aszUrl,
HI_ACTION_PLAY);
        usleep(100000);
    }
    if (eStatus != HI_DRM_RIGHTS_VALID)
    {
        s32Ret = HI_FAILURE;
        printf("get rights error:%d\n", eStatus);
        goto fail;
    }
}
```



```
fail:  
    if (hPlayer != (HI_HANDLE) NULL)  
    {  
        HI_SVR_PLAYER_Destroy(hPlayer);  
    }  
    if (hRequest != NULL)  
    {  
        HI_DRM_INFO_REQUEST_Destroy(&hRequest);  
    }  
    if (hInfoStatus != NULL)  
    {  
        HI_DRM_INFO_STATUS_Destroy(hInfoStatus);  
    }  
    if (hInfo != NULL)  
    {  
        HI_DRM_INFO_Destroy(&hInfo);  
    }  
    if (hClient != NULL)  
    {  
        HI_DRM_DestroyClient(&hClient);  
    }
```



# Contents

---

<b>26 Wi-Fi .....</b>	<b>1</b>
26.1 Overview .....	1
26.2 Important Concepts .....	3
26.3 Features .....	3
26.4 Development Guide.....	4
26.4.1 Enabling/Disabling the Wi-Fi STA .....	4
26.4.2 Scanning and Connecting to an AP .....	8
26.4.3 Enabling, Setting, and Disabling SoftAP .....	11
26.4.4 Obtaining the MAC Address of the Local Wi-Fi Device .....	13



# Figures

<b>Figure 26-1</b> Structure of the STA .....	1
<b>Figure 26-2</b> Structure of the SoftAP .....	2
<b>Figure 26-3</b> Process for enabling and disabling the Wi-Fi STA .....	4
<b>Figure 26-4</b> Conversion between STA states .....	5
<b>Figure 26-5</b> Process for connecting to and disconnecting from an AP .....	8
<b>Figure 26-6</b> Typical scanning process .....	9
<b>Figure 26-7</b> Typical connecting process .....	10
<b>Figure 26-8</b> Process for enabling and disabling SoftAP .....	12



## Tables

---

**Table 26-1 Wi-Fi event type .....** ..... 6



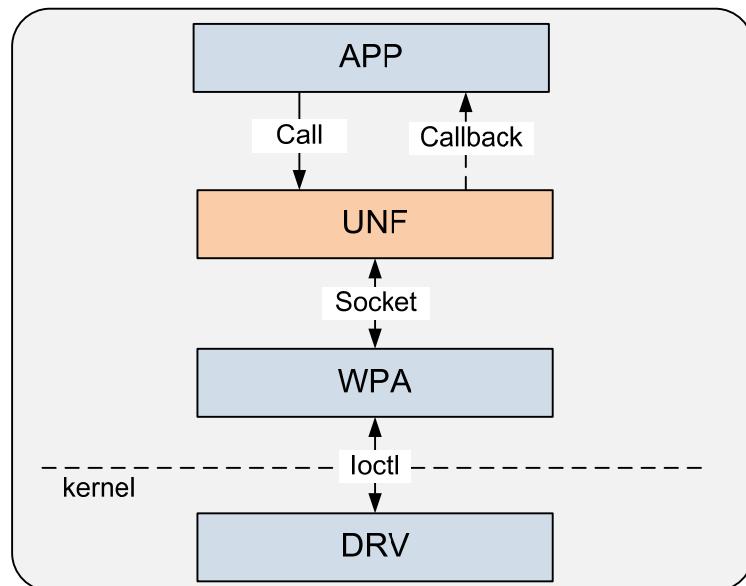
# 26 Wi-Fi

## 26.1 Overview

The Wi-Fi component implements the station (STA) and software enable access point (SoftAP) functions. The STA functions include enabling and disabling Wi-Fi devices, scanning nearby access points (APs), connecting to or disconnecting from APs, and obtaining the media access control (MAC) address for the local Wi-Fi device. The SoftAP functions include enabling and disabling Wi-Fi devices, enabling and disabling SoftAP, setting SoftAP, and obtaining the MAC address for the local Wi-Fi device.

Figure 26-1 shows the structure of the STA.

**Figure 26-1** Structure of the STA



The software architecture consists of the application layer (which is developed by the customer) and the SDK layer (which is provided by HiSilicon).

The SDK layer consists of the following three parts:

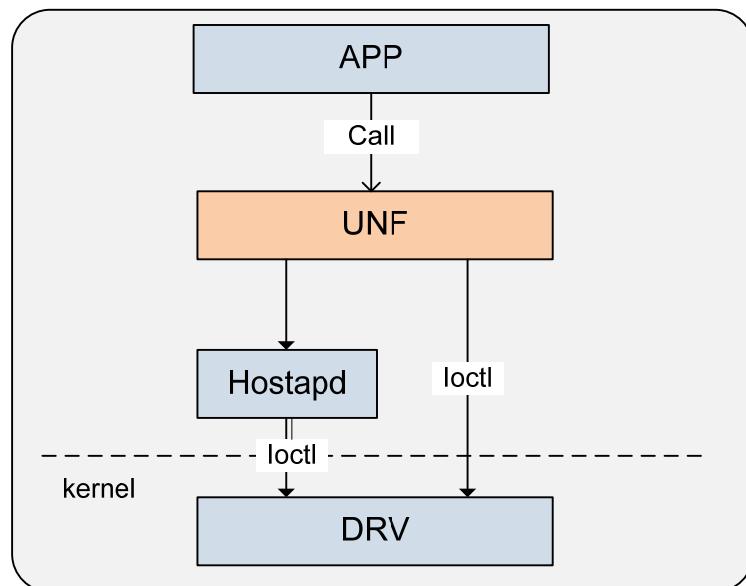
- UNF



- Includes all the application development APIs of the Wi-Fi component.
- Wi-Fi protected access (WPA)
  - Implements the user-mode processes for Wi-Fi scanning and connection processing. The open-source wpa\_supplicant is used.
- DRV
  - Includes the drivers of Wi-Fi devices, which interact with Wi-Fi hardware devices. The drivers run in kernel mode and are provided by device vendors.

Figure 26-2 shows the structure of the SoftAP.

**Figure 26-2** Structure of the SoftAP



The software architecture consists of the application layer (which is developed by the customer) and the SDK layer (which is provided by HiSilicon).

The SDK layer consists of the following three parts:

- UNF
  - Includes all the application development APIs of the Wi-Fi component.
  - The UNF enables the drivers to work in SoftAP mode by using the hostapd or by configuring the drivers using the **Ioctl** command.
- Hostapd
  - The open-source hostapd is used to enable the drivers to work in SoftAP mode and set the SSID, channel, security mode, and whether to hide the SSID.
- DRV
  - Includes the drivers of Wi-Fi devices, which interact with Wi-Fi hardware devices. The drivers run in kernel mode and are provided by device vendors.



## 26.2 Important Concepts

[STA]

The STA connects to the AP as a client to compose the wireless local area network (WLAN).

[AP]

Wi-Fi APs compose the WLAN.

[SoftAP]

The SoftAP enables the Wi-Fi device to work in AP mode based on the Wi-Fi hardware by using software to implement basic AP functions.

## 26.3 Features

The Wi-Fi component provides the following functions:

- Starts and stops the STA. The following APIs are provided:
  - HI\_WLAN\_STA\_Init: Initializes the STA component.
  - HI\_WLAN\_STA\_Open: Enables the Wi-Fi device.
  - HI\_WLAN\_STA\_Start: Enables the STA and registers the event callback function.
  - HI\_WLAN\_STA\_Stop: Disables the STA.
  - HI\_WLAN\_STA\_Close: Disables the Wi-Fi device.
  - HI\_WLAN\_STA\_DeInit: Deinitializes the STA.
- Scans and connects to APs. The following APIs are provided:
  - HI\_WLAN\_STA\_StartScan: Starts scanning.
  - HI\_WLAN\_STA\_GetScanResults: Obtains the scanning result.
  - HI\_WLAN\_STA\_Connect: Connects to an AP.
  - HI\_WLAN\_STA\_GetConnectionStatus: Obtains the current connection status.
  - HI\_WLAN\_STA\_Disconnect: Disconnects from the AP.
- Enables, sets, and disables the SoftAP. The following APIs are provided:
  - HI\_WLAN\_AP\_Init: Initializes the SoftAP.
  - HI\_WLAN\_AP\_Open: Enables the Wi-Fi device.
  - HI\_WLAN\_AP\_Start: Enables the SoftAP.
  - HI\_WLAN\_AP\_Stop: Disables the SoftAP.
  - HI\_WLAN\_AP\_Close: Disables the Wi-Fi device.
  - HI\_WLAN\_AP\_DeInit: Deinitializes the SoftAP.
  - HI\_WLAN\_AP\_SetSoftAP: Sets the SSID, channel, security mode, and whether to hide the SSID for the SoftAP.
- Obtains the MAC address for the local Wi-Fi device. The following API is provided:
  - HI\_WLAN\_STA\_GetMacAddress: Obtains the MAC address for the local Wi-Fi device in STA mode.
  - HI\_WLAN\_AP\_GetMacAddress: Obtains the MAC address for the local Wi-Fi device in SoftAP mode.



## 26.4 Development Guide

### 26.4.1 Enabling/Disabling the Wi-Fi STA

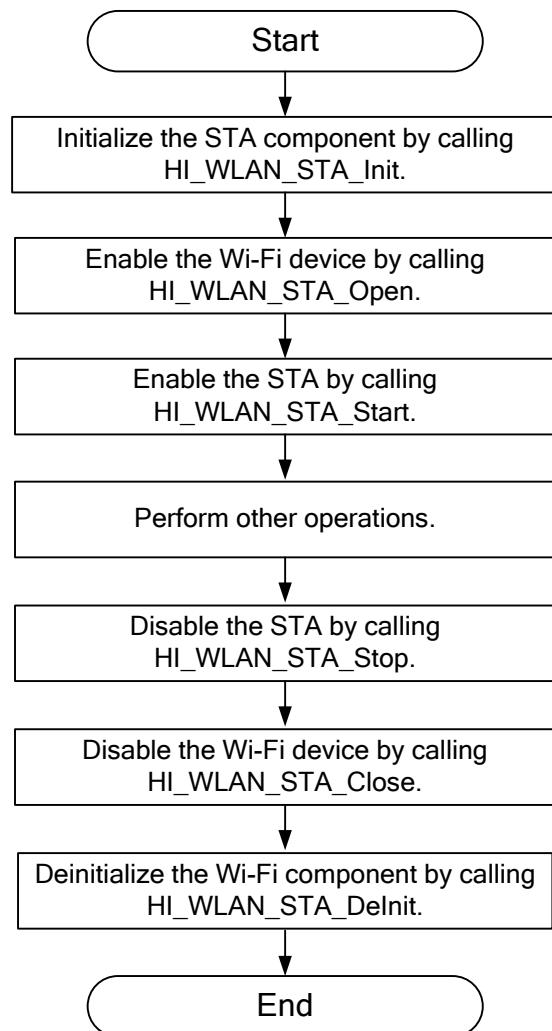
#### Scenario

The Wi-Fi STA needs to be enabled before using the Wi-Fi function and disabled before the Wi-Fi device connects to the AP or after the Wi-Fi function is not required.

#### Working Process

Figure 26-3 shows the process for enabling and disabling the Wi-Fi STA.

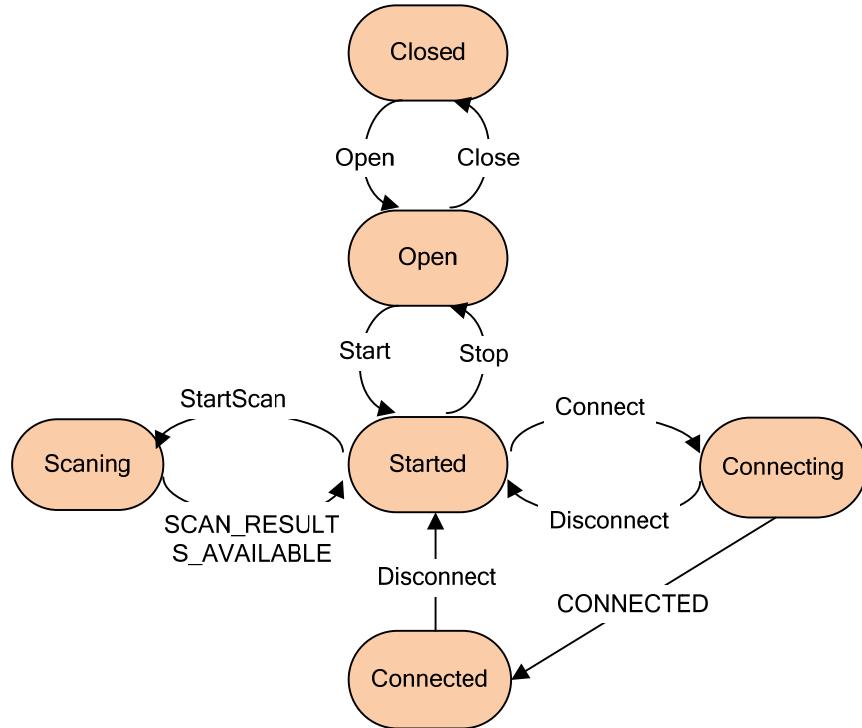
**Figure 26-3** Process for enabling and disabling the Wi-Fi STA



The STA status includes the following: closed, open, started, scanning, connecting, connected, and the initial status is closed. Figure 26-4 shows the conversion between STA states.



**Figure 26-4** Conversion between STA states



To enable and disable a Wi-Fi device, perform the following steps:

**Step 1** Initialize the STA component.

The STA interfaces are required. Therefore, initialize the STA component by calling `HI_WLAN_STA_Init`.

**Step 2** Enable the Wi-Fi device.

After initialization, enable the Wi-Fi device by calling `HI_WLAN_STA_Open`. When the device is enabled, a wireless network device interface is created. The function returns the name of the network interface, which must be correctly transferred when subsequent functions are called. Ensure that the buffer for storing the network interface name is no less than 17 bytes; otherwise, memory overflow occurs.

**NOTE**

For some Wi-Fi devices, `HI_WLAN_STA_Open` takes about one to two seconds, while for most Wi-Fi devices, it takes less than one second.

The initial status of the STA device is closed. When the function returns a success code, the STA device is enabled successfully and the status changes to open.

- If the function returns the error code `HI_WLAN_DEVICE_NOT_FOUND`, no Wi-Fi device is detected, the Wi-Fi device is not powered on, or the Wi-Fi device is not supported.
- If the function returns the error code `HI_WLAN_LOAD_DRIVER_FAIL`, the driver file cannot be found or the driver fails to be loaded. You need to check the driver file in `\kmod`.

Only one Wi-Fi device can be enabled. If there are multiple hardware devices, the device that is first detected is enabled.



**Step 3** Start the STA device.

After the Wi-Fi device is enabled, start the STA device by calling HI\_WLAN\_STA\_Start. The start process is complete after the STA device status is started.

HI\_WLAN\_STA\_Start also registers the event callback function with the UNF layer for receiving STA events. The StartScan, The connect operation is asynchronous. The operation results cannot be obtained immediately after the function returns. They can be obtained by using the returned events. The callback function type is HI\_VOID (\*HI\_WLAN\_STA\_Event\_CallBack)(HI\_WLAN\_STA\_EVENT\_E event, HI\_VOID \*pstPrivData). event indicates the event type, and **pstPrivData** indicates the data attached to the event.

**Table 26-1** Wi-Fi event type

Events	Attached Data	Description
HI_WLAN_STA_EVENT_DISCONNECTED	None	Disconnection event. This event is received when the Disconnect function is called in the connecting or connected status.
HI_WLAN_STA_EVENT_SCAN_RESULTS_AVAILABLE	None	Scanning completion event. This event is sent by the bottom layer when the scanning is complete after the Startscan function is called in the started, connecting, or connected status. Some Wi-Fi device drivers also scan during the connecting process, and the application also receives this event. After this event is received, the scanning result can be obtained by calling GetScanResults.
HI_WLAN_STA_EVENT_CONNECTING	MAC address for the AP, in the form of 00:11:22:33:44:55	Connecting AP event. This event is received when the connect function is successfully called in started status.
HI_WLAN_STA_EVENT_CONNECTED	MAC address for the AP, in the form of 00:11:22:33:44:55	AP connected event. This event is sent to the application when the Wi-Fi device successfully connects to the AP.
HI_WLAN_STA_EVENT_SUPP_STOPPED	None	wpa_supplicant exit event. When this event is received, the STA device must be returned to the closed status.
HI_WLAN_STA_EVENT_DRIVER_STOPPED	None	Driver exception exit event. When this event is received, the STA device must be returned to the closed status.



- If HI\_WLAN\_STA\_Start returns the error code HI\_WLAN\_START\_SUPPLICANT\_FAIL, the wpa\_supplicant process cannot be started. You need to check whether the **wpa\_supplicant** file exists in \sbin, and whether you have the execution permission.
- If HI\_WLAN\_STA\_Start returns the error code HI\_WLAN\_CONNECT\_TO\_SUPPLICANT\_FAIL, the wpa\_supplicant process cannot be connected. You need to call HI\_WLAN\_STA\_Stop and then call HI\_WLAN\_STA\_Start.

**Step 4** Stop the STA device.

Stop the STA device by calling HI\_WLAN\_STA\_Stop. After the operation, the STA device is in the open status and cannot be used. The application does not receive STA events any more. Before stopping the STA device, you are advised to enable the STA device to enter the started status because the stop operation may fail when HI\_WLAN\_STA\_Stop is called in other status.

**Step 5** Disable the Wi-Fi device.

Disable the Wi-Fi device by calling HI\_WLAN\_STA\_Close in the open status. The network interface is also deleted. After that, the STA device is in closed status. HI\_WLAN\_STA\_Close cannot be called when the Wi-Fi device is not in the open status. Otherwise, the device cannot be disabled.

**Step 6** Deinitialize the Wi-Fi component.

Deinitialize the STA by calling HI\_WLAN\_STA\_DeInit when the STA component is not required.

----End

## Notes

- Enable and disable the STA in the same process. Otherwise, other processes cannot use the STA.
- Run the application as the **root** user. Otherwise, some APIs may fail to be called.
- Implement the callback function and the start function in different processes. If they share some resources, use the synchronization mechanism to avoid resource competition. Avoid complex processing in the callback function because subsequent events cannot be received in time if the processing time is too long. It is recommended that events be transmitted to other processes. Do not call any API in the callback function to avoid dead lock.
- The STA status need to be stored by the application. No function can be used to obtain the current status of the bottom layer STA.
- The STA API header file is **hi\_wlan\_sta.h**. The STA component has one library **libwlansta**, which is provided as the dynamic library and static library files. The application needs to link to the two library files during compilation. **libwlansta.a** is used for static links, and **libwlansta.so** is used for dynamic links.
- Disable the STA and Wi-Fi device when the events HI\_WLAN\_STA\_EVENT\_SUPP\_STOPPED and HI\_WLAN\_STA\_EVENT\_DRIVER\_STOPPED are received.

## Sample

See **sample\wifi\sample\_stac.c**.



## 26.4.2 Scanning and Connecting to an AP

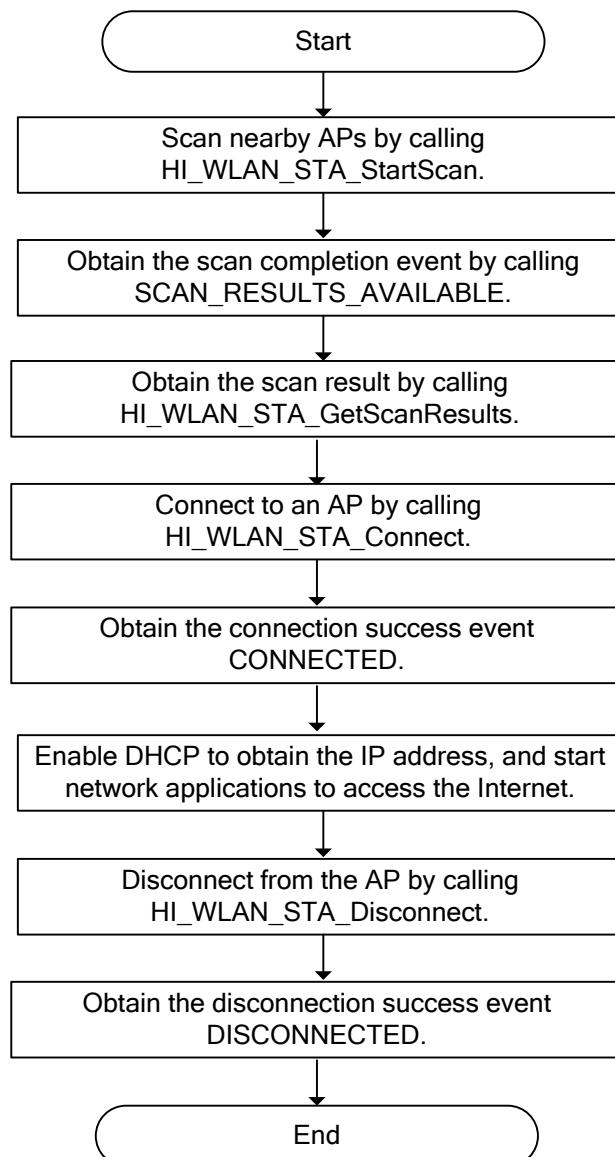
### Scenario

APs are scanned and connected when the user needs to connect to the Internet by using Wi-Fi. The AP connection can also be disconnected.

### Working Process

Figure 26-5 shows the process for connecting to and disconnecting from an AP.

**Figure 26-5** Process for connecting to and disconnecting from an AP



Perform the following steps:

**Step 1** Scan nearby APs.

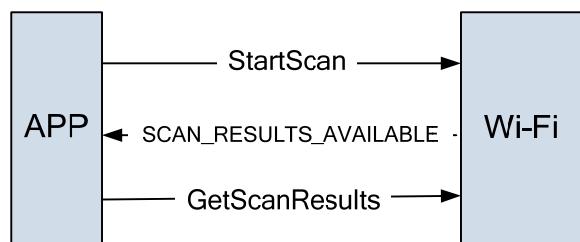


Scan nearby APs by calling `HI_WLAN_STA_StartScan` when the Wi-Fi device is in the started or connected status. You are advised not to scan APs in the connecting status because the connection may be slow or fail.

`HI_WLAN_STA_StartScan` does not return the scanning result. The bottom layer sends the `HI_WLAN_STA_EVENT_SCAN_RESULTS_AVAILABLE` event to the application after the scanning is complete. The application can obtain the scanning result by calling `HI_WLAN_STA_GetScanResults` after receiving the event. The application needs to define an array `HI_WLAN_STA_ACCESSPOINT_S` to store scanned APs, and the recommended array size is 32.

When `HI_WLAN_STA_GetScanResults` is called, the parameter `pstApNum` needs to be set to the size of the array (member number). If `pstApNum` is greater than the array, array overrun may occur. The function returns all scanned APs and sets `pstApNum` to the number of scanned APs.

**Figure 26-6** Typical scanning process



If the AP list needs to be updated in real time, call `HI_WLAN_STA_StartScan` periodically and obtain the scanning results based on the preceding process. The calling frequency depends on the required update period. You are advised to perform scanning every 10 seconds.

If the function returns the error code `HI_WLAN_SEND_COMMAND_FAIL`, the command cannot be sent to the `wpa_supplicant` process. If this error code is returned for three consecutive times, you need to switch the Wi-Fi device to the open status and then start it again. The processing for other functions is similar.

## Step 2 Connect to an AP.

Connect to an AP by calling `HI_WLAN_STA_Connect` when the Wi-Fi device is in the started or connected status. This operation is not recommended when the Wi-Fi device is in the scanning or connecting status because it affects the scanning or connecting process. If the Wi-Fi device is in the connecting status, call `HI_WLAN_STA_Disconnect` first. You can connect to an AP by calling `HI_WLAN_STA_Connect` after the `HI_WLAN_STA_EVENT_DISCONNECTED` event is received.

The connecting process is asynchronous. `HI_WLAN_STA_Connect` returns a value immediately. You can obtain the connection status by using the event. There are three connection states:

- Disconnected: `HI_WLAN_STA_EVENT_DISCONNECTED`
- Connecting: `HI_WLAN_STA_EVENT_CONNECTING`
- Connected: `HI_WLAN_STA_EVENT_CONNECTED`

The connection status changes in the following three scenarios:

- An AP is successfully connected.



After a connection is initiated, the connecting event and then the connected event are received. Each event is attached with the MAC address for the AP.

- An AP fails to be connected.

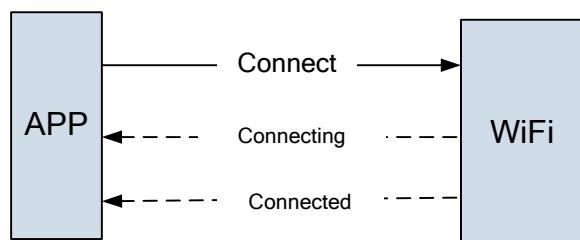
After a connection is initiated, the connecting event and then the disconnected event are received (some Wi-Fi devices may not receive the disconnected event in the security mode). The bottom layer attempts to connect again instead of ending the connection. Then the application receives the connecting event and then the disconnected event again. The process is repeated unless the application terminates the connection process by calling HI\_WLAN\_STA\_Disconnect.

Wi-Fi signals are susceptible to interference and cannot be connected sometimes. Therefore, the application needs to connect for several more times after the first attempt fails. It is recommended that the connection be terminated after three consecutive connecting failures.

- No AP can be detected.

There is no AP nearby to be connected. After a connection is initiated, the connecting event is not received, but the disconnected event is received periodically. The application determines whether to terminate the connection based on the number of received disconnected events. It is recommended that the connection be terminated by calling HI\_WLAN\_STA\_Disconnect after three disconnected events are received.

**Figure 26-7** Typical connecting process



An AP that is not in the AP list of the application, or an AP whose service set identifier (SSID) is hidden, can also be connected. If an AP with hidden SSID is to be connected, set **hidden\_ssid** of the **config** parameter in HI\_WLAN\_STA\_Connect to **HI\_TRUE**. If the SSID of the AP to be connected is not hidden, set **hidden\_ssid** to **HI\_FALSE**. Otherwise, the connection is slow.

**Step 3** Obtain the current connection status.

Obtain the current connection information by calling HI\_WLAN\_STA\_GetConnectionStatus. The information includes the connection status, and the SSID, MAC address, and safety mode of the connected AP or the AP being connected.

**Step 4** Disconnect from the AP.

The HI\_WLAN\_STA\_Disconnect function has the following two functions:

- Ends the connecting process in connecting status.
- Disconnects from the AP in connected status. The operation is asynchronous. The connection is disconnected after the DISCONNECTED event is received.

----End



## Notes

- The Wi-Fi device sometimes cannot detect some APs during scanning, but it may detect the APs when it scans again. Therefore, never delete the APs that cannot be scanned immediately from the AP list. It is recommended that APs that cannot be scanned for three consecutive times be deleted from the list.
- Drivers of some Wi-Fi devices scan periodically during the connecting process, and the application receives the HI\_WLAN\_STA\_EVENT\_SCAN\_RESULTS\_AVAILABLE event periodically and determines whether to obtain the scanning result. It is recommended that the scanning result not be obtained if HI\_WLAN\_STA\_StartScan is not called.

## Sample

See **sample\wifi\sample\_stac.c**.

Open source codes are used in **sample\wifi\sta\_ui.c**. You are advised not to use such codes during development in case of open source risk.

## 26.4.3 Enabling, Setting, and Disabling SoftAP

### Scenario

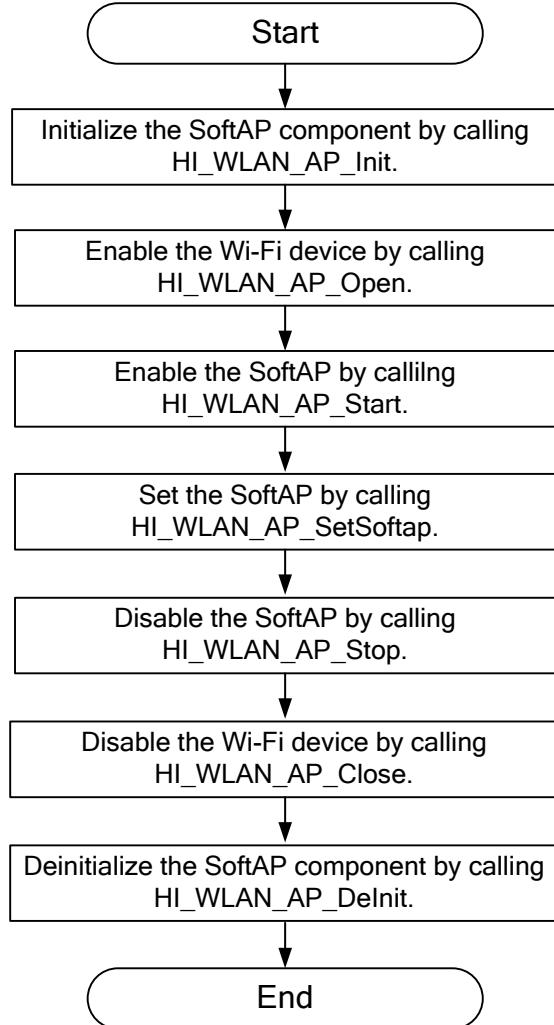
The SoftAP mode needs to be enabled to allow the mobile phone or other STA devices to access the Wi-Fi device when network sharing is implemented by using the Ethernet uplink and Wi-Fi downlink.

### Working Process

[Figure 26-8](#) shows the process for enabling and disabling SoftAP.



**Figure 26-8** Process for enabling and disabling SoftAP



- Initialize the SoftAP component.

The SoftAP interfaces are required. Therefore, initialize the SoftAP component by calling `HI_WLAN_AP_Init`.

- Enable the Wi-Fi device.

After initialization, enable the Wi-Fi device by calling `HI_WLAN_AP_Open`. When the device is enabled, a wireless network device interface is created. The function returns the name of the network interface, which must be correctly transferred when subsequent functions are called. Ensure that the buffer for storing the network interface name is no less than 17 bytes; otherwise, memory overflow occurs.

For some Wi-Fi devices, `HI_WLAN_AP_Open` takes about one to two seconds, while for most Wi-Fi devices, it takes less than one second.

If the function returns the error code `HI_WLAN_DEVICE_NOT_FOUND`, no Wi-Fi device is detected, the Wi-Fi device is not powered on, or the Wi-Fi device is not supported. If the function returns the error code `HI_WLAN_LOAD_DRIVER_FAIL`, the driver file cannot be found or the driver fails to be loaded. You need to check whether there is the driver file in `/kmod`.



Only one Wi-Fi device can be enabled. If there are multiple hardware devices, the device that is first detected is enabled.

- Enable the SoftAP.

After the Wi-Fi device is enabled, enable the SoftAP by calling `HI_WLAN_AP_Start`. You need to configure the SoftAP when calling `HI_WLAN_AP_Start`.

After the SoftAP is enabled, the STA device can scan this AP. The application can start the DHCP server, enable the DNS agent, and configure IP forwarding to forward IP packets.

- Set the SoftAP.

After the SoftAP is enabled, you can call `HI_WLAN_AP_SetSoftap` to reconfigure the SoftAP. During the configuration process, the SoftAP restarts, and the connected STA device is disconnected.

- Disable the SoftAP.

`HI_WLAN_AP_Stop` disables the SoftAP. After the SoftAP is disabled, the STA device cannot scan the AP, and the connected STA device is disconnected.

- Disable the Wi-Fi device.

Disable the Wi-Fi device by calling `HI_WLAN_AP_Close` after the SoftAP is disabled. The network interface is also deleted. You are advised not to call `HI_WLAN_STA_Close` when the SoftAP is not disabled. Otherwise, the Wi-Fi device cannot be disabled.

- Deinitialize the SoftAP component.

Deinitialize the SoftAP component by calling `HI_WLAN_AP_DeInit` when the SoftAP component is not required or before the process exits.

## Notes

- Enable and disable the SoftAP in the same process. Otherwise, other processes cannot use the SoftAP.
- Run the application as the **root** user. Otherwise, some APIs may fail to be called.
- The connected STA devices can be obtained by using the DHCP server.
- The SoftAP API header file is `hi_wlan_ap.h`. The SoftAP component has two library files `libwlanap.a` and `libwlanap.so`, which are provided as the dynamic and static libraries. The application needs to be linked to the two library files during compilation. `libwlanap.a` is used for static links, and `libwlanap.so` is used for dynamic links.
- If the Wi-Fi device is in STA mode, you need to disable the STA before enabling the SoftAP. If the Wi-Fi device is in SoftAP mode, you need to disable the SoftAP before enabling the STA.

## Sample

See `sample/wifi/sample_ap.c`.

### 26.4.4 Obtaining the MAC Address of the Local Wi-Fi Device

#### Scenario

The MAC address for the local Wi-Fi device can be obtained and displayed in STA mode and SoftAP mode. This section takes the STA mode as an example. The operations in SoftAP mode are the same.



## Working Process

The MAC address for the local Wi-Fi device can be obtained by calling HI\_WLAN\_STA\_GetMacAddress. This API cannot be used when the device status is closed.

## Notes

The MAC address returned by the function is a character string such as 00:11:22:33:44:55. Ensure that the input buffer is no less than 17 bytes. Otherwise, memory overflow occurs.

## Sample

See `sample\wifi\sample_stac.c`.



# Contents

---

<b>27 Startup Screen.....</b>	<b>1</b>
27.1 Overview .....	1
27.2 Important Concepts .....	1
27.3 Features .....	2
27.4 Development Guide.....	2
27.4.1 Displaying the Startup Screen.....	2
27.4.2 Smoothly Switching from the Startup Screen to User Applications .....	3



## Figures

**Figure 27-1** Startup screen in the system..... 1

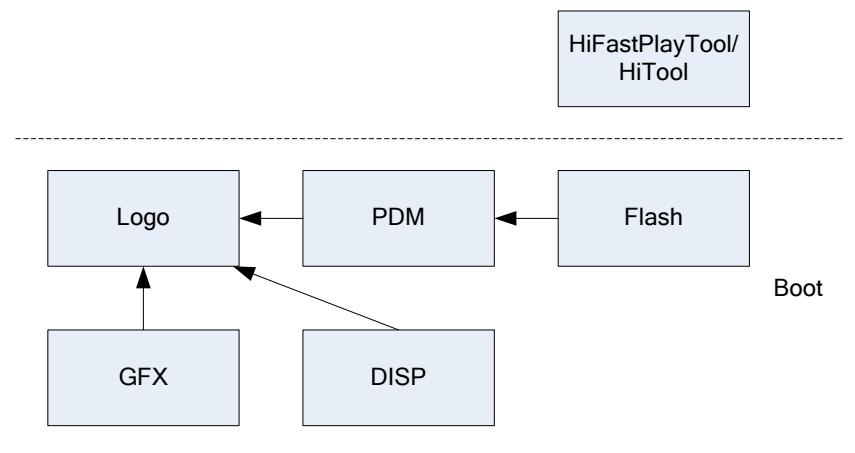


# 27 Startup Screen

## 27.1 Overview

The startup screen feature allows the picture to be quickly displayed during startup and then smoothly switched to user applications. [Figure 27-1](#) shows the startup screen in the system.

**Figure 27-1** Startup screen in the system



The basic parameter image and startup logo image are created by using the HiFastPlay integrated in the HiTool and then burnt to the flash memory by using the HiBurn integrated in the HiTool. The startup screen module obtains the basic parameter and startup picture data from the product data manager (PDM) module and then implements display of the startup picture by calling the GFX and DISP drivers.

## 27.2 Important Concepts

[Smooth switchover from the startup screen to user applications]

Smooth switchover from the startup screen to user applications indicates that if a startup picture is burnt, during the switchover from the startup screen to the GUI or video of user applications, no exceptions such as black screen, screen flickers, or artifacts, occur.



#### [HiFastPlay]

The HiFastPlay is used to create the basic parameter image and startup screen image. It is stored in **\$SDK\_DIR\tools\windows\HiTool**. Start HiTool and then choose HiFastplay.

#### [HiBurn]

The HiTool is used to burn images to the flash memory. It is stored in **\$SDK\_DIR\tools\windows\HiTool**. Start HiTool and then choose HiBurn.

#### [Logo image]

The logo image indicates the startup screen image created by using the HiFastPlay. It contains picture information and picture data.

#### [Basic parameter image]

The basic parameter image is created by using the HiFastPlay. It contains the basic display parameters.

## 27.3 Features

The startup screen module provides the following functions:

- Creates the startup screen image.
- Displays the startup screen.
- Smoothly switches from the startup screen to user applications. The following API is provided:  
`HI_UNF_MCE_ClearLogo`: Clears the startup screen.

## 27.4 Development Guide

The startup screen service is used in the following scenarios:

- Displaying the startup screen
- Smoothly switching from the startup screen to user applications

### 27.4.1 Displaying the Startup Screen

#### Scenario

The startup screen can be displayed by using the HiFastPlay and HiTool provided in the SDK.

#### Working Process

To display the startup screen, perform the following steps:

- Step 1** Create the basic parameter image and startup screen image by using the HiFastPlay. For details about the usage of the HiFastPlay, see the *HiFastPlay User Guide*.
- Step 2** Burn the images by using the HiBurn. For details about the usage of the HiTool, see **HiTool User Manual (Quick Start Video)**.



**Step 3** Properly configure the **bootargs** parameters. Ensure that the baseparam and logo partitions are configured in **bootargs**. Then start the board. The startup screen is displayed.

----End

## Notes

The startup screen image can occupy a separate partition or share a partition with other images. The configuration of **bootargs** is described as follows:

- The startup screen image occupies a separate partition.

```
setenv bootargs 'mem=48M console=ttyAMA0,115200 root=/dev/mtdblock6
rootfstype=squashfs
mtdparts=hi_sfc:192K(boot),64K(bootargs),64K(baseparam),128K(logos),20
48K(kernel),-(rootfs) mmz=ddr,0,0x83000000,80M'
```

- The startup screen image shares a partition with other images.

```
setenv bootargs 'mem=48M console=ttyAMA0,115200 root=/dev/mtdblock4
rootfstype=squashfs
mtdparts=hi_sfc:192K(boot),64K(bootargs),192K(mce),2048K(kernel),-
(rootfs) baseparam=mce,0x0,0x2000 logo=mce,0x2000,0x20000
mmz=ddr,0,0x83000000,80M'
```

In the preceding configuration, **baseparam=mce,0x0,0x2000** indicates that the basic parameter image is stored in the mce partition, **0x0** is the offset address of baseparam data in the mce partition, and **0x2000** is the length of baseparam data.

## 27.4.2 Smoothly Switching from the Startup Screen to User Applications

### Scenario

The smooth switchover mechanism ensures that no exceptions, such as black screen, screen flickers, or artifacts, occur during the switchover from the startup screen to user applications.

### Working Process

The following two switchover scenarios are involved:

- Smooth switchover from the startup screen to the application GUI.

No user operation is required. The SDK automatically implements smooth switchover from the startup screen to the application GUI when refreshing data at the graphics layer.

- Smooth switchover from the startup screen to the application video.

The user needs to clear the startup screen resources by calling `HI_UNF_MCE_ClearLogo` after the video is played.

## Notes

None



## Samples

See `sample\mce\sample_mce_trans.c`.



# Contents

---

<b>28 Fastplay .....</b>	<b>1</b>
28.1 Overview .....	1
28.2 Important Concepts .....	2
28.3 Features .....	2
28.4 Development Guide.....	2
28.4.2 Playing a DVB Program Quickly After Startup .....	5
28.4.3 Playing a Local TS File Quickly After Startup .....	6
28.4.4 Playing an Animation File Quickly After Startup .....	6
28.4.5 Controlling the Fastplay Duration and Times .....	7
28.4.6 Smoothly Switching from Fastplay to User Applications .....	8



## Figures

<b>Figure 28-1</b> Position of the fastplay module in the system.....	1
<b>Figure 28-2</b> Menuconfig_Kernel .....	3
<b>Figure 28-3</b> Menuconfig_BuildIn.....	3
<b>Figure 28-4</b> Menuconfig_MSP.....	4
<b>Figure 28-5</b> Menuconfig_MCE .....	4

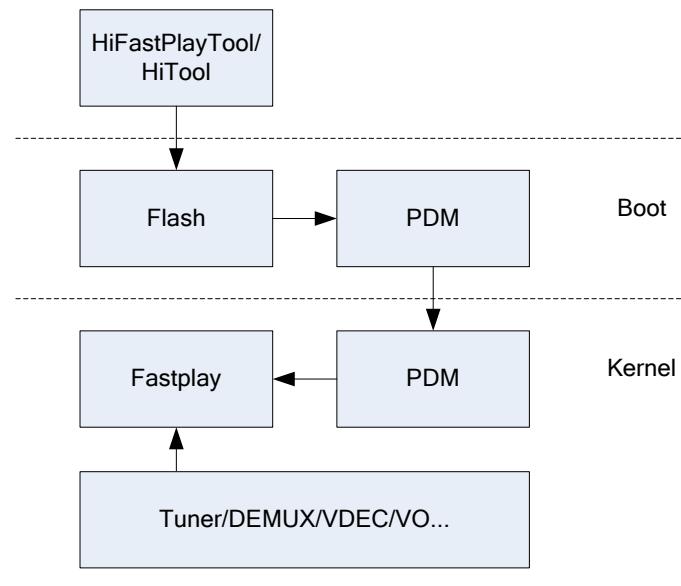


# 28 Fastplay

## 28.1 Overview

The fastplay service allows you to play audio, video, or animation files after the kernel has started while no application is running. It improves the visual and audio experience of users during the startup. It can be used to provide advertisement services. The fastplay service enables a DVB program, an animation file, or a local unscrambled TS file to be played quickly after startup and provides the mechanism that ensure smooth switchover from fastplay to application GUI or video. [0](#) shows the position of the fastplay module in the system.

**Figure 28-1** Position of the fastplay module in the system



The user creates the basic parameter image and fastplay image by using the HiFastPlay and burn the images to the flash memory by using the HiTool. The PDM module reads the basic parameters and fastplay data to the memory under boot, and transmits the read data to the kernel. The fastplay module obtains the parameters and data from the PDM module under the kernel and calls drivers of modules such as DEMUX, VDEC, and VO to implement fastplay.



## 28.2 Important Concepts

[DVB fastplay]

The system quickly plays the DVB program with a specific PID at a specific frequency after startup.

[Animation fastplay]

The system plays an animation file stored in the flash memory after startup.

[TS fastplay]

The system plays a TS file stored in the flash memory quickly after startup.

[Smooth switchover from fastplay to user applications]

Smooth switchover from fastplay to user applications indicates that if the fastplay service exists, during the switchover from fastplay to the GUI or video of user applications, no exceptions, such as black screen, screen flickers, or artifacts, occur.

[Fastplay image]

The fastplay image indicates the TS, animation, or DVB fastplay image created by using the HiFastPlay integrated in the HiTool. The TS fastplay image contains TS parameters and TS data, the animation fastplay image contains picture display parameters and picture data, and the DVB fastplay image contains the DVB program parameters.

## 28.3 Features

The fastplay module has the following functions:

- Creates fastplay images.
- Plays fastplay programs.
- Smoothly switches from fastplay to user applications. The following APIs are provided:
  - HI\_UNF\_MCE\_Stop: Stops fastplay.
  - HI\_UNF\_MCE\_Exit: Exits fastplay.

## 28.4 Development Guide

By default, the SDK does not support fastplay. To enable the fastplay function, perform the following steps:

- Step 1** Access the SDK root directory and run the **make menuconfig** command. The screen shown in [Step 1](#) is displayed.



Figure 28-2 Menuconfig\_Kernel



**Step 2** Select **Kernel**. The screen shown in [Step 2](#) is displayed. Select **Build MSP in Kernel** and return to the upper-level screen shown in [Figure 28-2](#).

Figure 28-3 Menuconfig\_BuildIn



**Step 3** Select **MSP**, as shown in [Step 3](#). The screen shown in [Figure 28-5](#) is displayed. Select **MCE Support**, save the settings, and exit.



Figure 28-4 Menuconfig\_MSP

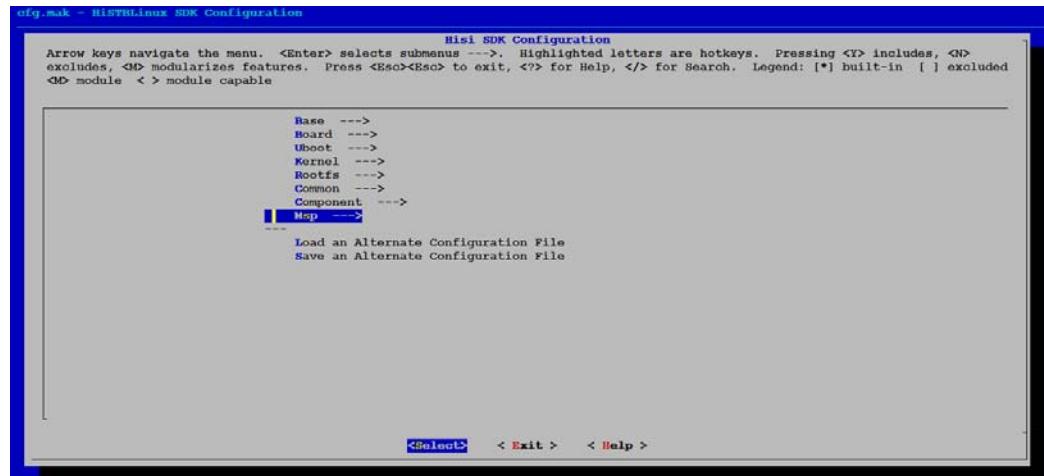
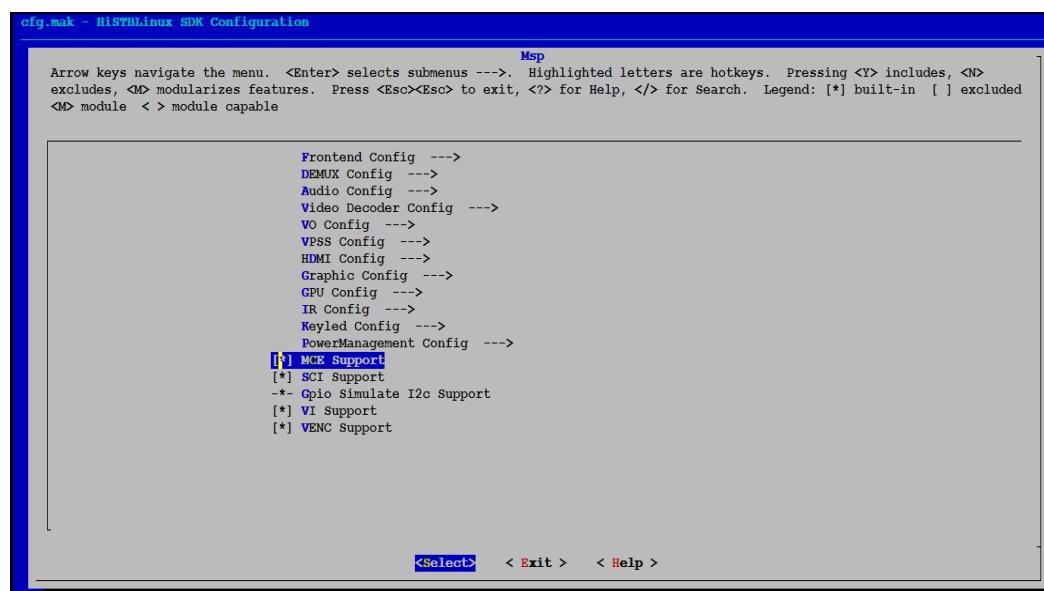


Figure 28-5 Menuconfig\_MCE



**Step 4** Run the **make build** command. The compiled fastboot, kernel, and rootfs images are burnt to the corresponding partitions on the flash.

----End

The fastplay feature is used in the following scenarios:

- Playing a DVB program quickly after startup
- Playing a local TS file quickly after startup
- Playing an animation file quickly after startup
- Controlling the fastplay duration and times
- Smoothly switching from fastplay to user applications



## 28.4.2 Playing a DVB Program Quickly After Startup

### Scenario

The system plays the unscrambled DVB program at a specific frequency quickly after startup.

### Working Process

To play a DVB program quickly after startup, perform the following steps:

- Step 1** Create the basic parameter image and fastplay image by using the HiFastPlay. For details about the usage of the HiFastPlay, see the *HiFastPlay User Guide*.
- Step 2** Burn the images by using the HiBurn. For details about the usage of the HiTool, see **HiTool User Manual (Quick Start Video)**.
- Step 3** Properly configure the **bootargs** parameters. Ensure that the baseparam and fastplay partitions are configured in **bootargs**. Then start the board. The DVB fastplay is implemented.

----End

### Notes

The DVB fastplay relies on the configuration of the tuner on the board. To configure the tuner, perform the following steps:

- Step 1** Run **make menuconfig** in the SDK root directory to open the SDK configuration interface.
- Step 2** Click **Board**.
- Step 3** Click **Tuner Config**.
- Step 4** Click **First Tuner Config**, and set the attributes of the tuner.

----End

The fastplay image can occupy a separate partition or share a partition with other images. The configuration of **bootargs** is described as follows:

- The fastplay image occupies a separate partition.

```
setenv bootargs 'mem=48M console=ttyAMA0,115200 root=/dev/mtdblock6
rootfstype=squashfs
mtdparts=hi_sfc:192K(boot),64K(bootargs),64K(baseparam),128K(logo),64
K(fastplay),2048K(kernel),-(rootfs) mmz=ddr,0,0x83000000,80M'
```
- The fastplay image shares a partition with other images.

```
setenv bootargs 'mem=48M console=ttyAMA0,115200 root=/dev/mtdblock4
rootfstype=squashfs
mtdparts=hi_sfc:192K(boot),64K(bootargs),192K(mce),2048K(kernel),-
(rootfs) baseparam=mce,0x0,0x2000 logo=mce,0x2000,0x20000
fastplay=mce,0x22000,0x10000 mmz=ddr,0,0x83000000,80M'
```



In the preceding configuration, **baseparam=mce,0x0,0x2000** indicates that the basic parameter image is stored in the mce partition, **0x0** is the offset address of baseparam data in the mce partition, and **0x2000** is the length of baseparam data.

## 28.4.3 Playing a Local TS File Quickly After Startup

### Scenario

The system plays a TS file stored in the flash memory quickly after startup.

### Working Process

To play a local TS file quickly after startup, perform the following steps:

- Step 1** Create the basic parameter image and local TS image by using the HiFastPlay. For details about the usage of the HiFastPlay, see the *HiFastPlay User Guide*.
- Step 2** Burn the images by using the HiBurn. For details about the usage of the HiTool, see **HiTool User Manual (Quick Start Video)**.
- Step 3** Properly configure the **bootargs** parameters. Ensure that the baseparam and fastplay partitions are configured in **bootargs**. Then start the board. The local TS fastplay is implemented.

----End

### Notes

The fastplay image can occupy a separate partition or share a partition with other images. The configuration of **bootargs** is described as follows:

- The fastplay image occupies a separate partition.

```
setenv bootargs 'mem=48M console=ttyAMA0,115200 root=/dev/mtblock6
rootfstype=squashfs
mtdparts=hi_sfc:192K(boot),64K(bootargs),64K(baseparam),128K(logo),64
K(fastplay),2048K(kernel),-(rootfs) mmz=ddr,0,0x83000000,80M'
```
- The fastplay image shares a partition with other images.

```
setenv bootargs 'mem=48M console=ttyAMA0,115200 root=/dev/mtblock4
rootfstype=squashfs
mtdparts=hi_sfc:192K(boot),64K(bootargs),192K(mce),2048K(kernel),-
(rootfs) baseparam=mce,0x0,0x2000 logo=mce,0x2000,0x20000
fastplay=mce,0x22000,0x10000 mmz=ddr,0,0x83000000,80M'
```

In the preceding configuration, **baseparam=mce,0x0,0x2000** indicates that the basic parameter image is stored in the mce partition, **0x0** is the offset address of baseparam data in the mce partition, and **0x2000** is the length of baseparam data.

## 28.4.4 Playing an Animation File Quickly After Startup

### Scenario

The system plays an animation file stored in the flash memory quickly after startup.



## Working Process

To play an animation file quickly after startup, perform the following steps:

- Step 1** Create the basic parameter image and animation image by using the HiFastPlay. For details about the usage of the HiFastPlay, see the *HiFastPlay User Guide*.
- Step 2** Burn the images by using the HiBurn. For details about the usage of the HiTool, see **HiTool User Manual (Quick Start Video)**.
- Step 3** Properly configure the **bootargs** parameters. Ensure that the baseparam and fastplay partitions are configured in **bootargs**. Then start the board. The animation fastplay is implemented.

----End

## Notes

The fastplay image can occupy a separate partition or share a partition with other images. The configuration of **bootargs** is described as follows:

- The fastplay image occupies a separate partition.

```
setenv bootargs 'mem=48M console=ttyAMA0,115200 root=/dev/mtdblock6  
rootfstype=squashfs  
mtdparts=hi_sfc:192K(boot),64K(bootargs),64K(baseparam),128K(logo),64  
K(fastplay),2048K(kernel),-(rootfs) mmz=ddr,0,0x83000000,80M'
```

- The fastplay image shares a partition with other images.

```
setenv bootargs 'mem=48M console=ttyAMA0,115200 root=/dev/mtdblock4  
rootfstype=squashfs  
mtdparts=hi_sfc:192K(boot),64K(bootargs),192K(mce),2048K(kernel),-  
(rootfs) baseparam=mce,0x0,0x2000 logo=mce,0x2000,0x20000  
fastplay=mce,0x22000,0x10000 mmz=ddr,0,0x83000000,80M'
```

In the preceding configuration, **baseparam=mce,0x0,0x2000** indicates that the basic parameter image is stored in the mce partition, **0x0** is the offset address of baseparam data in the mce partition, and **0x2000** is the length of baseparam data.

## 28.4.5 Controlling the Fastplay Duration and Times

### Scenario

The fastplay duration and times are controlled by using the APIs of the MCE module.

## Working Process

- Step 1** Initialize the system by calling `HI_SYS_Init`.
- Step 2** Initialize the MCE module by calling `HI_UNF_MCE_Init`.
- Step 3** Enter the fastplay duration and times and stop fastplay by calling `HI_UNF_MCE_Stop`.
- Step 4** Exit fastplay by calling `HI_UNF_MCE_Exit`.
- Step 5** Deinitialize the MCE module by calling `HI_UNF_MCE_DeInit`.

----End



## Notes

- For DVB fastplay, only the duration can be controlled; for TS fastplay, both the duration and times can be controlled.
- HI\_UNF\_MCE\_Stop is called in block mode and only returns values when the specified fastplay duration or times are reached.

## Samples

See `sample\mce\sample_mce_trans.c`.

## 28.4.6 Smoothly Switching from Fastplay to User Applications

### Scenario

The SDK provides the smooth switchover mechanism which ensures that no exceptions, such as black screen, screen flickers, or artifacts, occur during the switchover from fastplay to user applications.

### Working Process

To implement smooth switchover from fastplay to the application GUI, perform the following steps:

- Step 1** Initialize the system.
- Step 2** Initialize the MCE module by calling `HI_UNF_MCE_Init`.
- Step 3** Initialize the HiGo module, create a layer, and decode pictures.
- Step 4** Refresh the layer and display decoded pictures.
- Step 5** Stop fastplay by calling `HI_UNF_MCE_Stop`.
- Step 6** Exit fastplay and release resources by calling `HI_UNF_MCE_Exit`.
- Step 7** Deinitialize the MCE module by calling `HI_UNF_MCE_DeInit`.
- Step 8** Perform other service operations.

**----End**

To implement smooth switchover from fastplay to the application video, perform the following steps:

- Step 1** Initialize the system.
- Step 2** Initialize the MCE module by calling `HI_UNF_MCE_Init`.
- Step 3** Stop fastplay by calling `HI_UNF_MCE_Stop`.
- Step 4** Initialize the devices such as the VO, AVPLAY, and display devices.
- Step 5** Create an AVPLAY and enable the video channel.
- Step 6** Create a window, bind the window to the AVPLAY, and enable the window.
- Step 7** Exit fastplay, transfer the window handle to the MCE module, and release fastplay resources by calling `HI_UNF_MCE_Exit`.



**Step 8** Deinitialize the MCE module by calling HI\_UNF\_MCE\_DeInit.

**Step 9** Perform other service operations.

----End

## Notes

During smooth switchover from fastplay to the application video, you need to transfer the user-mode window handle to the MCE module by calling HI\_UNF\_MCE\_Exit after a window is created and enabled so that data is smoothly switched.

## Samples

See `sample\mce\sample_mce_trans.c`.



# Contents

---

<b>29 PDM.....</b>	<b>1</b>
29.1 Overview .....	1
29.2 Important Concepts .....	2
29.3 Features .....	2
29.4 Development Guide.....	2
29.4.1 Querying and Updating Basic Parameters .....	2
29.4.2 Querying and Updating Startup Screen Parameters and Data.....	3
29.4.3 Querying and Updating Fastplay Parameters and Data.....	3



# Figures

**Figure 29-1** Position of the PDM module in the system..... 1

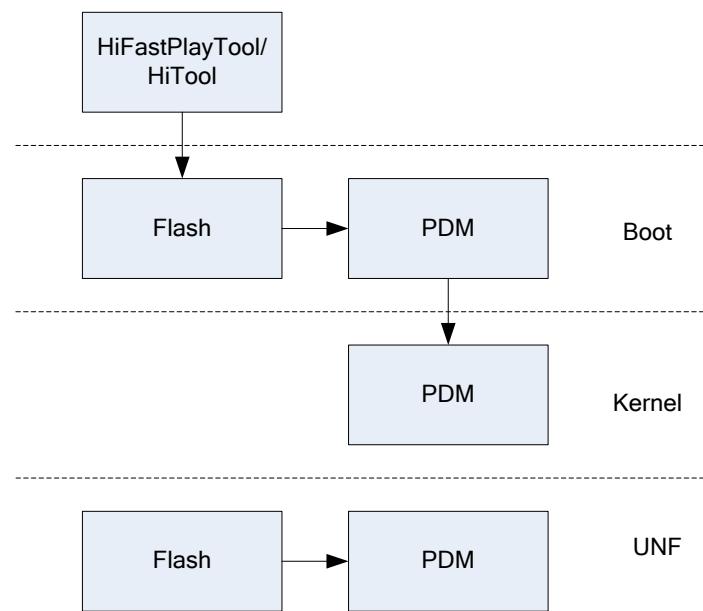


# 29 PDM

## 29.1 Overview

The PDM module manages product-related data, including basic parameters, startup screen parameters and data, and fastplay parameters and data. [Figure 29-1](#) shows the position of the PDM module in the system.

**Figure 29-1** Position of the PDM module in the system



In boot mode, the PDM module reads and parses the parameters and data burnt in the flash memory, provides the parsed data to modules (such as the startup screen), and transfers the address of the data in the memory to the kernel.

In kernel mode, the PDM module parses the parameters and data based on the data transferred under boot and provides the parsed data to modules such as the fastplay, display, and sound modules.

In user mode, the PDM module queries and updates the basic parameters, startup screen data, and fastplay data in the flash memory by calling the APIs of the flash module.



## 29.2 Important Concepts

[Basic parameter image]

The basic parameter image is created by using the HiFastPlay integrated in the HiTool. It contains the basic display parameters.

## 29.3 Features

The PDM module provides the following functions:

- Queries and updates basic parameters. The following APIs are provided:
  - HI\_UNF\_PDM\_GetBaseParam: Obtains basic parameters.
  - HI\_UNF\_PDM\_UpdateBaseParam: Updates basic parameters.
- Queries and updates startup screen parameters and data. The following APIs are provided:
  - HI\_UNF\_PDM\_GetLogoParam: Obtains logo parameters.
  - HI\_UNF\_PDM\_UpdateLogoParam: Updates logo parameters.
  - HI\_UNF\_PDM\_GetLogoContent: Obtains logo data.
  - HI\_UNF\_PDM\_UpdateLogoContent: Updates logo data.
- Queries and updates fastplay parameters and data. The following APIs are provided:
  - HI\_UNF\_PDM\_GetPlayParam: Obtains fastplay parameters.
  - HI\_UNF\_PDM\_UpdatePlayParam: Updates fastplay parameters.
  - HI\_UNF\_PDM\_GetPlayContent: Obtains fastplay data.
  - HI\_UNF\_PDM\_UpdatePlayContent: Updates fastplay data.

## 29.4 Development Guide

The PDM module is used in the following scenarios:

- Querying and updating basic parameters
- Querying and updating startup screen parameters and data
- Querying and updating fastplay parameters and data

### 29.4.1 Querying and Updating Basic Parameters

#### Scenario

The basic parameters stored in the flash memory can be queried and updated by using the APIs of the PDM module.

#### Working Process

To query and update basic parameters, perform the following steps:

**Step 1** Initialize the system by calling HI\_SYS\_Init.



**Step 2** Obtain the corresponding basic parameters by calling HI\_UNF\_PDM\_GetBaseParam.

**Step 3** Update basic parameters by calling HI\_UNF\_PDM\_UpdateBaseParam.

**Step 4** Deinitialize the system by calling HI\_SYS\_DeInit.

----End

## Samples

See sample\mce\sample\_mce\_update.c.

### 29.4.2 Querying and Updating Startup Screen Parameters and Data

#### Scenario

The startup screen contents can be queried and updated by using the APIs of the PDM module.

#### Working Process

To query and update startup screen parameters and data, perform the following steps:

**Step 1** Initialize the system by calling HI\_SYS\_Init.

**Step 2** Obtain the corresponding startup screen parameters by calling HI\_UNF\_PDM\_GeLogoParam.

**Step 3** Read a new startup picture.

**Step 4** Update the length of the new startup picture to the logo parameter area by calling HI\_UNF\_PDM\_UpdateLogoParam.

**Step 5** Update the content of the new startup picture to the logo data area by calling HI\_UNF\_PDM\_UpdateLogoContent.

**Step 6** Deinitialize the system by calling HI\_SYS\_DeInit.

----End

#### Notes

Only the startup picture in JPEG format can be updated by using the preceding API. For the startup picture in other formats, you need to convert it into the JPEG format and then update the startup picture.

## Samples

See sample\mce\sample\_mce\_update.c.

### 29.4.3 Querying and Updating Fastplay Parameters and Data

#### Scenario

The fastplay parameters and contents can be queried and updated by using the APIs of the PDM module.



## Working Process

To query and update fastplay parameters and data, perform the following steps:

- Step 1** Initialize the system by calling HI\_SYS\_Init.
- Step 2** Obtain the corresponding fastplay parameters by calling HI\_UNF\_PDM\_GePlayParam.
- Step 3** Transfer the new fastplay parameters and data.
- Step 4** Update the new fastplay parameters to the fastplay parameter area by calling HI\_UNF\_PDM\_UpdatePlayParam.
- Step 5** Update the new fastplay contents to the fastplay data area by calling HI\_UNF\_PDM\_UpdatePlayContent.
- Step 6** Deinitialize the system by calling HI\_SYS\_DeInit.

----End

## Notes

For DVB fastplay, only the parameters need to be updated; for TS fastplay, both the parameters and data can be updated.

## Samples

See `sample\mce\sample_mce_update.c`.



---

# Contents

---

<b>30 AI .....</b>	<b>1</b>
30.1 Overview .....	1
30.2 Features .....	1
30.3 Development Guide.....	3



# Figures

**Figure 30-1 AI data stream processing ..... 1**



## Tables

<b>Table 30-1</b> AI attributes and default configurations.....	2
---	---



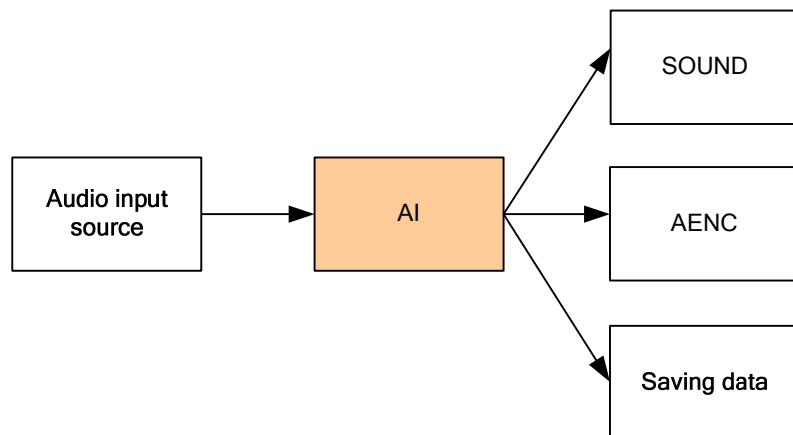
# 30 AI

## 30.1 Overview

The audio input (AI) module captures audio data from frontend modules and transmits the data to the next module for encoding or saving the audio data or other purposes.

[Figure 30-1](#) shows the data flowchart of the AI module.

**Figure 30-1** AI data stream processing



The AI module receives audio data over the I<sup>2</sup>S interface, saves the data in the specific buffer, and then transmits the data to the audio encoder (AENC) or SOUND module or saves the data for later processing. The AI module provides basic audio capturing interfaces for applications.

## 30.2 Features

The AI module provides the following functions:

- Initialization and deinitialization. The following APIs are provided:
  - HI\_UNF\_AI\_Init: Initializes the AI module and starts the AI device.
  - HI\_UNF\_AI\_DeInit: Deinitializes the AI module and stops the AI device.



- Audio capturing management, including creating and destroying the AI channel and setting and obtaining AI channel attributes. The following APIs are provided:
  - HI\_UNF\_AI\_GetDefaultAttr: Obtains the default AI channel attributes.
  - HI\_UNF\_AI\_Create: Creates an AI channel, allocates buffers for the AI channel, and sets attributes for the AI channel. Each input interface is exclusively used by an AI channel and cannot be shared.
  - HI\_UNF\_AI\_Destroy: Destroys the AI channel and releases AI channel resources.
  - HI\_UNF\_AI\_GetAttr: Obtains AI channel attributes.
  - HI\_UNF\_AI\_SetAttr: Sets AI channel attributes (you need to stop the AI first).
- Audio capturing control, including enabling and disabling the AI channel. The following API is provided:
  - HI\_UNF\_AI\_SetEnable: Enables or disables the AI channel.
- Audio frame obtaining and releasing. The following APIs are provided:
  - HI\_UNF\_AI\_AcquireFrame: Obtains an audio frame.
  - HI\_UNF\_AI\_ReleaseFrame: Releases the data buffer. Data in the buffer is invalid after the buffer is released.

Table 30-1 describes the attribute parameters and default configurations of the AI module.

**Table 30-1** AI attributes and default configurations

Attribute	Default Configuration	Description
enAiPort	None (user-defined)	Audio input interface (only the I <sup>2</sup> S interface is supported)
enSampleRate	HI_UNF_SAMPLE RATE_48K	AI sampling rate
u32PcmFrameMaxNum	6	Maximum number of frames that can be buffered
u32PcmSamplesPerFrame	960	Number of sampling points for each frame of pulse code modulation (PCM) data
stAdcAttr	None	ADC attributes, valid only when the port is ADC (not supported)
stHDMIAttr	None	HDMI attributes, valid only when the port is ADC (not supported)
stI2sAttr. bMaster	HI_TRUE	Whether the clock and reset generator (CRG) is the master mode
stI2sAttr.enI2sMode	HI_UNF_I2S_STD_MODE	Interface mode, valid only when the I <sup>2</sup> S interface is used
stI2sAttr.enMclkSel	HI_UNF_I2S_MCLK_256_FS	Frequency division relationship between the master clock (MCLK) and sampling clock (FS), valid only when the I <sup>2</sup> S interface is used MCLK = FS x enMclkSel



Attribute	Default Configuration	Description
stI2sAttr.stAttr.enBclkSel	HI_UNF_I2S_BCLK_4_DIV	Frequency division relationship between the bit clock (BCLK) and sampling clock (FS), valid only when the I <sup>2</sup> S interface is used BCLK = FS x enMclkSel
stI2sAttr.stAttr.enChannel	HI_UNF_I2S_CHNUM_2	Number of channels, valid only when the I <sup>2</sup> S interface is used
stI2sAttr.stAttr.enBitDepth	HI_UNF_I2S_BIT_DEPTH_16	Data bit width, valid only when the I <sup>2</sup> S interface is used
stI2sAttr.stAttr.bPcmSampleRiseEdge	HI_TRUE	Whether data is captured at the rising edge of the clock, valid only when the I <sup>2</sup> S interface is used
stI2sAttr.stAttr.enPcmDelayCycle	HI_UNF_I2S_PCM_0_DELAY	Number of delayed BCLK cycles of data relative to the frame sync signal, valid only when the PCM mode and I <sup>2</sup> S interface are used

## 30.3 Development Guide

### Scenario

Data from the AI interface is captured, stored to the specific buffer, and then transmitted to the AENC module or saved for other purposes.

### Working Process

The process for capturing audio data is as follows:

- Step 1** Initialize the AI module and audio input devices by calling HI\_UNF\_AI\_Init. If the SOUND or AENC device is connected, initialize the SOUND or AENC module.
- Step 2** Obtain the default AI channel attributes by calling HI\_UNF\_AI\_GetDefaultAttr, and set the audio input interface, sampling rate, number of audio channels, and bit width as required.
- Step 3** Create an AI handle by calling HI\_UNF\_AI\_Create.
- Step 4** Enable audio capturing by calling HI\_UNF\_AI\_SetEnable.
- Step 5** Create a thread, call HI\_UNF\_AI\_AcquireFrame and HI\_UNF\_AI\_ReleaseFrame in the thread to obtain audio frames from the AI buffer and release the buffer. If the AI is bound to the AENC, skip this step.
- Step 6** Disable audio capturing by calling HI\_UNF\_AI\_SetEnable.
- Step 7** Destroy the AI handle by calling HI\_UNF\_AI\_Destroy.
- Step 8** Deinitialize the AI module and related devices by calling HI\_UNF\_AI\_DeInit.



----End

## Notes

Note the following:

- Currently only the audio input interfaces I<sup>2</sup>S0 and I<sup>2</sup>S1 are supported.
- You need to disable the AI channel by calling HI\_UNF\_AI\_SetEnable before setting AI attributes by calling HI\_UNF\_AI\_SetAttr.
- Each input interface is exclusively used by an AI channel and cannot be shared by multiple AI channels.

## Sample

See `sample/ai/ai.c`.



# Contents

---

<b>31 AENC.....</b>	<b>1</b>
31.1 Overview .....	1
31.2 Important Concepts .....	2
31.3 Features .....	2
31.4 Development Guide.....	3
31.4.1 Transcoding.....	3
31.4.2 Mirroring.....	5
31.4.3 Transmitting Frames by the User .....	5



## Figures

<b>Figure 31-1</b> AENC module data processing .....	1
<b>Figure 31-2</b> Working process of the AENC module.....	4



## Tables

<b>Table 31-1 AENC attributes and default configurations .....</b>	3
--	---

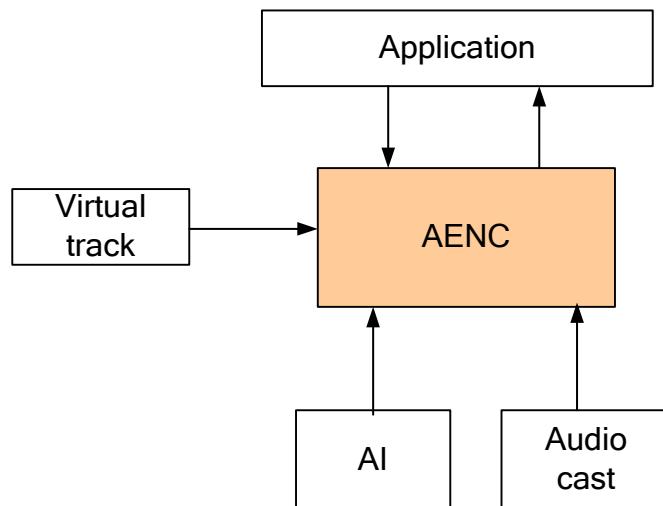


# 31 AENC

## 31.1 Overview

The audio encoder (AENC) module receives audio data transmitted from upper-level modules or users, processes the data by using algorithms such as algorithms for sampling rate conversion, sampling bit width conversion, and channel number conversion, and generates streams based on the required target sampling rate, sampling bit width, and number of channels. The generated stream data is returned for use, for example, the data is transmitted over the network to a remote end for decoding and display.

**Figure 31-1** AENC module data processing



Data transmitted to the AENC module mainly comes from the AI module, virtual track, audio cast, or upper-layer applications. The AENC module encodes the input data and then transmits the data to upper-layer application for other use.



## 31.2 Important Concepts

[Target sampling rate]

The target sampling rate is an output sampling rate (16 kHz to 48 kHz) configured in the encoder by the user as required. When the output sampling rate is different from the sampling rate of input streams (input sampling rate, 8 kHz to 192 kHz), the input sampling rate is converted by using the resampling module. If the difference between the output sampling rate and input sampling rate is large, the encoding effect is poor.

[Target sampling bit width]

The target sampling bit width is an output sampling bit width (16 bits) configured in the encoder by the user as required. When the output sampling bit width is different from the sampling bit width of input streams (input sampling bit width, 16 bits or 24 bits), the input sampling bit width is converted.

[Target number of audio channels]

The target number of audio channels is the number of output audio channels (only two channels are supported) configured in the encoder by the user as required. When the number of output audio channels is different from the number of input audio channels (one or two), the number of audio channels is changed.

[Binding mode]

The binding mode is an encoding mode supported by the AENC module. When the working mode is set to binding mode, the AENC is started after an input source is bound to it. Currently the input source can be the virtual track, audio cast, or AI module. After the binding mode is set, you cannot transmit frames to the AENC for decoding. Data is internally controlled and read by the SDK.

## 31.3 Features

The AENC module provides the following functions:

- AENC module management, including initializing and deinitializing the AENC module and creating and destroying audio encoding instances. The following APIs are provided:
  - HI\_UNF\_AENC\_Init: Initializes the AENC module.
  - HI\_UNF\_AENC\_DeInit: Deinitializes the AENC module.
  - HI\_UNF\_AENC\_Create: Creates an AENC instance.
  - HI\_UNF\_AENC\_Destroy: Destroys an AENC instance.
- Basic encoding services. The following APIs are provided:
  - HI\_UNF\_AENC\_RegisterEncoder: Registers an audio encoding library.
  - HI\_UNF\_AENC\_Start: Starts an AENC instance.
  - HI\_UNF\_AENC\_Stop: Stops an AENC instance.
  - HI\_UNF\_AENC\_SetAttr: Sets attributes of the AENC instance.
  - HI\_UNF\_AENC\_GetAttr: Obtains attributes of the AENC instance.
  - HI\_UNF\_AENC\_AttachInput: Binds the AENC instance to an input source.
  - HI\_UNF\_AENC\_DetachInput: Unbinds the AENC instance from the input source.
  - HI\_UNF\_AENC\_SendFrame: Sends frames to the AENC for encoding.



- HI\_UNF\_AENC\_AcquireStream: Obtains encoded streams output by the AENC.
- HI\_UNF\_AENC\_ReleaseStream: Releases the buffer for storing encoded streams output by the AENC.

[Table 31-1](#) describes the attribute parameters and default configurations of the AENC module.

**Table 31-1** AENC attributes and default configurations

Attribute	Default Configuration	Description
enAencType	HA_AUDIO_ID_AAC	Type of the audio encoding protocol Currently only the advanced audio coding (AAC) encoding is supported.
sOpenParam	-	AENC initialization parameters (including private attributes of the encoder) Currently only the output sampling rate can be changed.

## 31.4 Development Guide

The AENC module is used in the following scenarios:

- Transcoding
- Mirroring
- Transmitting frames by the user

### 31.4.1 Transcoding

#### Scenario

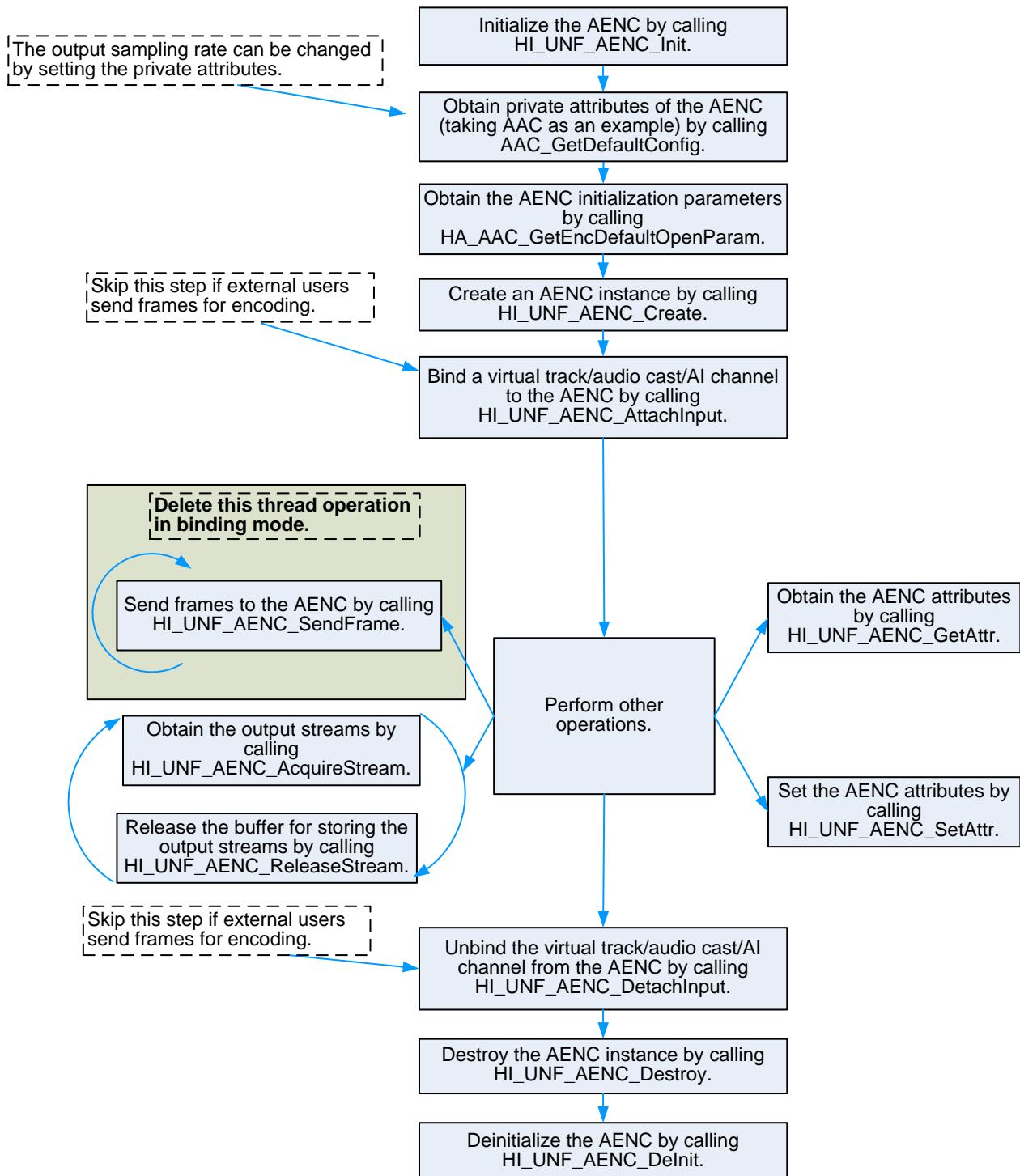
In the transcoding scenario, the AENC module is bound to an input source (virtual track). It encodes the audio PCM data received from the virtual track into streams that can be identified by terminals such as mobile phones, Pads, or PCs for playback.

#### Working Process

[Figure 31-2](#) shows the working process of the AENC module.



**Figure 31-2** Working process of the AENC module



## Notes

Note the following:

- Before binding the AENC to a virtual track, ensure that the virtual track is initialized and created.



- Bind the AENC to the input source before the encoding starts, and unbind the AENC from the input source after the encoding stops.
- During the binding process, the input source cannot transmit frames to the AENC instance. After the AENC is unbound from the input source, it can be bound to another input source, or frames can be input externally for encoding.
- Stop encoding before setting AENC attributes by calling `HI_UNF_AENC_SetAttr`. Encoding can be started again after the AENC attributes are configured.
- Ensure that the attribute values are valid when configuring the AENC attributes by calling `HI_UNF_AENC_SetAttr`. That is, the parameter values fall within the specified value ranges, and the non-dynamic attributes cannot be changed. You are advised to obtain the current AENC attributes by calling `HI_UNF_AENC_GetAttr` and then change the attributes as required by calling `HI_UNF_AENC_SetAttr`.

## Sample

See `sample/transcode/dvb_transcode.c` or `sample/transcode/ts_transcode.c`.

### 31.4.2 Mirroring

#### Scenario

For the AENC module, the mirroring scenario is similar to the transcoding scenario. However, the input source of the transcoding scenario is a virtual track, whereas that of the mirroring scenario is data obtained after resampling, audio channel number changing, and audio mixing of multiple tracks. In the mirroring scenario, the AENC is bound to the audio cast module.

#### Working Process

For details, see [Figure 31-2](#). Note that the AENC is bound to the audio cast module in this scenario.

#### Notes

See the notes in section [31.4.1 "Transcoding."](#)

## Sample

See `sample/audiocast/cast_aenc.c`.

### 31.4.3 Transmitting Frames by the User

#### Scenario

In this scenario, the AENC is not bound to any module. You can transmit audio frames to the AENC module for encoding as required, and then obtain streams after encoding.

#### Working Process

For details, see [Figure 31-2](#). Note that the binding and unbinding operations are not required in this scenario. You need to transmit frames to the AENC by calling `HI_UNF_AENC_SendFrame` repeatedly.



## Notes

Note the following:

- Ensure that the configured attributes of input audio frames are correct. Otherwise, the internal encoding processing is affected, and the streams after encoding cannot be played properly.
- You can transmit audio frames to the AENC by calling HI\_UNF\_AENC\_SendFrame in the thread.

## Sample

See `sample/aenc/aenc.c`.



# Contents

---

<b>32 VI .....</b>	<b>1</b>
32.1 Overview .....	1
32.2 Important Concepts .....	1
32.3 Function Description .....	2
32.3.1 Features .....	2
32.4 Development Guide.....	3
32.4.1 Capturing Video Data by Using the Virtual VI .....	3



## Figures

<b>Figure 32-1</b> VI module among modules related to video processing .....	1
<b>Figure 32-2</b> Function implementation for the virtual VI .....	3



## Tables

<b>Table 32-1</b> VI attributes and default configurations .....	2
--	---



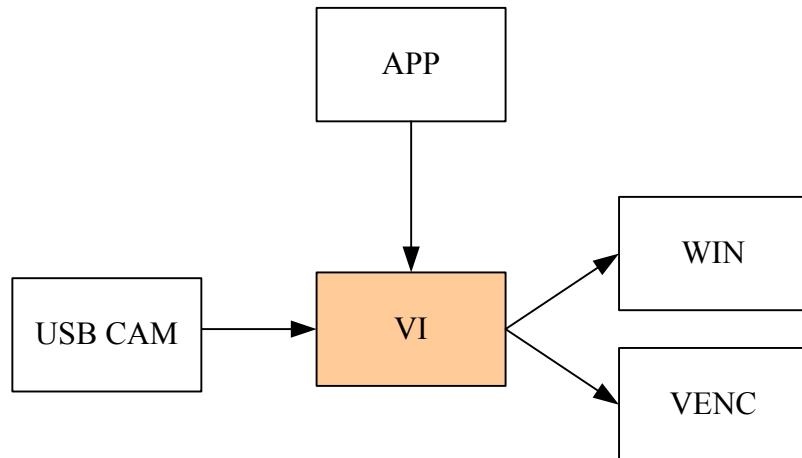
# 32 VI

## 32.1 Overview

The video input (VI) module captures video data from upper-level modules, internally crops video frames, and transmits the processed video data to the next module for encoding or display.

[Figure 32-1](#) shows the VI module among modules related to video processing.

**Figure 32-1** VI module among modules related to video processing



The VI module receives the video data pushed by the user (normally the video data captured by a USB camera), and then transmits the data to the video encoder (VENC) or window (WIN) module for further processing. The VI module provides basic video capturing interfaces for applications.

## 32.2 Important Concepts

[Virtual VI]

Different from the entity VI, the virtual VI captures pushed video data, such as video data from the USB camera.



## 32.3 Function Description

### 32.3.1 Features

The VI module provides the following functions:

- Video capturing management, including creating and destroying the VI instance and setting and obtaining VI attributes. The following APIs are provided:
  - HI\_UNF\_VI\_GetDefaultAttr: Obtains default VI attributes.
  - HI\_UNF\_VI\_Create: Creates a VI instance.
  - HI\_UNF\_VI\_Destroy: Destroys a VI instance.
  - HI\_UNF\_VI\_GetAttr: Obtains VI attributes.
  - HI\_UNF\_VI\_SetAttr: Sets VI attributes.
- Video capturing control, including starting and stopping the VI module. The following APIs are provided:
  - HI\_UNF\_VI\_Start: Starts the VI module.
  - HI\_UNF\_VI\_Stop: Stops the VI module.
- Video frame input, used by the virtual VI. The following APIs are provided:
  - HI\_UNF\_VI\_QueueFrame: Inputs an image frame for VI processing.
  - HI\_UNF\_VI\_DequeueFrame: Releases the buffer for storing the input image after VI processing.
- Video frame obtaining. The following APIs are provided:
  - HI\_UNF\_VI\_AcquireFrame: Obtains an image frame from the VI.
  - HI\_UNF\_VI\_ReleaseFrame: Releases the buffer for storing a frame.

Table 32-1 describes the attribute parameters and default configurations of the VI module.

**Table 32-1** VI attributes and default configurations

Attribute	Default Configuration	Description
bVirtual	HI_FALSE	Whether to create the virtual VI
stInputRect	(0, 0, 720, 576)	Input video aspect ratio. The default start coordinates are (0, 0).
enVideoFormat	HI_UNF_FORMAT_Y_UV_SEMIPLANAR_420	Input video format
u32BufNum	6	Number of available frame buffers, ranging from 4 to 16

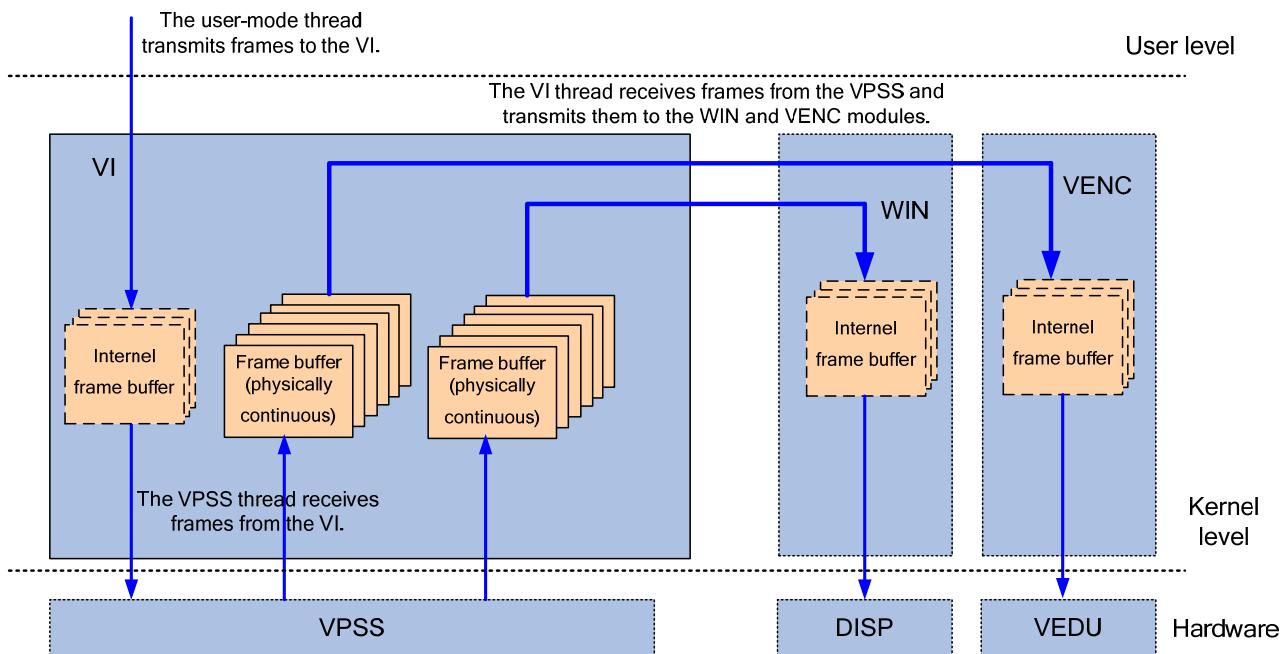
#### 32.3.1.2 Virtual VI

When the virtual VI is working, the hardware entity VI is not required. The user creates a thread to transmit frames to the VI, and the VI stores them to the internal queue after it obtains the physical address for the external frame buffer. The received frame data is



transmitted to the VPSS for scaling and one-to-multiple dividing. The VI thread receives the frame data processed by the VPSS and then transmits them to the WIN and VENC modules for video display and encoding respectively.

**Figure 32-2** Function implementation for the virtual VI



## 32.4 Development Guide

The VI module is used in the following scenario.

### 32.4.1 Capturing Video Data by Using the Virtual VI

#### Scenario Description

The virtual VI captures external video data, such as video frames from the USB camera, and transmits the data to the WIN and VENC modules for loopback display and video encoding.

#### Working Process

To capture video data by using the virtual VI, perform the following steps:

- Step 1** Initialize the VI module and related devices. If the WIN or VENC module is connected, initialize the VO and VENC modules.
- Step 2** Create a VO window for displaying captured videos.
- Step 3** Create a virtual VI instance by calling `HI_UNF_VI_Create`. Ensure that the virtual mode is set to `HI_TRUE`, and the number of internal buffers is specified. Other attributes can be ignored.
- Step 4** Obtain the default VENC attributes by calling `HI_UNF_VENC_GetDefaultAttr`.



- Step 5** Create a VENC by calling HI\_UNF\_VENC\_Create.
- Step 6** Bind the window to the VI instance by calling HI\_UNF\_VO\_AttachWindow.
- Step 7** Enable the window by calling HI\_UNF\_VO\_SetWindowEnable.
- Step 8** Bind the VENC to the VI instance by calling HI\_UNF\_VENC\_AttachInput.
- Step 9** Start video capturing by calling HI\_UNF\_VI\_Start.
- Step 10** Start VENC encoding by calling HI\_UNF\_VENC\_Start.
- Step 11** Create a thread, call HI\_UNF\_VI\_QueueFrame and HI\_UNF\_VI\_DequeueFrame in the thread to push video frames to the VI, and release a frame VI processing is complete.
- Step 12** Create a thread, and call HI\_UNF\_VENC\_AcquireStream and HI\_UNF\_VENC\_ReleaseStream in the thread to obtain encoded data from the VENC.
- Step 13** Disable the window by calling HI\_UNF\_VO\_SetWindowEnable.
- Step 14** Destroy the threads for obtaining streams from the VENC and transferring streams to the VI.
- Step 15** Stop video capturing by calling HI\_UNF\_VI\_Stop.
- Step 16** Disable the window by calling HI\_UNF\_VO\_SetWindowEnable.
- Step 17** Unbind the window from the VI instance by calling HI\_UNF\_VO\_DetachWindow.
- Step 18** Destroy the window by calling HI\_UNF\_VO\_DestroyWindow.
- Step 19** Stop VENC encoding by calling HI\_UNF\_VENC\_Stop.
- Step 20** Unbind the VENC from the VI instance by calling HI\_UNF\_VENC\_DetachInput.
- Step 21** Destroy the VENC by calling HI\_UNF\_VENC\_Destroy.
- Step 22** Destroy the VI instance by calling HI\_UNF\_VI\_Destroy.
- Step 23** Deinitialize the VI module and related devices.

----End

## Notes

- If the VI is bound to a lower-level module, video frames cannot be obtained.
- Frames can be successfully transferred to the VI by calling HI\_UNF\_VI\_QueueFrame and HI\_UNF\_VI\_DequeueFrame in the user thread only after the VI is started.

## Sample

See `sample/vi_venc/sample_vivenc.c`.



# Contents

---

<b>33 VENC.....</b>	<b>1</b>
33.1 Overview .....	1
33.2 Important Concepts .....	1
33.3 Function Description .....	3
33.3.1 Features.....	3
33.3.2 Function Implementation .....	7
33.4 Development Guide.....	8
33.4.1 Transcoding.....	8
33.4.2 Video Phone .....	10
33.4.3 Mirroring.....	11
33.4.4 Transmitting Frames by the User .....	11



## Figures

<b>Figure 33-1</b> VENC module data processing .....	1
<b>Figure 33-2</b> VENC working principles.....	8
<b>Figure 33-3</b> Working process of the VENC module.....	9



## Tables

<b>Table 33-1</b> VENC attributes and default configurations.....	3
<b>Table 33-2</b> Bit rate configuration reference (frame rate: 30 fps) .....	7

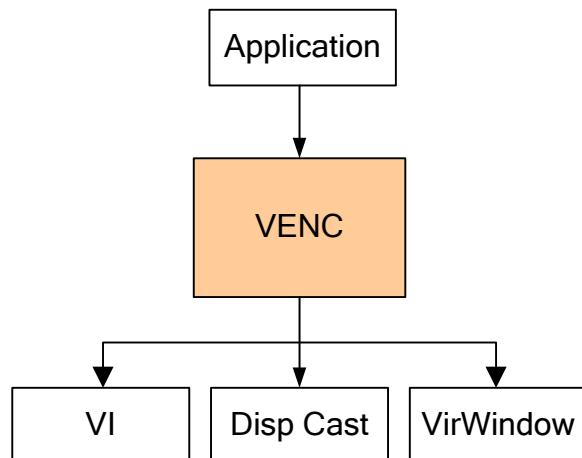


# 33 VENC

## 33.1 Overview

The video encoder (VENC) module receives video and graphics data transmitted from upper-level modules or users, processes the data by using protocols or algorithms such as those for frame rate control, bit rate control, inter-frame prediction, intra-frame prediction, conversion, and quantization, and generates streams based on the required output frame rate, target bit rate, and resolution. The generated stream data is returned for use, for example, the data is transmitted over the network to a remote end for decoding and display.

**Figure 33-1** VENC module data processing



The VENC module relies on the VI, Disp Cast, and VirWindow modules to provide video encoding functions for upper-layer applications.

## 33.2 Important Concepts

[Target bit rate]

The target bit rate is the encoding bit rate configured in the encoder by the user as required. The difference between the target bit rate and the actual output bit rate cannot be greater than 20%. The target bit rate affects the picture quality. The appropriate bit rate range varies



according to the resolution. If the bit rate is lower than the minimum value of the appropriate range, the picture quality is unacceptable; if the bit rate is higher than the maximum value of the appropriate range, the network and storage resources are wasted. You can set the target bit rate based on the encoding resolution and real-time network conditions in the actual application scenarios to achieve excellent encoding effect.

#### [Frame rate]

The frame rate is used to measure the number of display frames. The unit is fps, indicating the number of frames displayed per second. You can set the input and output frame rates for the VENC. If the two frame rates are inconsistent, the VENC automatically controls them. Note that the input frame rate configured by the user is invalid in binding mode, because the VENC controls the frame rate by using the actual input frame rate as reference. However, in non-binding mode, ensure that the configured input frame rate is correct.

#### [GOP]

The group of pictures (GOP) indicates a group of consecutive pictures. The GOP is the minimum unit for the decoder to restore pictures. A GOP typically starts with an I frame and ends with the frame before the next I frame. The GOP length affects the encoding quality. A longer GOP length indicates higher picture compression efficiency.

#### [QP]

The quantization parameter (QP) is an important parameter required during compression encoding quantization. There are two QPs in the VENC attributes: Max QP and Min QP. This allows you to control the QP range of the encoder. The picture quality and bit rate can be controlled by adjusting the QP within the range [Min QP, Max QP]. When [Min QP, Max QP] falls within the fore part of [0, 51], most details are retained during encoding, the bit rate is high, picture distortion is not obvious, and picture quality is good; otherwise, details are lost, the bit rate is low, picture distortion is obvious, and picture quality is reduced.

#### [Scaling encoding]

Scaling encoding is an encoding mode supported by the VENC. In this mode, the input video source is scaled and then encoded. The resolution of the video source is ignored, and the resolution of streams after encoding is the same as that configured in the VENC attributes. If the video source needs to be scaled at an uneven proportion, it is directly compressed, stretched, or tiled.

#### [Slice encoding]

Slice encoding is an encoding mode based on the H.264 protocol. In this mode, the VENC encodes a picture frame into multiple slices, which are independent of each other during encoding. When the network transmission environment is poor, slice encoding helps reduce bit errors.

#### [Fast encoding mode]

The fast encoding mode is an encoding mode supported by the VENC module. It is used in scenarios that have high requirements on real-time encoding, such as the video phone scenario. In this mode, the VENC controls the frame rate based on the configured input and output frame rates. To ensure real-time encoding and avoid delays, the VENC only encodes the latest frame that is inserted and discards other frames in the buffer. This ensures that the latest picture in the buffer is encoded.

#### [Binding mode]

The binding mode is an encoding mode supported by the VENC module. To set the working mode to binding mode, you need to bind the VENC to a video source before starting the



VENC. The video source can be the virtual screen, virtual window, or the VI. After setting the binding mode, you cannot directly transmit frames to the VENC for encoding.

## 33.3 Function Description

### 33.3.1 Features

The VENC module provides the following functions:

- VENC module management, including initializing and deinitializing the VENC module and creating and destroying VENC instances. The following APIs are provided:
  - HI\_UNF\_VENC\_Init: Initializes the VENC module.
  - HI\_UNF\_VENC\_DeInit: Deinitializes the VENC module.
  - HI\_UNF\_VENC\_GetDefaultAttr: Obtains default attributes of the VENC.The obtained default attribute values are those that apply to most scenarios. You are advised to set the attributes based on the reference data in [Table 33-2](#) to achieve better encoding compression effect and picture quality.
  - HI\_UNF\_VENC\_Create: Creates a VENC instance.
  - HI\_UNF\_VENC\_Destroy: Destroys a VENC instance.
- Basic encoding services. The following APIs are provided:
  - HI\_UNF\_VENC\_Start: Starts a VENC instance.
  - HI\_UNF\_VENC\_Stop: Stops a VENC instance.
  - HI\_UNF\_VENC\_SetAttr: Sets attributes of the VENC instance.
  - HI\_UNF\_VENC\_GetAttr: Obtains attributes of a VENC instance.
  - HI\_UNF\_VENC\_RequestIFrame: Requests an I frame. The VENC encodes an I frame as soon as possible.
  - HI\_UNF\_VENC\_AttachInput: Sets the binding mode.
  - HI\_UNF\_VENC\_DetachInput: Unbinds the VENC instance from a video source.
  - HI\_UNF\_VENC\_QueueFrame: Sends frames to the VENC for encoding.
  - HI\_UNF\_VENC\_DequeueFrame: Returns the processed frames and releases the frame buffer.
  - HI\_UNF\_VENC\_AcquireStream: Obtains encoded streams output by the VENC.
  - HI\_UNF\_VENC\_ReleaseStream: Releases the buffer for storing encoded streams output by the VENC.

[Table 33-1](#) describes the attribute parameters and default configurations of the VENC module.

**Table 33-1** VENC attributes and default configurations

Attribute	Default Configuration	Dynamic Configuration Allowed	Description
enVencType	HI_UNF_VCOP_EC_TYPE_H264	No	Type of the video encoding protocol
enCapLevel	HI_UNF_VCOP_EC_CAP_LEVEL	No	Encoding capability level



Attribute	Default Configuration	Dynamic Configuration Allowed	Description
	L_720P		
enVencProfile	HI_UNF_H264_PROFILE_HIGH	No	H.264 encoding level This attribute is valid only when the H.264 encoder is defined.
u32Width	1280	Yes	Encoding output height/width <ul style="list-style-type: none"><li>The output resolution must fall within the range of encoding capability level.</li></ul>
u32Height	720	Yes	<ul style="list-style-type: none"><li>The output resolution can be dynamically configured within the range of encoding capability level.</li><li>The height and width of the encoding resolution must be the integral multiple of 4.</li></ul>
u32StrmBufSize	1280*720*2	No	Size of the buffer for storing output streams The relationship between the size of the stream buffer and the encoding capability level is as follows: Resolution of the encoding capability level (width x height) x 2 ≤ Size of the stream buffer ≤ 20 x 1024 x 1024
u32RotationAngle	0	No	Rotation angle This attribute must be set to 0 because rotated encoding is not supported currently.
bSlcSplitEn	0	No	Slice encoding mode enable <ul style="list-style-type: none"><li><b>HI_TRUE</b> indicates enabled, and <b>HI_FALSE</b> indicates disabled.</li><li>If the upper-level module enables low-delay (the open low-delay switch of the VENC module), enabling the slice encoding mode allows each slice to be transmitted to the stream buffer in time after the slice is encoded. You are advised to set the low-delay attribute of the bound upper-level module to <b>HI_TRUE</b>.</li></ul>



Attribute	Default Configuration	Dynamic Configuration Allowed	Description
u32SplitSize	2*1024	Yes	<p>Slice size</p> <ul style="list-style-type: none"><li>• This attribute is valid only when <b>bSlcSplitEn</b> is set to 1.</li><li>• The unit is byte.</li></ul> <p>According to the H.264 protocol, certain margin must be reserved for the slice size. Because of the VEDU limitation, the reserved margin must be the slice size in the worst scenario plus 2304 bytes.</p>
u32TargetBitRate	4*1024*1024	Yes	<p>Target bit rate</p> <ul style="list-style-type: none"><li>• The value ranges from 32 x 1024 to 50 x 1024 x 1024, and the unit is bit/s.</li><li>• For details about the reference values of the target bit rate in various scenarios, see <a href="#">Table 33-2</a>.</li></ul>
u32TargetFrmRate	25	Yes	<p>Target frame rate</p> <p>The value ranges from 1 to <b>InputFrmRate</b>.</p>
u32InputFrmRate	25	Yes	<p>Input frame rate</p> <ul style="list-style-type: none"><li>• The value ranges from 1 to 60.</li><li>• This attribute is invalid in binding mode. The frame rate of the video source is used as the input frame rate in binding mode.</li></ul>
u32Gop	100	Yes	<p>GOP length</p> <ul style="list-style-type: none"><li>• The value ranges from 1 to 0xffffffff.</li><li>• It is recommended that the value range from 30 to 100 in the transcoding or mirroring scenario.</li><li>• It is recommended that the value range from 100 to 300 in the video phone scenario.</li></ul>
u32MaxQp	48	Yes	<p>Maximum/Minimum QP</p> <ul style="list-style-type: none"><li>• The value ranges from 0 to 51, and <b>u32MaxQp</b> must be greater than <b>u32MinQp</b>.</li></ul>
u32MinQp	16	Yes	



Attribute	Default Configuration	Dynamic Configuration Allowed	Description
			<ul style="list-style-type: none"><li>• These two attributes ensure that the VENC adjusts the QP within the configured range [MinQp, MaxQp]. For details, see section 34.2.</li></ul>
u32Qlevel	0	Yes	JPEG quantization level <ul style="list-style-type: none"><li>• The value ranges from 1 to 99.</li><li>• This attribute is valid only when the JPEG encoder is defined.</li></ul>
bQuickEncode	0	Yes	Fast encoding mode enable <ul style="list-style-type: none"><li>• <b>HI_TRUE</b> indicates enabled, and <b>HI_FALSE</b> indicates disabled.</li><li>• You are advised to set this attribute to <b>HI_TRUE</b> if the scenario requires high real-time encoding performance.</li></ul>
u8Priority	0	Yes	VENC instance priority. It is used to determine the priority when multiple VENC instances exist. The VENC instance with higher priority is scheduled in priority. The VENC instances with the same priority are scheduled in sequence. <ul style="list-style-type: none"><li>• The value ranges from 0 to 7.</li><li>• 0 indicates the lowest priority, and 7 indicates the highest priority. You can set the priority levels of multiple VENC instances to the same.</li></ul>
u32DriftRateThr	0xffffffff	Yes	Encoder bit rate drift threshold The value ranges from 0 to 100, or 0xffffffff. If it is set to 20, the drift threshold is $\pm 20\%$ . Within the value range from 0 to 100, a smaller <b>u32DriftRateThr</b> value indicates stricter bit rate control and smaller drift range.

Table 33-2 describes the reference range of the appropriate target bit rate in various scenarios.

**Table 33-2** Bit rate configuration reference (frame rate: 30 fps)

Resolution	Video Phone		Transcoding/Mirroring/Transmitting Frames by the User	
	Range (Mbit/s)	Recommended Value (Mbit/s)	Range (Mbit/s)	Recommended Value (Mbit/s)
1080p	2–8	4	2–10	6
720p	1–5	2	1–6	3
D1 to VGA	0.5–3	1.5	0.5–4	2
Lower than CIF	0.5–2	1	0.5–3	1.5

### 33.3.2 Function Implementation

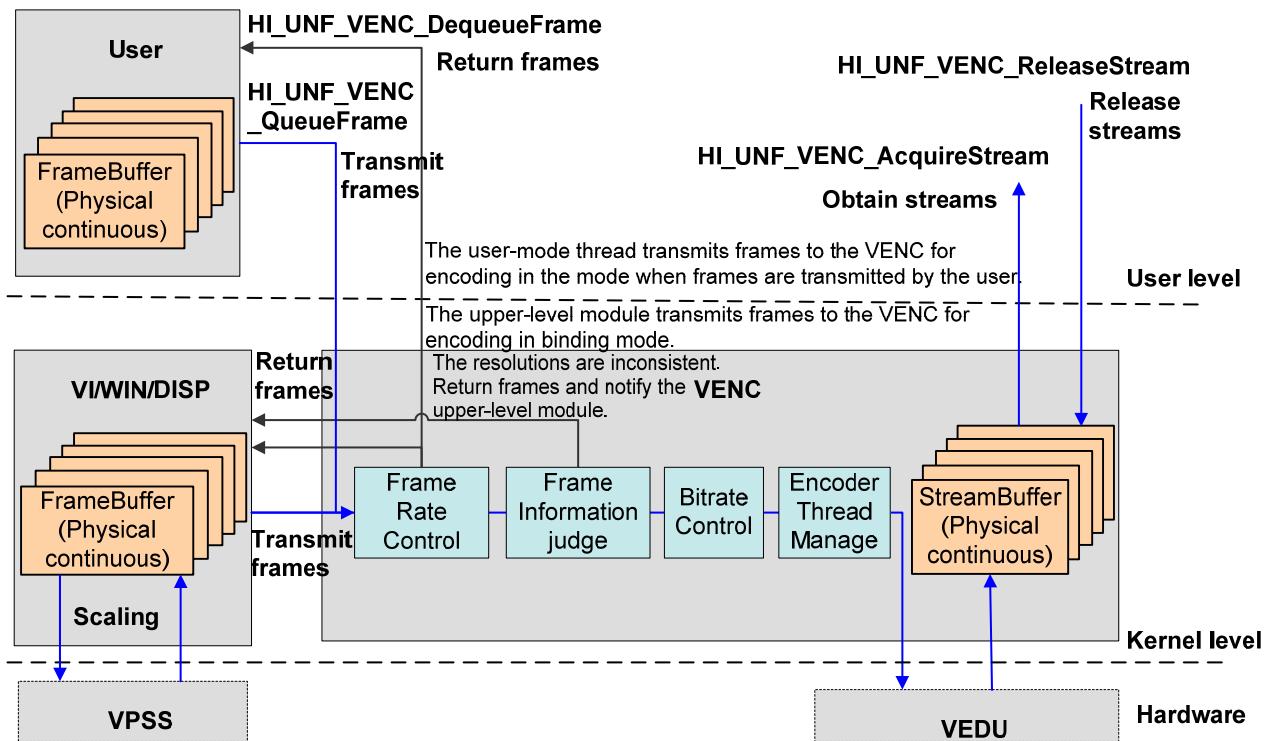
The VNEC module works in either of the following two modes:

- Mode in which video frames are transmitted externally by the user
- Binding mode

The application or upper-level module sends the frame buffer information to the VENC for encoding. The VENC implements frame rate control and bit rate control over the received frame buffer, starts the VEDU hardware to encode the frames that need to be encoded, and stores the encoded streams in the buffer. Then the user can obtain the streams and transmit the streams over the network to a remote end for decoding and display.

[Figure 33-2](#) shows the working principles of the VENC module. If the resolution in the frame information and the encoding resolution are inconsistent in binding mode, the VENC module asks the bound upper-level module.

Figure 33-2 VENC working principles



## 33.4 Development Guide

The VENC module is used in the following scenarios:

- Transcoding
- Video phone
- Mirroring
- Transmitting frames by the user

### 33.4.1 Transcoding

#### Scenario

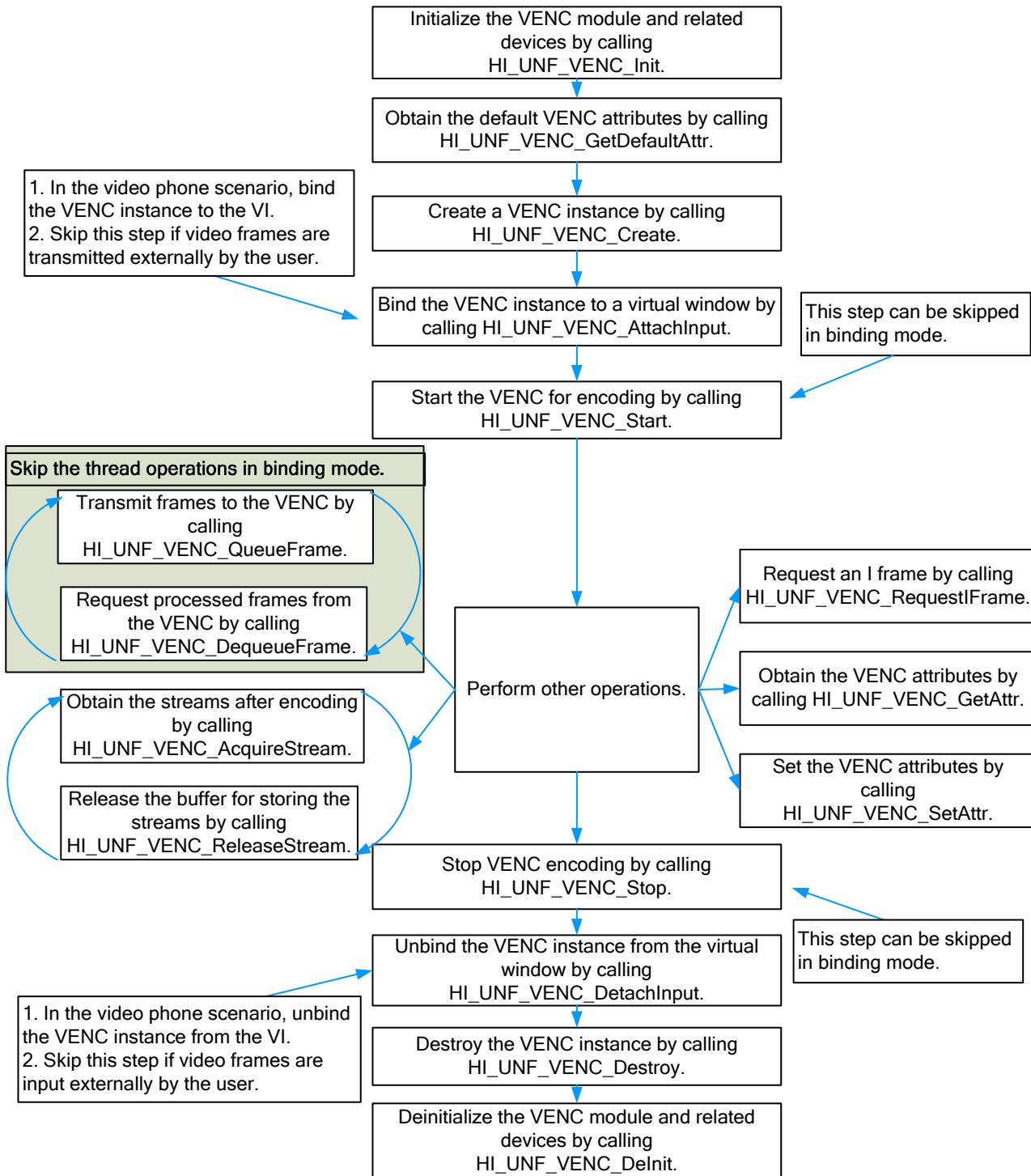
In the transcoding scenario, the VENC module is bound to an input source (virtual window). It encodes the video data received from the virtual window into streams that can be identified by terminals such as mobile phones, Pads, or PCs for playback.

#### Working Process

Figure 33-3 shows the working process of the VENC module.



**Figure 33-3** Working process of the VENC module



## Notes

Note the following:



- Before binding the VENC to a virtual window, ensure that the virtual window is initialized and created.
- After AttachInput is executed, the VENC is started, and you do not need to call HI\_UNF\_VENC\_Start to start the VENC. After DetachInput is executed, the VENC is stopped, and you do not need to call HI\_UNF\_VENC\_Stop to stop the VENC.
- In binding mode, external devices cannot transmit frames to the VENC or request frames from the VENC. After the VENC is unbound from the input source, it can be bound to another input source, or frames can be transmitted externally for encoding.
- In the transcoding or mirroring scenario, the contents to be encoded are highly unpredictable and constantly variable, and the scenario is frequently switched. In this case, you can set a large target bit rate and a small GOP value to ensure good picture quality as the scenario often applies to the environment with high bandwidth such as the wireless LAN.
- Ensure that the attribute values are valid when configuring the AENC attributes by calling HI\_UNF\_VENC\_SetAttr. That is, the parameter values are within the specified value ranges, and the non-dynamic attributes cannot be changed. You are advised to obtain the current VENC attributes by calling HI\_UNF\_VENC\_GetAttr and then change the attributes as required by calling HI\_UNF\_VENC\_SetAttr.

## Sample

See [sample/transcode/dvb\\_transcode.c](#) or [sample/transcode/ts\\_transcode.c](#).

## 33.4.2 Video Phone

### Scenario

In the video phone scenario, the VENC module is bound to an input source (VI module). It encodes video data received from the VI module and transmits the encoded streams to a remote end for decoding and display.

### Working Process

For details, see [Figure 33-3](#). Note that the VENC is bound to the VI module in this scenario.

### Notes

- In the video phone scenario, the contents to be encoded are relatively smooth and not fierce. You can set a small target bit rate and a large GOP value to increase the video compression ratio and reduce network transmission costs.
- The video phone scenario requires high real-time encoding performance. You can statically or dynamically enable the fast encoding mode to improve real-time encoding performance and avoid delays during encoding.
- For details about other information, see the notes in section [33.4.1 "Transcoding."](#)

## Sample

See [sample/vi\\_venc/sample\\_vivenc.c](#).



### 33.4.3 Mirroring

#### Scenario

For the VENC module, the mirroring scenario is similar to the transcoding scenario. However, transcoding involves only the video layer, whereas mirroring involves both the video layer and the graphics layer. In the mirroring scenario, the VENC is bound to the DISPLAY module.

#### Working Process

For details, see [Figure 33-3](#). Note that the VENC is bound to the virtual device CAST of the DISPLAY module in this scenario.

#### Notes

See the notes in section [33.4.1 "Transcoding."](#)

#### Sample

See [sample/avcast/dvb\\_cast.c](#) or [sample/avcast/ts\\_cast.c](#).

### 33.4.4 Transmitting Frames by the User

#### Scenario

In this scenario, the VENC is not bound to any module. You can transmit video frames to the VENC module for encoding as required, and then obtain encoded streams.

#### Working Process

For details, see [Figure 33-3](#). Note that the binding and unbinding operations are not required in this scenario. You need to transmit frames to the VENC and request processed frames from the VENC by calling `HI_UNF_VENC_QueueFrame` and `HI_UNF_VENC_DequeueFrame` respectively.

#### Notes

- Ensure that the configured input frame rate is correct. Otherwise, the internal frame rate control is affected, which results in exceptions. For example, frames are discarded, and the bit rate is unstable.
- You can transmit video frames to the VENC and obtain processed frames from the VENC by calling `HI_UNF_VENC_QueueFrame` and `HI_UNF_VENC_DequeueFrame` in the thread.

#### Sample

See [sample/venc/sample\\_venc.c](#).



# Contents

---

<b>34 HDCPKey .....</b>	<b>1</b>
34.1 Overview .....	1
34.2 Important Concepts .....	1
34.3 Function Description .....	1
34.3.1 Features .....	1
34.3.2 Function Implementation .....	1
34.4 Development Guide.....	2



# Figures

**Figure 34-1** HDCP Key function implementation ..... 2



# 34 HDCPKey

## 34.1 Overview

The HDCPKey module encrypts the high-bandwidth digital content protection (HDCP) key by using the HDCP root key or HiSilicon key. For details about the usage of the HDCP key, see the *HDCP Key User Guide*.

## 34.2 Important Concepts

[HDCP]

- The HDCP key is used to protect data transmitted over HDMI. It is obtained from the DCP organization.
- The HDCP root key is used to encrypt the HDCP key. It is stored in the chip and cannot be read.

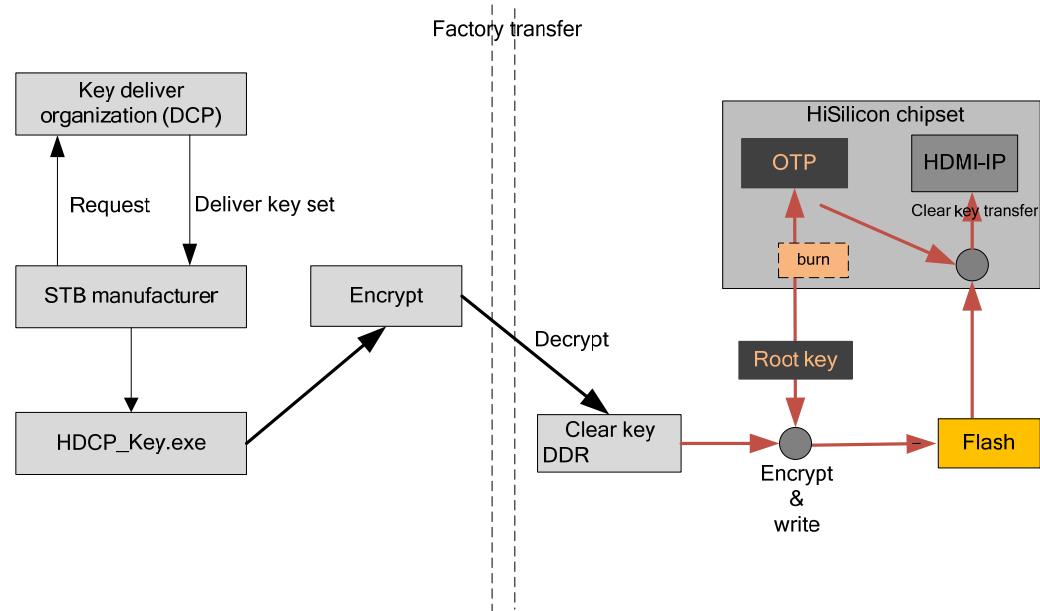
## 34.3 Function Description

### 34.3.1 Features

The HDCPKey module encrypts the HDCP key by using the HDCP root key or HiSilicon key. The following API is provided: HI\_UNF\_HDCP\_EncryptHDCPKey.

### 34.3.2 Function Implementation

[Figure 34-1](#) shows the HDCP key function implementation diagram.

**Figure 34-1** HDCP Key function implementation

- Step 1** Obtain the HDCP key from the DCP organization and encrypt it with the fixed key of HiSilicon.
- Step 2** Burn the root key for encrypting the HDCP key to the internal one-time programmable (OTP) area of the chip during factory production.
- Step 3** Encrypt the HDCP key by calling HI\_UNF\_HDCP\_EncryptHDCPKey and store the encrypted HDCP key in the flash memory during factory production.
- Step 4** The HDMI module loads the encrypted HDCP key from the flash memory and decrypts it to the chip.
- Step 5** The HDMI module obtains the HDCP key from the chip to implement content protection for data transmission.

----End

## 34.4 Development Guide

### Scenario

Before transmitting sensitive data, the HDMI module uses the HDCP technology to encrypt and protect the data content. The HDCP key is encrypted by using the specific root key and stored in the flash memory. To use the HDCP key, decrypt it first, and then protect the content to be transmitted by using the decrypted HDCP key. (The HDCP key is decrypted in the chip, and the master CPU cannot read the HDCP key.)

### Working Process

The HDCP key is encrypted as follows:



- Step 1** Initialize the cipher module by calling HI\_UNF\_CIPHER\_Init.
  - Step 2** Encrypt the HDCP key by calling HI\_UNF\_HDCP\_EncryptHDCPKey.
  - Step 3** Deinitialize the cipher module by calling HI\_UNF\_CIPHER\_DeInit.
- End

## Notes

None

## Sample

See **sample\_encryptHdcpKey.c**.



# Contents

---

<b>35 OTP .....</b>	<b>1</b>
35.1 Overview .....	1
35.2 Important Concepts .....	1
35.3 Features .....	1
35.4 Development Guide.....	2
35.4.1 Storing User Private Data.....	2
35.4.2 Storing User-Defined Data.....	2
35.4.3 Storing the HDCP Root Key .....	3
35.4.4 Storing the STB Root Key .....	4



# 35 OTP

## 35.1 Overview

The OTP area is used to store secure and sensitive data in the chip. It can be programmed for only once. Data stored in the OTP area can be saved permanently and cannot be modified.

## 35.2 Important Concepts

None

## 35.3 Features

The OTP area can be used to store the following important data:

- 16-byte user private data. The following APIs are provided:
  - HI\_UNF OTP\_SetStbPrivData: Sets private data.
  - HI\_UNF OTP\_GetStbPrivData: Obtains private data.
- 16-byte user-defined data (the function is the same as the function of the customer key). The following APIs are provided:
  - HI\_UNF OTP\_SetCustomerKey: Sets user-defined data.
  - HI\_UNF OTP\_GetCustomerKey: Obtains user-defined data.
- 16-byte HDCP root key. The following APIs are provided:
  - HI\_UNF OTP\_WriteHdcpRootKey: Sets the HDCP root key.
  - HI\_UNF OTP\_ReadHdcpRootKey: Obtains the HDCP root key. This function is not supported currently. This function is not supported currently.
  - HI\_UNF OTP\_LockHdcpRootKey: Locks the HDCP root key.
  - HI\_UNF OTP\_GetHdcpRootKeyLockFlag: Obtains the lock flag of the HDCP root key.
- 16-byte STB root key (root key for the anti-copy technology). For details about the usage, see the documents related to the chip anti-copy technology of HiSilicon. The following APIs are provided:
  - HI\_UNF OTP\_WriteStbRootKey: Sets the STB root key.



- HI\_UNF OTP ReadStbRootKey: Obtains the STB root key. This function is not supported currently. This function is not supported currently.
- HI\_UNF OTP LockStbRootKey: Locks the STB root key. It cannot be written after being locked.
- HI\_UNF OTP GetStbRootKeyLockFlag: Obtains the lock flag of the STB root key.

## 35.4 Development Guide

The OTP module is used in the following scenarios:

- Storing user private data
- Storing user-defined data
- Storing the HDCP root key
- Storing the STB root key

### 35.4.1 Storing User Private Data

#### Scenario

You can store 16-byte private data in the OTP area as required. The data can be written once only, and can be read.

#### Working Process

The user private data is stored as follows:

- Step 1** Initialize the OTP module by calling HI\_UNF OTP Init.  
**Step 2** Set the private data by calling HI\_UNF OTP SetStbPrivData.  
**Step 3** Deinitialize the OTP module by calling HI\_UNF OTP DeInit.
- End

#### Notes

When setting private data, you can write 1-byte data each time you call HI\_UNF OTP SetStbPrivData. You need to call HI\_UNF OTP SetStbPrivData repeatedly to write the 16-byte data. After writing the data, you can check whether the written data is correct by calling HI\_UNF OTP GetStbPrivData.

#### Sample

See `sample_otp.c`.

### 35.4.2 Storing User-Defined Data

#### Scenario

You can store 16-byte user-defined data in the OTP area as required. The data can be written once, and can be read.



## Working Process

The user-defined data is stored as follows:

- Step 1** Initialize the OTP module by calling HI\_UNF OTP\_Init.
- Step 2** Set the user-defined data by calling HI\_UNF OTP\_SetCustomerKey.
- Step 3** Deinitialize the OTP module by calling HI\_UNF OTP\_DeInit.

----End

## Notes

You can write the 16-byte data at a time. After writing the data, you can check whether the written data is correct by calling HI\_UNF OTP\_GetCustomerKey.

## Sample

See `sample_otp.c`.

### 35.4.3 Storing the HDCP Root Key

## Scenario

You need to store the HDCP root key in the OTP area, encrypt the HDCP key by using the HDCP root key, and then store the encrypted HDCP key to the flash memory. Before using the HDCP key to protect data transmitted over the HDMI, you can decrypt the HDCP key by using the HDCP root key. For details about the usage of the HDCP root key, see the *HDCP Key User Guide*.

## Working Process

The HDCP root key is stored as follows:

- Step 1** Initialize the OTP module by calling HI\_UNF OTP\_Init.
- Step 2** Set the HDCP root key by calling HI\_UNF OTP\_WriteHdcpRootKey.
- Step 3** Lock the HDCP root key by calling HI\_UNF OTP\_LockHdcpRootKey.
- Step 4** Deinitialize the OTP module by calling HI\_UNF OTP\_DeInit.

----End

## Notes

Ensure that the HDCP root key is stored according to the preceding procedure. The HDCP root key cannot be modified or read after being written. To check whether the HDCP root key is successfully burnt, call HI\_UNF OTP\_GetHdcpRootKeyLockFlag. If the obtained burn flag is 1, the HDCP root key is written. Otherwise, it is not written.

## Sample

See `sample_otp.c`.



## 35.4.4 Storing the STB Root Key

### Scenario

The OTP area can store the STB root key as the anti-copy root key. For details, see the *Anti-Copy Technology User Guide*.

### Working Process

The STB root key is stored as follows:

- Step 1** Initialize the OTP module by calling HI\_UNF OTP\_Init.
- Step 2** Set the STB root key by calling HI\_UNF OTP\_WriteStbRootKey.
- Step 3** Lock the STB root key by calling HI\_UNF OTP\_LockStbRootKey.
- Step 4** Deinitialize the OTP module by calling HI\_UNF OTP\_DeInit.

----End

### Notes

Ensure that the STB root key is stored according to the preceding procedure. The STB root key cannot be modified or read after being written. To check whether the STB root key is successfully burnt, call HI\_UNF OTP\_GetStbRootKeyLockFlag. If the obtained burn flag is 1, the STB root key is written. Otherwise, it is not written.

### Sample

See **sample\_otp.c**.



---

# Contents

---

<b>36 COMMON.....</b>	<b>1</b>
36.1 Overview .....	1
36.2 Important Concepts .....	1
36.3 Features .....	1
36.4 Development Guide.....	3



## Figures

<b>Figure 36-1</b> Process for reading a flash partition.....	3
<b>Figure 36-2</b> Process for erasing a flash partition .....	4
<b>Figure 36-3</b> Process for writing to a flash partition .....	5
<b>Figure 36-4</b> Process for obtaining information about a flash partition .....	6



# 36 COMMON

## 36.1 Overview

The COMMON module is a basic module of the SDK. It provides APIs for other modules of the SDK.

The COMMON module has the following functions:

- Initializing and deinitializing the system
- Obtaining version information
- Obtaining the presentation time stamp (PTS)
- Reading, writing to, and mapping registers
- Operating the media memory zone (MMZ) or System Memory Management Unit (SMMU)
- Controlling the debugging information
- Operating the flash memory
- Others

## 36.2 Important Concepts

[MMZ]

The MMZ is a physically continuous memory module for media playback.

[SMMU]

The HI\_MMZ\_XXX series interfaces will operate SMMU when the chip supported SMMU, otherwise, operate MMZ.

## 36.3 Features

The COMMON module provides the following functions:

- Initialization and deinitialization. The following APIs are provided:
  - HI\_SYS\_Init: Initializes system basic modules.



- HI\_SYS\_DeInit: Deinitializes system basic modules.  
HI\_SYS\_Init needs to be called when applications are started, and HI\_SYS\_DeInit needs to be called when applications are stopped.
  - Version information obtaining. The following APIs are provided:
    - HI\_SYS\_GetBuildTime: Obtains the compilation time of the SDK.
    - HI\_SYS\_GetVersion: Obtains the versions of the chip and SDK.
    - HI\_SYS\_GetChipAttr: Obtains chip attributes, for example, whether the chip supports Dolby.
    - HI\_SYS\_GetChipPackageType: Obtains the package type of the chip.You can also call these APIs to obtain system information.
  - PTS obtaining. The following API is provided:  
`HI_SYS_GetTimeStampMs`: Obtains the PTS in the unit of ms.  
The obtained PTS is not affected by the system time configured by the protocols such as the NTP.
  - Register reading, writing, and mapping. The following APIs are provided:
    - HI\_SYS\_WriteRegister: Writes to a register.
    - HI\_SYS\_ReadRegister: Reads a register.
    - HI\_SYS\_MapRegister: Maps the register address.
- `HI_SYS_UnmapRegister`: Unmap the register address.
- MMZ/SMMU operations. The following APIs are provided:
    - `HI_MMZ_Malloc`: Requests an MMZ/SMMU and maps it to the user-mode virtual address.
    - `HI_MMZ_Free`: Unmaps the user-mode virtual address and releases the MMZ/SMMU.
    - `HI_MMZ_New`: Requests for an MMZ/SMMU with a specified name and obtains its MMZ/SMMU address.
    - `HI_MMZ_Delete`: Releases an MMZ/SMMU.
    - `HI_MMZ_Map`: Maps the MMZ/SMMU to a user-mode virtual address You can determine whether to cache the address.
    - `HI_MMZ_Unmap`: Unmaps the user-mode address of the MMZ/SMMU.
    - `HI_MMZ_Flush`: Flushes data from the data cache to the memory (for the cached MMZ/SMMU).
    - `HI_MMZ_GetPhyaddr`: Obtains the MMZ/SMMU and memory size based on the virtual address.



## CAUTION

- Calling `HI_MMZ_New` returns the address of MMZ/SMMU. If the MMZ/SMMU is accessed by the CPU, you need to call `HI_MMZ_Map` to map a virtual address.
- The SMMU must use the `HI_MMZ_Map` function for mapping. If mapping is performed by using the `HI_MMZ_Map` or `mmap` function, suspensions or other exceptions may occur.



- When the SMMU uses the HI\_MMZ\_Map function for mapping, the start address of the SMMU must be input.
- Debugging information control. The following APIs are provided:
  - HI\_SYS\_SetLogLevel: Sets the log display level of a module.
  - HI\_SYS\_SetLogPath: Sets the log storage path of a module.
  - HI\_SYS\_SetStorePath: Sets the path for storing recorded files, such as the streams recorded by the DEMUX module.
- Flash memory operations. The following APIs are provided:
  - HI\_Flash\_Open: Opens a flash operation handle based on different parameters. An operation handle needs to be opened before reading, writing to, or erasing the flash memory.
  - HI\_Flash\_OpenByName: Opens a flash operation handle based on the partition name.
  - HI\_Flash\_OpenByTypeAndName: Opens a flash operation handle based on the flash type and partition name.
  - HI\_Flash\_OpenByTypeAndAddr: Opens a flash operation handle based on the flash type and address.
  - HI\_Flash\_Close: Closes a flash operation handle.
  - HI\_Flash\_Erase: Erases the flash memory. Note that the address and length must be block-aligned.
  - HI\_Flash\_Read: Reads the flash memory. Note that the address and length must be page-aligned.
  - HI\_Flash\_Write: Writes to the flash memory. Note that the address and length must be page-aligned.
  - HI\_Flash\_GetInfo: Obtains information such as the page size and block size.
- For details about the APIs of the flash memory, see the header file **hi\_flash.h**.
- Others.
  - HI\_SYS\_CRC32: Obtains the 32-byte CRC value of the data with specified length.

## 36.4 Development Guide

The invocation of APIs provided by the COMMON module is simple. This section focuses on the use of the HiFlash module.

### Scenario

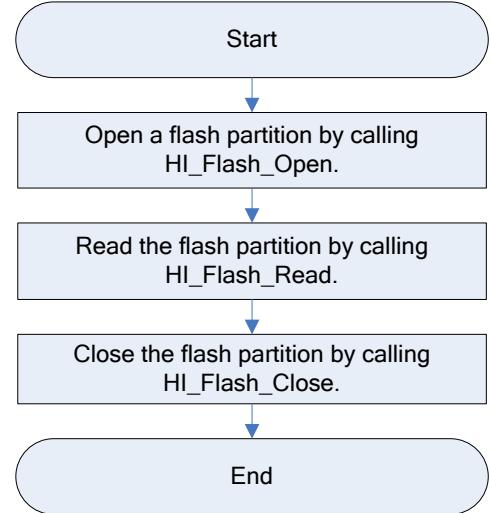
The HiFlash module provides APIs for upper-layer users to operate the flash memory (SPI flash, NAND flash, or eMMC).

### Working Process

[Figure 36-1](#) shows the process for reading a flash partition.



**Figure 36-1** Process for reading a flash partition



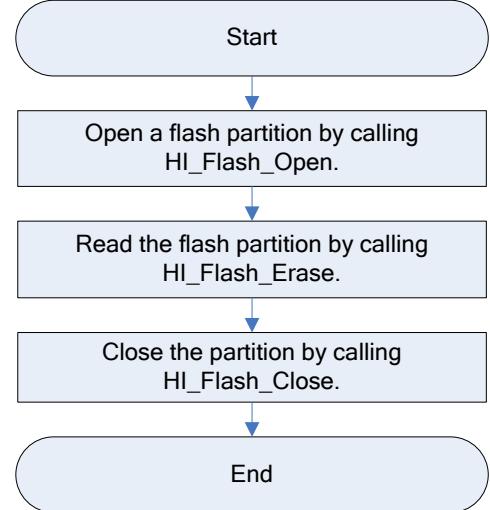
The process is as follows:

- Step 1** Open a flash partition by calling HI\_Flash\_Open and obtain the handle.
- Step 2** Read the offset data in the specified partition to the buffer by calling HI\_Flash\_Read.
- Step 3** Close the partition by calling HI\_Flash\_Close.

----End

**Figure 36-2** shows the process for erasing a flash partition.

**Figure 36-2** Process for erasing a flash partition



The process is as follows:

- Step 1** Open a flash partition by calling HI\_Flash\_Open and obtain the handle.



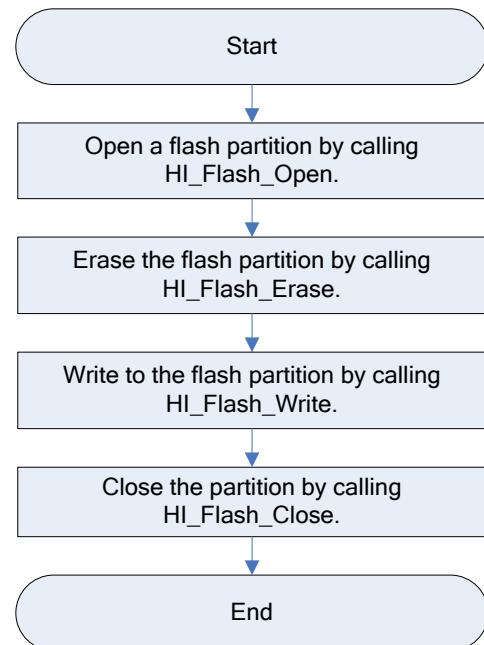
**Step 2** Erase the offset data in the specified partition by calling HI\_Flash\_Erase.

**Step 3** Close the partition by calling HI\_Flash\_Close.

----End

[Figure 36-3](#) shows the process for writing to a flash partition.

**Figure 36-3** Process for writing to a flash partition



The process is as follows:

**Step 4** Open a flash partition by calling HI\_Flash\_Open and obtain the handle.

**Step 5** Erase the offset data in the specified partition by calling HI\_Flash\_Erase.

**Step 6** Write buffer data into the offset position in the specified partition by calling HI\_Flash\_Write.

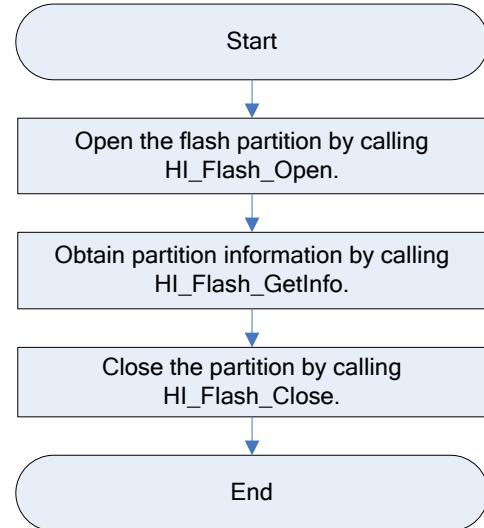
**Step 7** Close the partition by calling HI\_Flash\_Close.

—End

[Figure 36-4](#) shows the process for obtaining information about a flash partition.



**Figure 36-4** Process for obtaining information about a flash partition



The process is as follows:

**Step 8** Open a flash partition by calling `HI_Flash_Open` and obtain the handle.

**Step 9** Obtain the partition information by calling `HI_Flash_GetInfo`.

**Step 10** Close the partition by calling `HI_Flash_Close`.

----End

## Notes

- To open the flash partition, you can also call `HI_Flash_OpenByName`, `HI_Flash_OpenByTypeAndName`, or `HI_Flash_OpenByTypeAndAddr`. For details, see the sample.
- For the SPI flash and NAND flash, you need to erase the flash memory before writing data to it.

## Sample

See `sample/flash/sample_flash.c`.



# Contents

---

<b>37 GPU.....</b>	<b>1</b>
37.1 Overview .....	1
37.2 Important Concepts .....	2
37.3 Function Description .....	7
37.4 Development Guide.....	8
37.4.1 Development Process of the FBDEV EGL .....	8
37.5 OpenGL ES .....	9
37.6 OpenCL .....	10
37.7 3D UI Engine .....	10
37.8 FAQs .....	10
37.8.1 Utgard GPU Platform FAQs .....	10
37.8.2 Midgard GPU Platform FAQ .....	12



# Figures

Figure 37-1 GPU standards in the system ..... 2



# Tables

Table 37-1 Definitions of the display, window, and pixmap in different window systems .....	3
Table 37-2 flags and data in the data structure fbdev_pixmap .....	4
Table 37-3 Data members in the data structure egl_linux_pixmap .....	5



# 37 GPU

## 37.1 Overview

The graphics processing unit (GPU) is a 3D graphics processing unit provided by HiSilicon. It provides the standard OpenGL ES 1.1/2.0/3.0/3.1, OpenVG 1.1, OpenCL1.1, and EGL 1.4 interfaces that comply with industry standards.

- As a 3D graphics library in the embedded system, the OpenGL ES is customized and developed by the Khronos Group based on the desktop OpenGL standard.
  - With the fixed rendering pipeline, OpenGL ES 1.1 is intended for low-end embedded devices.
  - With the programmable rendering pipeline, OpenGL ES 2.0 is intended for high-end embedded devices.
  - Forward compatible with OpenGL ES 2.0, OpenGL ES 3.0 provides more powerful shading functions and supports more texture formats.
  - Based on OpenGL ES 3.0, OpenGL ES 3.1 provides new features such as the compute shader for general-purpose computing tasks, separate shader (vertex shader and fragment shader) objects, and indirectly obtaining drawing commands from the memory.

In August 2013, the Khronos Group released the OpenGL ES 3.0 standard with new functions that allows the development of more dazzling effects and higher performance. OpenGL ES 3.X is compatible with OpenGL ES 2.0. Programs developed based on OpenGL ES 3.X can run on OpenGL ES 2.0. However, OpenGL ES 2.0 is incompatible with OpenGL ES 1.1.

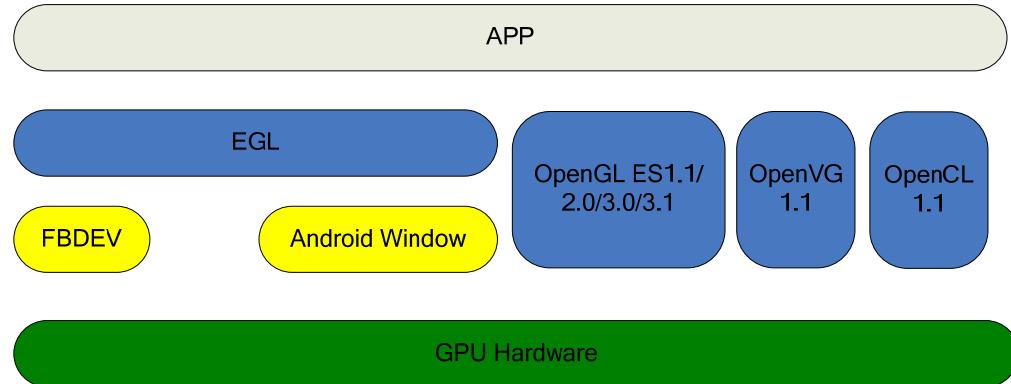
- As a 2D vector graphics library, the OpenVG has been marginalized because all its functions can be replaced by the functions of OpenGL ES 2.0.
- As the programming interface for general-purpose computing, OpenCL 1.1 distributes the computing tasks to multiple operation units in the GPU module to improve the computing efficiency.
- The EGL APIs are between the drawing APIs, such as OpenGL ES and OpenVG, and the bottom-layer window system. OpenGL ES or OpenVG is a state machine of the graphics rendering pipeline, and the EGL maintains the states and transmits the rendering results to the corresponding windows or surfaces.

For details about the standard specifications of the preceding standards, go to <http://www.khronos.org/>.



[Figure 37-1](#) shows the position of the GPU standards in the system. This section describes only the usage of the EGL. OpenGL ES 1.1/2.0/3.0/3.1, OpenCL 1.1, and OpenVG are standard drawing interfaces. For details, see the development guide provided by ARM.

**Figure 37-1** GPU standards in the system



The HiSilicon EGL can connect to the FBDEV and Android window system. 3D contents rendered by the OpenGL ES can be transmitted to the windows or surfaces of the preceding window systems for display.

As the EGL is related to the platform, this section only describes its usage. OpenGL ES, OpenCL, and OpenVG APIs are standard APIs. You can find many related materials on the network, and this section does not provide the details.

#### **NOTE**

Currently, only Hi3798C V200\_A supports the OpenGL ES 3.x and OpenCL. Other chips support only the OpenGL ES 1.1/2.0 and OpenVG 1.1 interfaces.

## 37.2 Important Concepts

[Window surface]

The window surface is the surface that can be displayed on the screen in the EGL. It is created by EGLNativeWindowType.

[Pixmap surface]

The pixmap surface is an off-screen surface in the EGL. It is obtained by encapsulating the EGLNativePixmapType.

[Pbuffer surface]

The pbuffer surface is an off-screen surface in the EGL that is not related to the platform. It is rendered as textures or used to draw a surface larger than the window.

[EGLNativeDisplayType]

EGLNativeDisplayType is an abstract of the local display device. The EGL needs to create the corresponding EGLDisplay based on the local display device.

[EGLNativeWindowType]



EGLNativeWindowType is an abstract of the local window. The EGL needs to create the corresponding window surface for display on the screen based on the local window.

[EGLNativePixmapType]

EGLNativePixmapType is an abstract of the local off-screen buffer. The EGL needs to create the corresponding pixmap surface for off-screen drawing based on the local off-screen buffer.

**Table 37-1** Definitions of the display, window, and pixmap in different window systems

Object	FBDEV	Remarks
EGLNativeDisplayType	0: /dev/fb0 is used as the display device. 2: /dev/fb2 is used as the display device.	/dev/fb0 is used as the default display device.
EGLNativeWindowType	typedef struct fbdev_window { • unsigned short width; • unsigned short height; } fbdev_window;	width: window width height: window height
EGLNativePixmapType	typedef struct fbdev_pixmap { • unsigned int height; • unsigned int width; • unsigned int bytes_per_pixel; • unsigned char buffer_size; • unsigned char red_size; • unsigned char green_size; • unsigned char blue_size; • unsigned char alpha_size; • unsigned char luminance_size; • fbdev_pixmap_flags flags; • unsigned short *data; • unsigned int format; } fbdev_pixmap;	This structure applies to the Hi3716C V200/Hi3718C V100/Hi3719M V100/Hi3719C V100/Hi3798M V100/Hi3796M V100/Hi3798C V100/Hi3796C V100 chips.



Object	FBDEV	Remarks
EGLNativePixmapType	<pre>typedef struct egl_linux_pixmap {     int width, height;     struct     {         khronos_usize_t         stride;         khronos_usize_t         size;         khronos_usize_t         offset;     }planes[3];     uint64_t     pixmap_format;     mem_handle     handles[3]; } egl_linux_pixmap;</pre>	This structure applies to the Hi3798C V200_A chip.

Table 37-2 describes the members **flags** and **data** in the data structure fbdevPixmap.

**Table 37-2** flags and data in the data structure fbdevPixmap

Flags	Data	Description
FBDEV_PIXMAP_DEFAULT	Memory pointer	The pixmap memory is the system memory.
FBDEV_PIXMAP_PHYADDR	<pre>fbdevPixmap_addr data structure pointer typedef struct fbdevPixmap_addr {     void *viraddr;     unsigned int     phyaddr;     unsigned int     memsize; } fbdevPixmap_addr;</pre>	The pixmap memory is a physical memory, for example, the memory allocated by the MMZ.
FBDEV_PIXMAP_EGL_MEMORY	NULL	The pixmap memory is allocated and released within the EGL.
FBDEV_PIXMAP_SUPPORTS_UMP	-	The pixmap memory is UMP memory and is not supported currently.



Flags	Data	Description
FBDEV_PIXMAP_DMA_BUF	-	The pixmap memory is in DMA buffer mode, which is not supported currently.
FBDEV_PIXMAP_ALPHA_FORMAT_PRE	-	The pixmap contains alpha premultiply information.
FBDEV_PIXMAP_COLORSPACE_sRGB	-	The pixmap color space is sRGB.

#### NOTE

The physical memory is recommended for improving performance.

[Table 37-3](#) describes the data members in the data structure egl\_linuxPixmap.

**Table 37-3** Data members in the data structure egl\_linuxPixmap

Data	Description
width	Surface width
height	Surface height
planes[3]	Memory allocation of the pixmap: <ul style="list-style-type: none"><li>For the ARGB format, there is only one plane. Therefore, only <b>planes[0]</b> is valid, and <b>handles[0]</b> describes the memory objects.</li><li>For the YUV semi-planar format, there are two planes. <b>planes[0]</b> describes the memory allocation for the Y component, and <b>handles[0]</b> is the pointer to the memory objects. <b>planes[1]</b> describes the memory allocation for the UV component, and <b>handles[1]</b> is the pointer to the memory objects.</li><li>For the YUV planar format, there are three planes. <b>planes[0]</b> describes the memory allocation for the Y component, and <b>handles[0]</b> is the pointer to the memory objects. <b>planes[1]</b> describes the memory allocation for the U component, and <b>handles[1]</b> is the pointer to the memory objects. <b>planes[2]</b> describes the memory allocation for the V component, and <b>handles[2]</b> is the pointer to the memory objects.</li><li>For each plane:<ul style="list-style-type: none"><li><b>stride</b> indicates the line spacing of each component.</li><li><b>size</b> indicates the component memory size.</li><li><b>offset</b> indicates the offset of each component in the memory.</li></ul></li></ul>
pixmap_format	Pixel formats supported: <ul style="list-style-type: none"><li>16-bit RGB pixel formats:<ul style="list-style-type: none"><li>EGL_COLOR_BUFFER_FORMAT_BGR565</li><li>EGL_COLOR_BUFFER_FORMAT_RGB565</li></ul></li></ul>



Data	Description
	<p>EGL_COLOR_BUFFER_FORMAT_ABGR4444 EGL_COLOR_BUFFER_FORMAT_ARGB4444 EGL_COLOR_BUFFER_FORMAT_BGRA4444 EGL_COLOR_BUFFER_FORMAT_RGBA4444 EGL_COLOR_BUFFER_FORMAT_ABGR1555 EGL_COLOR_BUFFER_FORMAT_ARGB1555 EGL_COLOR_BUFFER_FORMAT_BGRA5551 EGL_COLOR_BUFFER_FORMAT_RGBA5551</p> <ul style="list-style-type: none"><li>• 24-bit RGB pixel formats: EGL_COLOR_BUFFER_FORMAT_BGR888 EGL_COLOR_BUFFER_FORMAT_RGB888</li><li>• 32-bit RGB pixel formats: EGL_COLOR_BUFFER_FORMAT_ABGR8888 EGL_COLOR_BUFFER_FORMAT_sABGR8888 EGL_COLOR_BUFFER_FORMAT_ARGB8888 EGL_COLOR_BUFFER_FORMAT_BGRA8888 EGL_COLOR_BUFFER_FORMAT_RGBA8888 EGL_COLOR_BUFFER_FORMAT_XBGR8888 EGL_COLOR_BUFFER_FORMAT_sXBGR8888 EGL_COLOR_BUFFER_FORMAT_XRGB8888 EGL_COLOR_BUFFER_FORMAT_BGRX8888 EGL_COLOR_BUFFER_FORMAT_RGBX8888</li><li>• 8-bit luminance component: EGL_COLOR_BUFFER_FORMAT_L8</li><li>• YV12 format, color space conversion (CSC) in BT.601/BT.709 narrow/wide mode EGL_COLOR_BUFFER_FORMAT_YV12_BT601_NARROW EGL_COLOR_BUFFER_FORMAT_YV12_BT601_WIDE EGL_COLOR_BUFFER_FORMAT_YV12_BT709_NARROW EGL_COLOR_BUFFER_FORMAT_YV12_BT709_WIDE</li><li>• NV12 format, CSC in BT.601/BT.709 narrow/wide mode EGL_COLOR_BUFFER_FORMAT_NV12_BT601_NARROW EGL_COLOR_BUFFER_FORMAT_NV12_BT601_WIDE EGL_COLOR_BUFFER_FORMAT_NV12_BT709_NARROW EGL_COLOR_BUFFER_FORMAT_NV12_BT709_WIDE</li><li>• YUYV format, CSC in BT.601/BT.709 narrow/wide mode EGL_COLOR_BUFFER_FORMAT_YUYV_BT601_NARROW EGL_COLOR_BUFFER_FORMAT_YUYV_BT601_WIDE EGL_COLOR_BUFFER_FORMAT_YUYV_BT709_NARROW EGL_COLOR_BUFFER_FORMAT_YUYV_BT709_WIDE</li></ul>



Data	Description
	<ul style="list-style-type: none"><li>• NV21 format, CSC in BT.601/BT.709 narrow/wide mode<ul style="list-style-type: none"><li>EGL_COLOR_BUFFER_FORMAT_NV21_BT601_NARROW</li><li>EGL_COLOR_BUFFER_FORMAT_NV21_BT601_WIDE</li><li>EGL_COLOR_BUFFER_FORMAT_NV21_BT709_NARROW</li><li>EGL_COLOR_BUFFER_FORMAT_NV21_BT709_WIDE</li></ul></li></ul>
handles[3]	Pixmap memory handle

## 37.3 Function Description

The EGL module provides the following functions:

- Initialization and deinitialization. The following APIs are provided:
  - eglInitialize: Initializes the EGL.
  - eglTerminate: Terminates the EGL.
  - eglGetDisplay: Obtains the display device.
- Configuration management. The following APIs are provided:
  - eglGetConfigs: Obtains the configuration list of frame buffers supported by the display device.
  - eglGetConfigAttrib: Obtains information in the display frame buffer configuration.
  - eglChooseConfig: Selects a list of frame buffers that meet the specified configuration requirements.
- Drawing surface management. The following APIs are provided:
  - eglCreateWindowSurface: Creates a window surface.
  - eglCreatePbufferSurface: Creates an off-screen pbuffer surface.
  - eglCreatePixmapSurface: Creates an off-screen pixmap surface.
  - eglDestroySurface: Destroys an EGL surface.
  - eglQuerySurface: Queries surface attributes.
- Drawing context management. The following APIs are provided:
  - eglCreateContext: Creates a context.
  - eglDestroyContext: Destroys a context.
  - eglMakeCurrent: Binds the current context.
  - eglGetCurrentContext: Obtains the current context.
  - eglGetCurrentSurface: Obtains the current read or write surface.
  - eglGetCurrentDisplay: Obtains the current display.
  - eglQueryContext: Queries context information.
- Buffer deliver display. The following APIs are provided:
  - eglSwapBuffers: Transmits the surface contents to the local window for display.
  - eglCopyBuffers: Copies the surface contents to the local pixmap.
- EGL extended functions. The following APIs are provided:



- eglCreateImageKHR: Creates an EGL image.
- eglDestroyImageKHR: Destroys an EGL image.

## 37.4 Development Guide

Because the EGL masks the differences between local windows, you only need to create a local display system and a window; then the 3D effect can be rendered to the local window. The following describes the EGL development process based on the FB window system.

### 37.4.1 Development Process of the FBDEV EGL

#### Scenario

In the FEDEV EGL scenario, the current system does not have the corresponding window system. Drawing is directly based on the frame buffer, and the surface overlapping process is managed by the user.

#### Working Process

The development process is as follows:

**Step 1** Select the local display device. **0** corresponds to **/dev/fb0**, and **2** corresponds to **/dev/fb2**. For details about the device supported by the frame buffer, see the description about the FB driver. Select **/dev/fb0** in this sample.

```
int nativeDisplay = 0;
```

**Step 2** Create a local window. Assume that the resolution is 1280 x 720.

```
fbdev_window * nativeWindow = malloc( sizeof( fbdev_window ) );
if ( NULL == fbwin )
{
    return 0;
}

nativeWindow ->width = 1280;
nativeWindow ->height = 720;
```

**Step 3** Obtain the corresponding handle of the EGL display based on the handle of the local display device.

```
EGLDisplay eglDisplay = eglGetDisplay(nativeDisplay);
```

**Step 4** Initialize the EGL.

```
eglInitialize(eglDisplay, NULL, NULL);
```

**Step 5** Select the corresponding configuration.

```
eglChooseConfig(eglDisplay, configAttrs, &configs[0], 10,
&matchingConfigs);
```



**Step 6** Create an EGL surface.

```
EGLSurface eglSurface = eglCreateWindowSurface(eglDisplay, configs[0],  
nativeWindow, configAttribs);
```

**Step 7** Create an EGL context.

```
EGLContext eglContext = eglCreateContext(eglDisplay, configs[0], NULL,  
ctxAttribs);
```

**Step 8** Set the current context.

```
eglMakeCurrent(eglDisplay, eglSurface, eglSurface, eglContext);
```

Then, the EGL is initialized and you can start drawing graphics. After drawing graphics, you need to output the drawn graphics to the screen by calling `eglSwapBuffers`. In the preceding procedure, only steps 2 and 3 are related to the local window system. The interfaces used in other steps are standard EGL interfaces. For details about these interfaces, go to <http://www.khronos.org/>.

----End

The EGL is destroyed as follows:

**Step 1** Set the current context to **NULL**.

```
eglMakeCurrent(eglDisplay, NULL, NULL, NULL);
```

**Step 2** Destroy the context.

```
eglDestroyContext(eglDisplay, eglContext);
```

**Step 3** Destroy the eglSurface.

```
eglDestroySurface(eglDisplay, eglSurface);
```

**Step 4** Terminate the EGL.

```
eglTerminate(g_EglDisplay);
```

**Step 5** Destroy the local window.

```
free(nativeWindow);
```

----End

## Sample

See `egl_native.c` and `hi_egl.c` in `sample/gpu/src/common/egl/src`.

## 37.5 OpenGL ES

For details about the development and optimization of OpenGL ES, see the *Mali™ GPU OpenGL ES Application Development Guide* provided by the ARM at



<http://malideveloper.arm.com/develop-for-mali/tutorials-developer-guides/developer-guides/mali-gpu-opengl-es-application-development-guide/>.

Chapter 2 in this document elaborates the development procedure by using the OpenGL ES.

## 37.6 OpenCL

For details about the development and optimization of OpenCL, see the *ARM® Mali™-T600 Series GPU OpenCL Developer Guide* provided by the ARM at <http://malideveloper.arm.com/develop-for-mali/tutorials-developer-guides/developer-guides/mali-t600-series-gpu-opencl-developer-guide/>.

Chapter 5 in this document elaborates the development procedure by using the OpenCL.



### NOTE

The architectures of the Mali T700 series and Mali T600 series are the same, and the development guide to the Mali T600 series applies to the Mali T700 series.

## 37.7 3D UI Engine

The ARM provides an open-source 3D UI engine library for users. You can go to the following website:

<http://malideveloper.arm.com/develop-for-mali/tools/mali-gpu-user-interface-engine/>

## 37.8 FAQs

### 37.8.1 Utgard GPU Platform FAQs



### NOTE

The Mali 400 and Mali 450 uses the same Utgard structure. This chapter applies to the Hi3716C V200/Hi3718C V100/Hi3719M V100/Hi3719C V100/Hi3798M V100/Hi3796M V100/Hi3798C V100/Hi3796C V100 chips.

#### 37.8.1.1 How Do I Set the Maximum Memory for the GPU Driver?

Run **make menuconfig** in the SDK root directory, choose **Msp > Gpu Config > Mali400 and OS maximal shared memory size**, and set the maximum shared memory size, which is 128 MB in the current Linux platform and 256 MB in the Android platform by default. You are advised to set this value a bit smaller than the OS memory as required. Because the GPU only requests for the OS memory only when it requires the memory, you can set the GPU memory to a large value possible.

#### 37.8.1.2 How Do I Compile the Debug Version Driver for Debugging?

Run **make menuconfig** in the SDK root directory, and choose **Msp > Gpu Config > Enable debug in Mali driver**. Recompile the kernel. The default display level of the debug version is 2, and you can adjust the level. For example, to change the level to 3, run the following command:



```
echo 3 >/sys/module/mali/parameters/mali_debug_level
```

### 37.8.1.3 How Do I Enable or Disable DVFS?

The dynamic voltage and frequency scaling (DVFS) is enabled by default. To disable the DVFS, run the following command:

```
echo 0 >/sys/module/mali/parameters/mali_dvfs_enable
```

To enable the DVFS, run the following command:

```
echo 1 >/sys/module/mali/parameters/mali_dvfs_enable
```

### 37.8.1.4 How Do I Check the GPU Memory Usage on Android?

The GPU memory can be divided into two parts: memory occupied by kernel-mode drivers and surface memory allocated by the user-mode Gralloc. The Gralloc on Android uses the standard memory management module ION to manage the memory.

You can run the following commands to check the memory usage:

- mount -t debugfs debugfs /sys/kernel/debug
- cat /sys/kernel/debug/mali/memory\_usage: Check the memory consumed by GPU kernel drivers.
- cat /sys/kernel/debug/ion/ddr: Check the ION memory consumed by the GPU.
- cat /sys/kernel/debug/ion/<<pid>>: Check the ION memory consumed by a process.

### 37.8.1.5 How Do I View the GPU Running Information Such As the Clock Frequency, Voltage, and Usage?

The GPU running information can be obtained in real time, including the clock frequency, voltage, GP/PP usage, power status, size of the OS shared memory, start address and size of the physical memory allocated for the GPU, DVFS status, and AVS status.

Run **cat /proc/msp/pm\_gpu** in the command-line interface.

Information similar to the following is displayed:

```
-----Hisilicon GPU Info-----  
Frequency : 0 (kHz)  
Voltage : 1300 (mv)  
GP_utilization : 0 (percent)  
PP_utilization : 0 (percent)  
Power_status : power down  
Shared_mem_size : 256 (MB)  
Dedicated_mem_start : 0x0  
Dedicated_mem_size : 0 (MB)  
DVFS_enable : enable  
AVS_enable : disable
```



## 37.8.2 Midgard GPU Platform FAQ

### NOTE

The Mali T720 belongs to the Utgard structure, and this chapter applies to the Hi3798C V200\_A chip.

### 37.8.2.1 How Do I View the GPU Running Information Such As the Clock Frequency, Voltage, and Usage?

The proc information supports the check of GPU voltage, working frequency, and usage by running the following command:

```
cat /proc/msp/pm_gpu
```

The following result is displayed:

```
-----Hisilicon GPU Info-----
Frequency : 198000 (kHz)
Voltage   : 850 (mv)

Utilization : 0(%)
```



# Contents

---

<b>38 VP .....</b>	<b>1</b>
38.1 Overview .....	1
38.2 Important Concepts .....	1
38.3 Function Description .....	2
38.3.1 Features .....	2
38.3.2 Function Implementation .....	3
38.4 Development Guide.....	5
38.4.1 Video Phone with a USB Camera .....	5



## Figures

<b>Figure 38-1</b> VP module among modules related to video processing .....	1
<b>Figure 38-2</b> Capturing MJPEG video data .....	4
<b>Figure 38-3</b> Capturing YUV video data .....	5



## Tables

<b>Table 38-1</b> VP attribute parameters and default configurations.....	3
---	---



# 38 VP

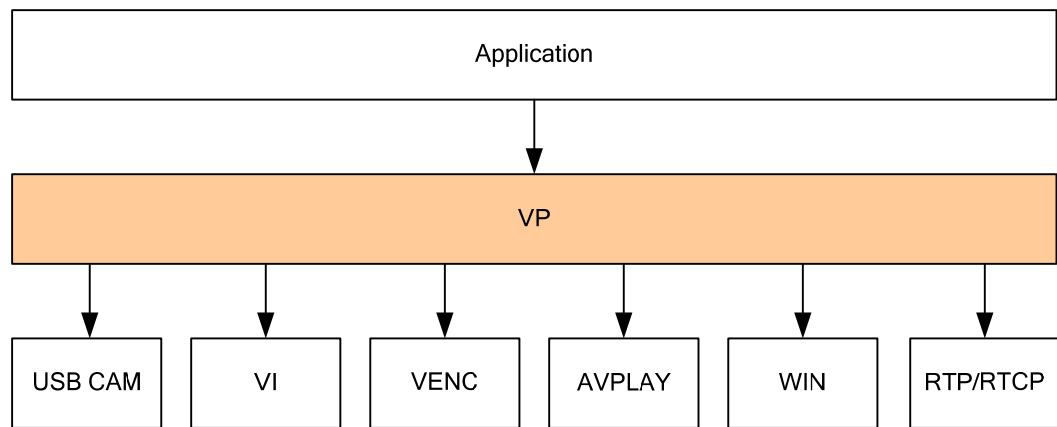
## 38.1 Overview

The video phone (VP) module provides the video communication function, and the voice over Internet Protocol (VoIP) module provides the voice communication function. The VP and VoIP modules work together to provide the video phone functions.

The VP module is a component which relies on video-related modules, including the VI, VENC, AVPLAY, and WIN modules. It captures and encodes video data, displays local videos, transmits and receives streams, decodes videos, and displays remote videos, implementing end-to-end video communication.

[Figure 38-1](#) shows the VP module among modules related to video processing.

**Figure 38-1** VP module among modules related to video processing



## 38.2 Important Concepts

[Video input source]

The video input source indicates the video data capturing mode of the video phone, which is the USB camera. The USB camera input source is divided into the YUV data source and



MJPEG data source based on whether compressed encoding is implemented on the captured data.

[Local video]

The local video is the video of your own end during video communication. Video data is captured and directly transmitted to the local window for display without encoding.

[Remote video]

The remote video is the video of the peer end during video communication. Video data is captured, encoded, and transmitted to the network by the peer end, and then the data is received, decoded, and displayed by the local end.

## 38.3 Function Description

### 38.3.1 Features

The VP module provides the following functions:

- Video phone management, such as creating and destroying the VP, obtaining and setting the VP attributes, binding and unbinding the video input source and local and remote windows. The following APIs are provided:
  - HI\_UNF\_VP\_GetDefaultAttr: Obtains default VP attributes.
  - HI\_UNF\_VP\_Create: Creates a VP.
  - HI\_UNF\_VP\_Destroy: Destroys a VP.
  - HI\_UNF\_VP\_GetAttr: Obtains VP attributes.
  - HI\_UNF\_VP\_SetAttr: Sets VP attributes.
  - HI\_UNF\_VP\_AttachSource: Binds the VP to a video input source.
  - HI\_UNF\_VP\_DetachSource: Unbinds the VP from a video input source.
  - HI\_UNF\_VP\_AttachLocalWin: Binds the VP to a local window.
  - HI\_UNF\_VP\_DetachLocalWin: Unbinds the VP from a local window.
  - HI\_UNF\_VP\_AttachRemoteWin: Binds the VP to a remote window.
  - HI\_UNF\_VP\_DetachRemoteWin: Unbinds the VP from a remote window.
- Video phone control, including starting and stopping the VP. The following APIs are provided:
  - HI\_UNF\_VP\_Start: Starts the VP.
  - HI\_UNF\_VP\_Stop: Stops the VP.
  - HI\_UNF\_VP\_StartPreView: Starts VP previewing.
  - HI\_UNF\_VP\_StopPreView: Stops VP previewing.
- Network parameter configuration and network statistics collection. The following APIs are provided:
  - HI\_UNF\_VP\_ConfigRtcp: Sets RTCP network attributes.
  - HI\_UNF\_VP\_GetNetStatics: Obtains network statistics.

Table 38-1 describes the attribute parameters and default configurations of the VP module.

**Table 38-1** VP attribute parameters and default configurations

Attribute	Default Configuration	Description
stVencAttr	stVencAttr.enVencType=HI_UNF_VCO DEC_TYPE_H264; stVencAttr.enCapLevel= HI_UNF_VCODEC_CAP_LEVEL_D1 stVencAttr.u32Width= 640 stVencAttr.u32Height= 480 stVencAttr.u32InputFrmRat= 30 stVencAttr.u32TargetFrmRate= 30 stVencAttr.u32TargetBitRate= 3 / 2 * 1024 * 1024 stVencAttr.u32StrmBufSize= 640 * 480 * 2 stVencAttr.u32Gop = 60 stVencAttr.bSlcSplitEn = HI_FALSE stVencAttr.bQuickEncode = HI_TRUE	Video encoding attributes Only the H.264 encoding type is supported in the video phone service. The encoding resolution, bit rate, and size of the stream buffer are configured by using the VGA as an example. For details about the configuration of other resolution, see <b>sample_vp_usbcam.c</b> .
enDecType	HI_UNF_VCODEC_TYPE_H264	Video decoding attributes Only the H.264 decoding type is supported in the video phone service.
bIsIpv4	HI_TRUE	Whether to use the IPv4 parameters. HI_TRUE: IPv4 HI_FALSE: IPv6
stLocalNetAttr	None	Local network parameters
stRemoteNetAttr	None	Remote network parameters
stRtpAttr	stRtpAttr.u32SplitSize= 1450; stRtpAttr.u32PayLoadType = RTP_PT_H264; stRtpAttr.u32RecvBufSize = RTP_MAX_UDP_LEN; stRtpAttr.u32RecvBufNum= RTP_REORDER_BUF_NUM; stRtpAttr.u32SortDepth = 5;	RTP attributes

### 38.3.2 Function Implementation

The VP module relies on the VI, VENC, AVPLAY, and WIN modules. The captured video data is transmitted to the local window for display, and it is also transmitted to the VENC for encoding. The encoded H.264 streams are transmitted to the network remote end over the RTP



protocol. Then the remote end receives the video, sends it to the AVPLAY for decoding, and then displays the video on the remote window.

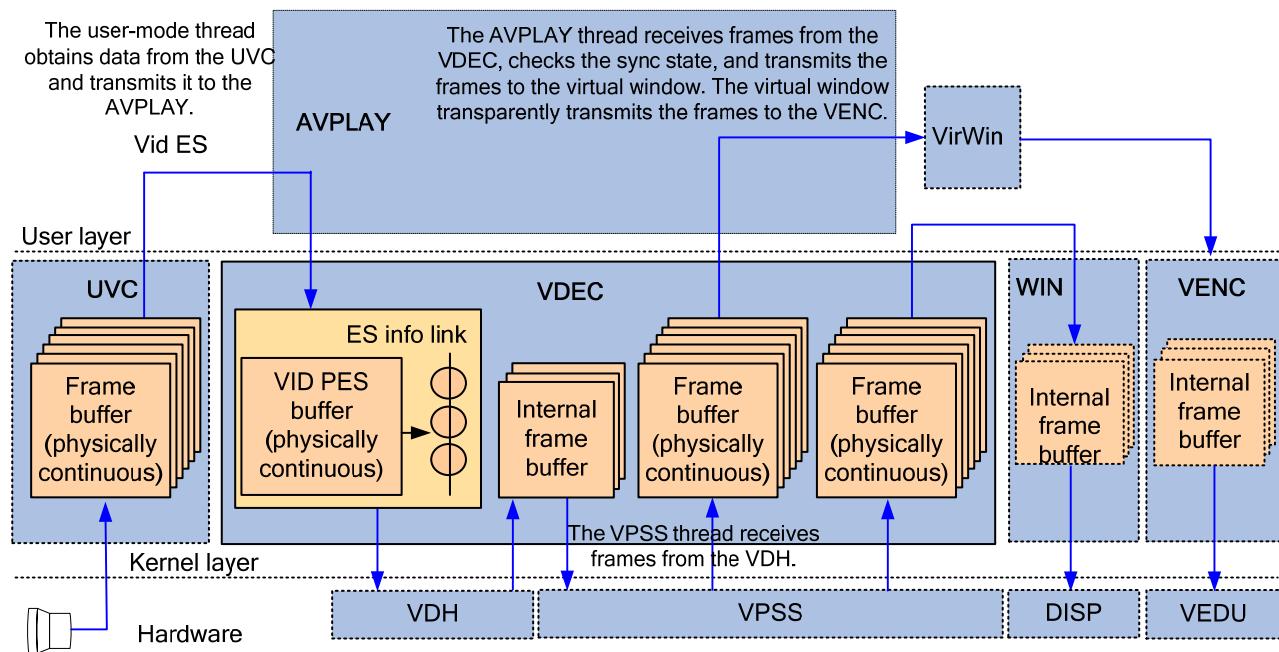
This section describes the working principles of the VP module with two different input sources. The focus is the process from video capturing until network transmission.

### 38.3.2.1 MJPEG Data As the Input Source

High-end USB cameras typically support the MJPEG data source, which applies to HD video phone scenarios with 720p or higher resolution. Compared with the YUV data source, the MJPEG data source provides significantly fewer data amount but higher resolution.

The USB camera captures MJPEG streams and transmits the streams to the AVPLAY module for decoding. An entity window is bound to the AVPLAY for local video display and a virtual window is bound to the VENC for H.264 video encoding.

**Figure 38-2** Capturing MJPEG video data



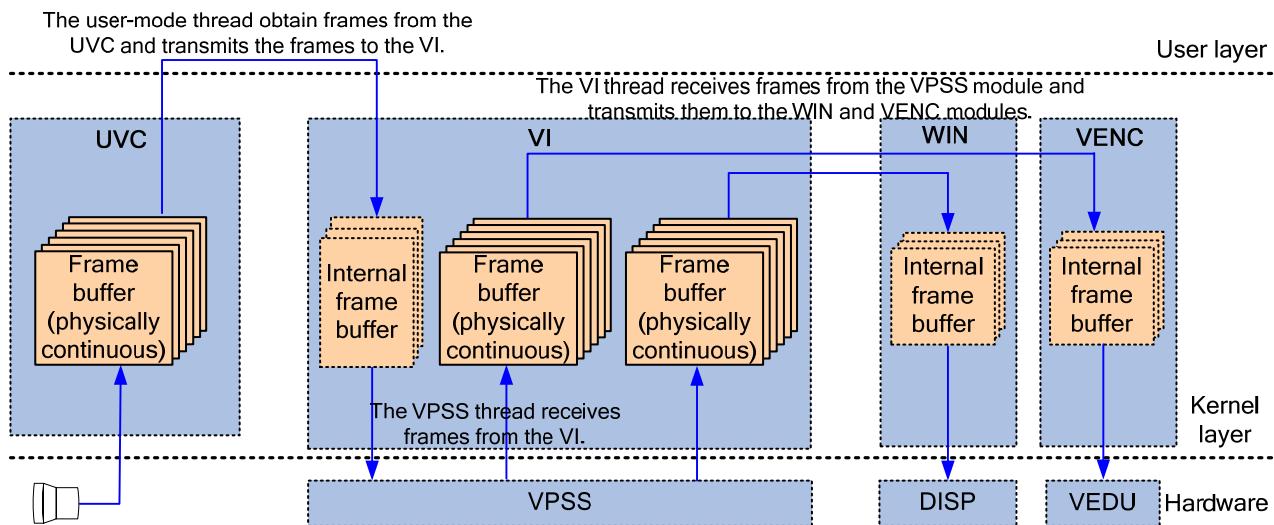
### 38.3.2.2 YUV Data As the Input Source

Common USB cameras support the YUV data source, which has huge data amount and applies to video phone scenarios with 720p or lower resolution. The VGA resolution is recommended.

The USB camera captures YUV data and transmits the data to the VI module. An entity window is bound for local video display and a VENC is bound for H.264 video encoding.



**Figure 38-3** Capturing YUV video data



## 38.4 Development Guide

The VP module is used in the following scenario:

### 38.4.1 Video Phone with a USB Camera

#### Scenario

The video phone with a USB camera captures video data from the USB camera to implement the VP service. YUV or MJPEG data can be captured based on whether the video data is encoded.

#### Working Process

To capture YUV or MJPEG data by using a USB camera in the video phone scenario, perform the following steps:

- Step 1** Initialize the VP module and related devices, such as the VI, VENC, AVPLAY, RTP, and WIN modules.
- Step 2** Create a local window and a remote window for displaying the video.
- Step 3** Start the USB camera, and ensure that it supports video capturing.
- Step 4** Set the resolution and data format (YUV or MJPEG) for the USB camera.
- Step 5** Set the frame rate for the USB camera.
- Step 6** Obtain default VP attributes by calling HI\_UNF\_VP\_GetDefaultAttr.
- Step 7** Set the local IP address and port number in the attributes, and create a VP by calling HI\_UNF\_VP\_Create.



- Step 8** Negotiate parameters with the SIP server, and call HI\_UNF\_VP\_SetAttr to set the encoding attributes, decoding attributes, and IP address and port ID of the peer end.
- Step 9** Set the USB camera as the video input source and input the camera handle by calling HI\_UNF\_VP\_AttachSource.
- Step 10** Bind a local window by calling HI\_UNF\_VP\_AttachLocalWin.
- Step 11** Bind a remote window by calling HI\_UNF\_VP\_AttachRemoteWin.
- Step 12** Start the VP by calling HI\_UNF\_VP\_Start.
- Step 13** Stop the VP by calling HI\_UNF\_VP\_Stop after the service is stopped.
- Step 14** Unbind the remote window by calling HI\_UNF\_VP\_DetachRemoteWin.
- Step 15** Unbind the local window by calling HI\_UNF\_VP\_DetachLocalWin.
- Step 16** Unbind the video input source by calling HI\_UNF\_VP\_DetachSource.
- Step 17** Destroy the VP by calling HI\_UNF\_VP\_Destroy.
- Step 18** Destroy the local window and the remote window.
- Step 19** Deinitialize the VP module and related devices.

----End

## Notes

Before starting the VP, you must bind the video input source, local window, and remote window.

## Sample

See [sample/vp/sample\\_vp\\_usbcam.c](#).



# Contents

---

<b>39 3G Dongle.....</b>	<b>3</b>
39.1 Overview .....	3
39.2 Important Concepts .....	4
39.3 Features .....	4
39.4 Development Guide.....	5
39.4.2 Connecting to/Disconnecting from the 3G Network.....	7
39.4.3 Obtaining Operator Information and Registering with an Operator Network.....	8



## Figures

<b>Figure 39-1</b> Typical application architecture .....	3
<b>Figure 39-2</b> Process for connecting to and disconnecting from the 3G network.....	7
<b>Figure 39-3</b> Process for obtaining operator information and registering with an operator network.....	9

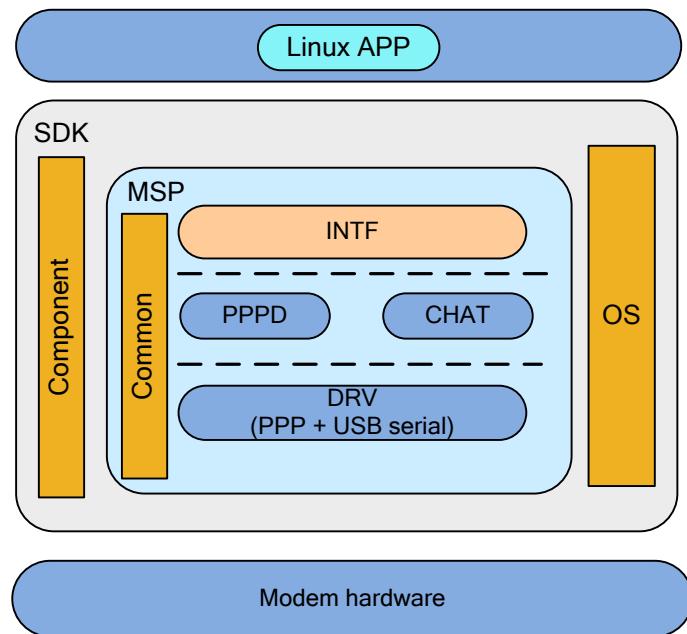


# 39 3G Dongle

## 39.1 Overview

The 3rd-generation (3G) dongle enables access to the Internet over the operator's network (2G or 3G network) by using the 3G subscriber identity module (SIM) card. It provides APIs for applications to implement corresponding functions. [Figure 39-1](#) shows the typical application architecture of the 3G dongle.

**Figure 39-1** Typical application architecture



The software architecture consists of the following three layers:

- INTF layer

The interface (INTF) layer provides APIs for Linux applications to implement functions such as connecting to or disconnecting from the 3G network.

- Tool layer



- Chat implements the dialing function. It ensures that the 3G dongle is normal and registers the card with the operator network, which enables the card to transmit data received from the serial port to the network or forward data received from the network over the serial port.
- As a daemon process, the Point-to-Point Protocol Daemon (PPPD) interacts with the Point-to-Point Protocol (PPP) of the kernel to transmit or receive data.
- DRV layer
  - The driver (DRV) layer includes the PPP layer contents and the USB serial driver.
    - The PPPD creates the network interface by using ioctl. The PPP layer processes the ioctl requests from the PPPD and registers the network interface with the kernel. After that, applications can transmit or receive data over the PPP protocol.
    - The USB serial driver transmits data from the PPP layer over the serial port, or forwards the data received from the serial port to the PPP layer.

Modem hardware indicates the 3G dongle with inserted SIM card, which implements data transmission and reception. For details, see the list of compatible components for the corresponding version.

## 39.2 Important Concepts

[3G]

Currently the third generation of the mobile telecommunications technology supports four standards: CDMA2000, WCDMA, TD-SCDMA, and WiMAX.

[AT instruction]

AT instructions are used to communicate with the modem, including configuring and obtaining the modem status.

[PC UI]

The PC UI port is dedicated for receiving AT instructions and configuring and obtaining the modem status. After the 3G dongle is inserted, multiple ttyUSB devices are generated. Some are used for dialing, some are used for voice communication, and some are used to receive AT instructions and configure and obtain the modem status. The PC UI is the device for receiving AT instructions and configuring and obtaining the modem status.

## 39.3 Features

The INTF layer of the 3G dongle module provides the following functions:

- Initializes/Deinitializes the 3G dongle module.
  - HI\_3G\_InitCard: Obtains vendor and product information and dialing port and PC UI port, and sets the command output mode, reporting mode, and error code mode.
  - HI\_3G\_DeInitCard: Deinitializes the 3G dongle module.
- Scans the 3G dongle.
  - HI\_3G\_ScanCard: Scans the inserted 3G dongle.
- Connects to/Disconnects from the 3G network.
  - HI\_3G\_ConnectCard: Connects to the 3G network.



- HI\_3G\_DisconnectCard: Disconnects from the 3G network.
- Sets the operator searching mode and network registration priority.
  - HI\_3G\_SearchOperatorModeAcqorder: Sets the operator searching mode (for example, WCDMA or GSM) and network registration priority (for example, whether to register WCDMA or GSM in priority).
- Registers with an operator/Obtains operator information.
  - HI\_3G\_GetAllOperators: Obtains information about all operators.
  - HI\_3G\_GetCurrentOperator: Obtains information about the operator with which the SIM card is registered.
  - HI\_3G\_RegisterOperator: Registers with an operator.
- Obtains the signal strength.
  - HI\_3G\_GetQuality: Obtains the current signal strength.
- Obtains data traffic information.
  - HI\_3G\_GetDataFlow: Obtains the current data traffic information.
- Obtains the 3G dongle status.
  - HI\_3G\_GetCardStatus: Obtains the current 3G dongle status.

## 39.4 Development Guide

To use the 3G dongle, perform the following steps:

- Step 1** Scan the inserted 3G dongle and check whether it is compatible.
- Step 2** Initialize the 3G dongle and obtain card information, such as information about the port, vendor, and product.
- Step 3** Obtain the status of the 3G dongle to check whether it is ready.
- Step 4** Connect to the 3G network.
- Step 5** Disconnect from the 3G network.

**----End**

[Table 39-1](#) describes the typical error codes of the 3G dongle module.

**Table 39-1** Typical error codes of the 3G dongle module

Error Code	Cause Analysis	Solution
HI_ERR_SYS (-1)	The system invocation error or library C invocation failure occurs. For details, check errno.	Check errno to obtain the detailed error code.
HI_ERR_REPLY (-2)	The modem command return format is incorrect.	Check whether the 3G dongle is properly inserted and run the command in which the error occurs for several times.



Error Code	Cause Analysis	Solution
HI_ERR_TIMEOUT (-3)	Waiting for the return value of the modem command times out.	Check whether the 3G dongle is properly inserted and run the command in which the error occurs for several times.
HI_ERR_ENOCARD (-4)	The compatible 3G dongle cannot be detected.	Debug the inserted 3G dongle or replace it with a compatible one. For details about the dongle model, see the list of compatible components for the corresponding version.
HI_ERR_INVAL (-5)	The input parameter of an INTF API is invalid.	Check the parameters.
HI_ERR_NOMEMORY (-6)	The memory fails to be allocated.	Check the memory.
HI_ERR_UNSUPPORT (-7)	The AT command is not supported.	The inserted 3G card does not support this command.
operation not allowed (3)	The operation is forbidden.	Consult the supplier whether the command is supported.
SIM failure (13)	The SIM card does not exist or is faulty.	Check the SIM card.
SIM busy (14)	The SIM card is busy.	Check whether the SIM card is executing other commands, and consult the supplier.
not found (22)	The long name or short name of corresponding supplier cannot be found.	Check whether the long name and short name of the operator are correct.
no network service (30)	The network service is rejected.	Check the network.
Error codes that are greater than 0	Check the AT command interface regulation provided by the 3G dongle supplier.	Check the AT command interface regulation provided by the 3G dongle supplier, and consult the supplier.

The INTF layer of the 3G dongle module is used in the following scenarios:

- Connecting to/Disconnecting from the 3G network
- Obtaining operator information and registering with an operator network
- Extending a 3G dongle



## 39.4.2 Connecting to/Disconnecting from the 3G Network

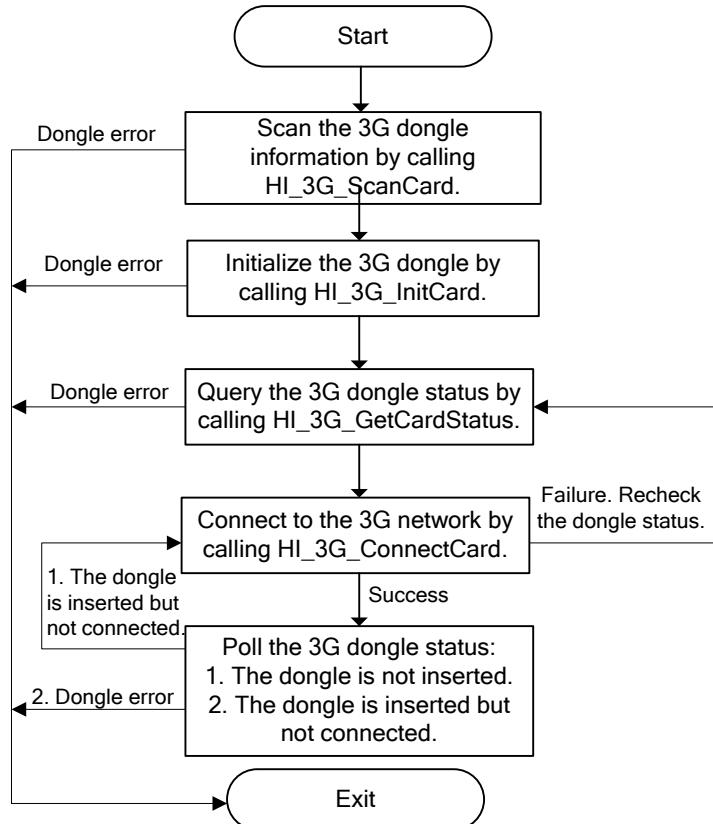
### Scenario

The 3G network needs to be connected and disconnected as required.

### Working Process

Figure 39-2 shows the process for connecting to and disconnecting from the 3G network.

**Figure 39-2** Process for connecting to and disconnecting from the 3G network



### Notes

Before the network is connected, a new thread needs to be started to monitor the 3G dongle status by calling `HI_3G_GetCardStatus` periodically. The card has two states.

`HI_3G_CARD_STATUS_UNAVAILABLE` indicates that the card is not inserted or removed.

`HI_3G_CARD_STATUS_DISCONNECTED` indicates that the card is inserted but not connected possibly because the signal is weak. The processing for the two states is as follows:

**Step 1** Obtain the 3G dongle status by calling `HI_3G_GetCardStatus`.

**Step 2** If `HI_3G_CARD_STATUS_UNAVAILABLE` is returned, the card is not inserted. Ask the user to insert the card and then exit. If `HI_3G_CARD_STATUS_DISCONNECTED` is returned, the card is inserted but not connected. Go to step 3 to reconnect the card.



**Step 3** Connect to the 3G network.

----End

## Samples

See `3g_sample.c`.

### 39.4.3 Obtaining Operator Information and Registering with an Operator Network

#### Scenario

There are multiple networks to be connected. The 3G dongle can automatically register with an operator network or the user can manually select a network.

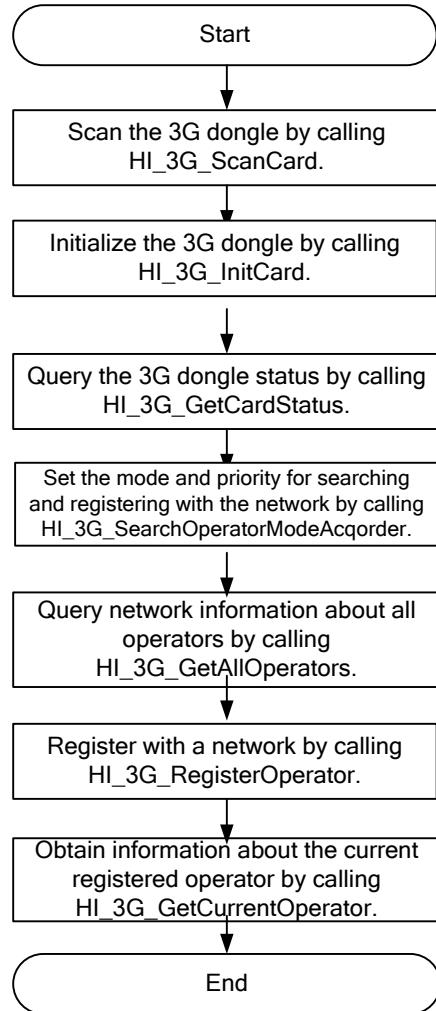
Automatic or manual network registration can be chosen by specifying the parameter of `HI_3G_SearchOperatorModeAcqorder`. Typically the 3G dongle automatically registers with the network after it is inserted, and only the connection needs to be started. The sample described in this section applies to the scenario in which the registered network needs to be switched, or the 3G dongle does not or fails to register with the network.

#### Working Process

[Figure 39-3](#) shows the process for obtaining operator information and registering with an operator network.



**Figure 39-3** Process for obtaining operator information and registering with an operator network



## Notes

The process for switching from a registered network to another network is as follows (if the 3G dongle has registered with the 3G WCDMA network but needs to switch to the 2G GSM network):

- Step 1** Disconnect from the current network by calling `HI_3G_DisConnectCard`.
- Step 2** Set the network searching mode and registration priority by calling `HI_3G_SearchOperatorModeAcqorder`. Set the following parameters:
  - HI\_3G\_CARD\_MODE\_GSM\_ONLY**: Indicates that only the GSM network is searched.
  - HI\_3G\_CARD\_ACQORDER\_GSM\_PRIOR\_UTRAN**: Indicates that the card registers with the GSM network in priority.
- Step 3** Obtain the network operators that meet the searching criteria (GSM network) by calling `HI_3G_GetAllOperators`.
- Step 4** Register with a found GSM network by calling `HI_3G_RegisterOperator`.



**Step 5** Connect the 3G dongle to the GSM network.

**Step 6** Disconnect the 3G dongle from the GSM network.

----End

## Samples

For details, see **3g\_sample.c**.



# Contents

<b>40 DirectFB .....</b>	<b>40-1</b>
40.1 Overview .....	40-1
40.1.1 Application Architecture .....	40-1
40.1.2 Important Concepts.....	40-2
40.2 Environment Configuration.....	40-3
40.2.1 Compiling the DirectFB .....	40-3
40.2.2 Set the Running Environment .....	40-4
40.3 Function Description .....	40-6
40.3.1 Working Process.....	40-6
40.3.2 Modules .....	40-6
40.4 Development Guide.....	40-8
40.4.1 Creating a Window .....	40-9
40.4.2 Displaying an Image .....	40-11
40.5 FAQs .....	40-13
40.5.1 What Do I Do If the DirectFB Fails to Be Compiled? .....	40-13
40.5.2 How Do I Compile the DirectFB that Supports Multiple Processes? .....	40-14
40.5.3 How Do I Compile the DirectFB of the Debug Version?.....	40-14
40.5.4 How Do I Reduce the Size of the DirectFB Library After Compilation? .....	40-14
40.5.5 What Do I Do If the Input Event in the DirectFB Affects the Performance of Other Input Event Applications? .....	40-15
40.5.6 What Do I Do If Intermittence Occurs During DirectFB Overlaying or Refreshing?.....	40-15
40.5.7 How Do I Set the Environment Variables When the DirectFB Is Running?.....	40-15
40.5.8 What Do I Do to Display BMP Images Properly? .....	40-16
40.5.9 What Do I Do to Display Chinese Characters Properly? .....	40-16
40.5.10 What Do I Do If the Failure of Running Multiple Processes Occurs? (1) .....	40-17
40.5.11 What Do I Do If the Failure of Running Multiple Processes Occurs? (2).....	40-17
40.5.12 What Do I Do If the Failure of Running Multiple Processes Occurs? (3) .....	40-17
40.5.13 What Do I Do to If the Drawn Rectangle Cannot Be Displayed? .....	40-18
40.5.14 How Do I Use Dual Buffers? .....	40-18
40.5.15 How Do I Set the Path of a Third-Party Library? .....	40-19
40.5.16 What Do I Do If the DirectFB Fails to Run? .....	40-19
40.5.17 What Do I Do If the Memory for the DirectFB Is Insufficient? .....	40-20



40.5.18 How Do I Create the Surface to Minimize the Memory Usage?.....	40-20
40.5.19 What Is the DirectFB Memory Usage Mechanism?.....	40-21
40.5.20 What Is the Memory Type for the Surface in the DirectFB?.....	40-21
40.5.21 How Do I Check the MMZ Memory that Has Been Used and the Surface Information in the DirectFB?.....	40-21
40.5.22 What Are the Hardware Acceleration Specifications of the DirectFB? .....	40-22



## Figures

<b>Figure 40-1</b> Overall architecture .....	40-2
<b>Figure 40-2</b> Invocation process of the DirectFB .....	40-6
<b>Figure 40-3</b> Creating a window .....	40-9
<b>Figure 40-4</b> Displaying an image .....	40-11



## Tables

<b>Table 40-1</b> Hardware acceleration specifications 1 .....	40-22
<b>Table 40-2</b> Hardware acceleration specifications 2 .....	40-22
<b>Table 40-3</b> Hardware acceleration specifications 3 .....	40-24



# 40 DirectFB

## 40.1 Overview

The DirectFB is a lightweight graphics library that provides hardware graphics acceleration and processing and abstraction of input devices. It integrates the window system that supports translucency, and enables multi-layer display based on the Linux FrameBuffer driver. The DirectFB encapsulates the graphics algorithms that are not supported by hardware by using software to implement hardware acceleration. It is designed for embedded systems to deliver the highest hardware acceleration performance with the lowest resource consumption.

- DirectFB-1.4.2 and DirectFB-1.6.1 are available in 32-bit system.
- Only DirectFB-1.6.1 is available in 64-bit system.

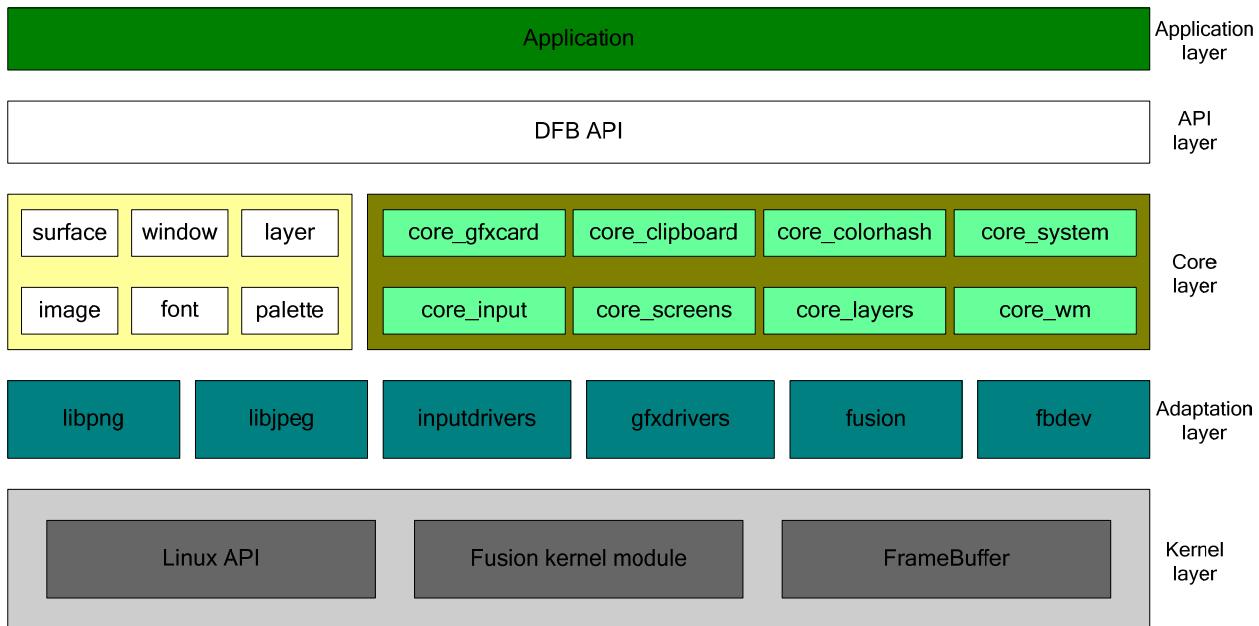
The DirectFB used by HiSilicon has been adapted for the two-dimensional engine (TDE), HiFB, and infrared (IR).

### 40.1.1 Application Architecture

[Figure 40-1](#) shows the overall architecture of the DirectFB.



**Figure 40-1** Overall architecture



For details, see the description in section [40.3.2 "Modules."](#)

## 40.1.2 Important Concepts

[DirectFB]

The DirectFB is a super class. All other classes are derived from it.

[Screen]

The screen device shows operations on the device, including setting the resolution. Each screen has multiple layers.

[Layer]

A layer is an independent image buffer. Most embedded devices have multiple layers. Those layers are blended by the hardware using the appropriate alpha values and then displayed.

[Window]

Each window has a surface, which is used to generate the image for composing the overlapping window. Each layer can contain multiple windows. The drawing of the DirectFB can be implemented through the surfaces of the windows.

[Surface]

A surface is a reserved memory area for storing pixel data. The drawing and blitting operations of the DirectFB are performed on surfaces. The surface memory can be allocated from the video memory or system memory, which can be specified in the parameter. The video memory corresponds to the continuous physical memory of the media memory zone (MMZ), and the system memory corresponds to the malloc system memory.

[Primary surface]



A primary surface is a special surface of the frame buffer at a specific layer. If the primary surface is in single-buffer mode, all operations performed on the surface can be directly seen from the screen.

## 40.2 Environment Configuration

### 40.2.1 Compiling the DirectFB

To compile the DirectFB, perform the following steps:

**Step 1** Go to the compilation directory by running the following command:

```
cd SDK root directory
make menuconfig
msp -->
    Graphic Config -->
        [ ] DirectFB Support -->
```

**Step 2** Press **y** on the keyboard to enter the DirectFB option to select the DirectFB version.

```
----- DirectFB Support
    DirectFB Version (DirectFB-1.6.1) -->
        () DirectFB-1.6.1
        () DirectFB-1.4.2
```

**Step 3** Choose the process (single or multiple).

```
----- DirectFB Support
    [ ] DirectFB Multi Process support
```

**Step 4** Choose whether to compile the debug version.

```
----- DirectFB Support
    [ ] DirectFB Debug support
```

**Step 5** Choose whether to compile the advanced CA version in the advanced CA environment.

```
----- DirectFB Support
    [ ] DirectFB Ca Support
```

**Step 6** Go to the DirectFB compilation library directory.

```
cd source/component/directfb
make
```

**Step 7** Compile the fusion driver if multiple processes are compiled.

```
cd source/msp/drv/directfb
make
```

**Step 8** Compile DirectFB Example.

```
cd sample/directfb
```



### Step 9 Make

Note 1:

When you compile the DirectFB on the server for the first time, the compilation may fail because the related tool is missing on the server. In this case, you need to install corresponding tools based on the prompt messages.

Note 2:

There are lots of compilation options. You can change the settings of some options in the **ci.sh** file.

For example, to disable the cursor function in this script, perform the following steps:

Open **Makefile** in **cd source/component/directfb**, and delete **ps2mouse** after the DirectFB input device option **--with-inputdrivers=** as follows:

```
./configure --host=$CROSS --prefix=$INSTALL_DIR --build=x86 --$prototype-
multi --enable-freetype --$comtype-debug --with-gfxdrivers=tde --with-
inputdrivers=keyboard,linuxinput,lirc --without-setsockopt --disable-osx
--disable-x11 --disable-x11vdpau --disable-mmxa --disable-sse --disable-
vnc --disable-mesa --disable-devmem --disable-pnm --disable-mpeg2 --
disable-bmp
```

The method of changing the settings of other options is similar.

Note 3:

To change the installation path of the DirectFB (the DirectFB libraries are installed under the root directory of the DirectFB by default), perform the following steps:

1. Open **Makefile**.

2. Enter a new installation path such as **/home/newinstall\_dir** as follows:

```
INSTALL_DIR=/home/newinstall_dir
```

**----End**

Note 4:

To change the installation path of samples (samples are installed under the root directory of the SDK by default).

**----End**

## 40.2.2 Set the Running Environment

To run a DirectFB application, perform the following steps:

**Step 1** Start the board and set the board IP address.

**Step 2** Load the core drivers **hifb.ko** and **tde.ko**, **hi\_media.ko**, **hi\_mmz.ko**, and **hi\_common.ko** required by the DirectFB. If the multiple processes are required, you also need to load **hi\_fusion.ko**.

**Step 3** Set the following environment variables for the DirectFB running library:



```
cd (installation directory for DirectFB libraries);
export LD_LIBRARY_PATH=$PWD
```

- For the DirectFB-1.4.2 version, run the following command:  
`cd directfb-1.4-0;`
- For the DirectFB-1.6.1 version, run the following commands:  
`cd directfb-1.6-0;`  
`export DFBARGS=module-dir=$PWD`

**Step 4** Set the environment variables for the DirectFB attributes, for example, resolution, pixel format, background, and cursor. The configuration methods are as follows:

- Method 1: Configure the variables using the export mode through the following script:  
`export DFBARGS=depth=32,layer-size=1280x720,pixelformat=ARGB,layer-bg-none,no-cursor,module-dir=$PWD`
- Method 2: Configure the variables through the configuration file **.directfbrc** or the configuration file of an executable application (if you know the name of the executable application as each executable application contains a configuration file). For example, for the executable application **df\_cursor**, its configuration file is **.directfbrc.df\_cursor**. In this case, you can run multiple cases with different configurations. The contents of the configuration file are as follows:

```
system=fbdev                      #output device
fbdev=/dev/fb0                     #layer to be used
wm=default                          #default for the window system
layer-size=1280x720                 #input resolution
depth=32                            #pixel depth
pixelformat=ARGB                     #pixel format
layer-bg-none                       #background without a layer
desktop-buffer-mode=frontonly       #buffer type
```

You can place the DirectFB library and the **.directfbrc** files in the **/mnt** directory, and then configure the following scripts by running **run.sh**:

```
export PWD=/mnt
export HOME=$PWD
export LD_LIBRARY_PATH=$PWD/directfb-lib
- For the DirectFB-1.4.2 version, use the following script:
  export DFBARGS=module-dir=$PWD/directfb-lib/directfb-1.4-0
- For the DirectFB-1.6.1 version, use the following script:
  export DFBARGS=module-dir=$PWD/directfb-lib/directfb-1.6-0
```

Execute **source run.sh** or **./run.sh**. In this case, the **.directfbrc** file under the **/mnt** directory is parsed.

- Method 3: Use the executable programs to bring in parameters and view the parameters by running **./df\_cursor --dfb:help**.  
`./df_cursor --dfb:desktop-buffer-mode=backvideo`

**Step 5** Initialize the display module by running the following commands:

```
cd SDK root directory; cd SDK/sample/fb; Run sample_fb and put the
```



running process in background. Otherwise, the display module cannot be enabled by the DirectFB; Or call the display initialization function in the case.

#### Step 6 Run the sample.

```
cd SDK/sample/directfb/sample;  
cd bin;  
.df_dok
```

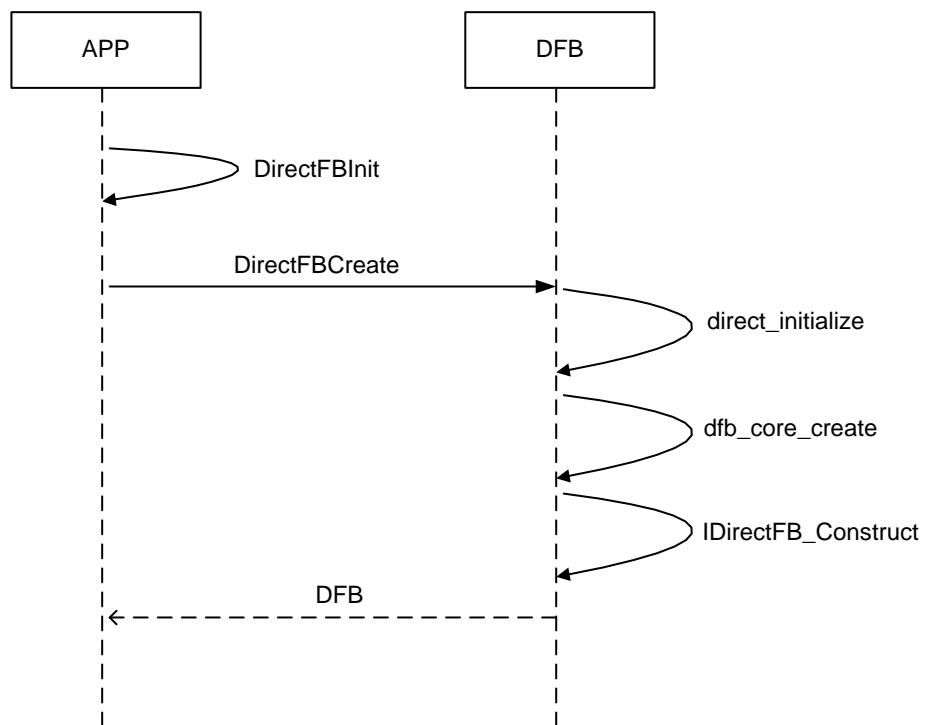
----End

## 40.3 Function Description

### 40.3.1 Working Process

Figure 40-2 shows the main invocation processes of the DirectFB.

**Figure 40-2** Invocation process of the DirectFB



### 40.3.2 Modules

#### 40.3.2.1 Lib Module

The code of the lib module is stored under the **lib** directory. The lib module consists of the following two sub modules:



- Direct

The public functions include the functions related to the hash table, linked list, thread, debugging information, signal processing, optimized memcpy, and platform.
- Fusion

The following two versions are provided:

  - Version applicable only to a single process. For this version, all the applications must run in one process.
  - Version applicable only to multiple processes. For this version, applications can run in multiple processes. The kernel driver **Linux-fusion** is required.

This module is the fusion module in [Figure 40-1](#).

### 40.3.2.2 Interface Module

The code of the interface module is stored under the **interfaces** directory and can be used to incorporate third-party components into the DirectFB. The interface module consists of the following three sub modules:

- Font

Fonts include dot-matrix fonts and vector fonts. Vector fonts support multiple formats such as truetype. These fonts can be implemented by using third-party libraries in freetype format.

The IDirectFBFont interface of the DirectFB is used to process fonts. After an adapter is added to a third-party font library, the IDirectFBFont interface is available.
- Image

A third-party library is required for supporting JPG and PNG images. The DirectFBImageProvider interface of the DirectFB is used to process images. After an adapter is added to a third-party image library, the IDirectFBImageProvider interface is available.
- Video

As videos have multiple formats, a third-party library is required for supporting videos. The IDirectFBVideoProvider interface of the DirectFB is used to process videos. After an adapter is added to a third-party font library, the IDirectFBVideoProvider interface is available.

### 40.3.2.3 Core Module

The code of the core module is stored under the **src** directory. The core module includes the following:

- Core components
- External interfaces

## Core Components

The DirectFB uses the master/slave model. That is, the first running application is the master process, and the subsequent applications are slave processes. The master process initializes and deinitializes the active area arena for all processes, and it can exit only after all slave processes exit. Slave processes only need to join in or exit the arena.

Core components are described as follows:

- dfb\_core\_clipboard: clipboard



- dfb\_core\_colorhash: palette
- dfb\_core\_gfxcard: graphics card
- dfb\_core\_input: input device
- dfb\_core\_layers: graphics layer. Only the HD graphics layer is supported.
- dfb\_core\_screens: logical screen
- dfb\_core\_system: display output. That is, the graphics drawn by dfb\_core\_gfxcard is output to the screen through fbdev.
- dfb\_core\_wm: window manager (WM).

## External Interfaces

The following external interfaces are provided only for upper-layer applications.

- IDirectFBIInputDevice: input device
- IDirectFBScreen: screen
- IDirectFBSurface: drawing surface
- IDirectFBPalette: palette
- IDirectFBFont: font
- IDirectFBIImageProvider: image
- IDirectFBVideoProvider: video
- IDirectFBWindow: window
- IDirectFBEventBuffer: event buffer

### 40.3.2.4 WM Module

The DirectFB provides two WMs, default manager and unique manager. The default manager is used by default. The two WMs are described as follows:

- Default manager: provides basic window management functions.
- Unique manager: provides a few functions and supports function extension.

### 40.3.2.5 Inputdrivers Module

The inputdrivers module is located at the adaptation layer of the input device. This module converts the events read from device files into the format supported by the DirectFB, and transmits the events by calling dfb\_input\_dispatch.

### 40.3.2.6 System Module

The system module is located at the adaptation layer of the frame buffer. This module implements abstraction of the display device.

As a component of the system module, fbdev is used to output data to the frame buffer.

## 40.4 Development Guide

The DirectFB is used in the following scenarios:

- Creating a window
- Displaying an image



## 40.4.1 Creating a Window

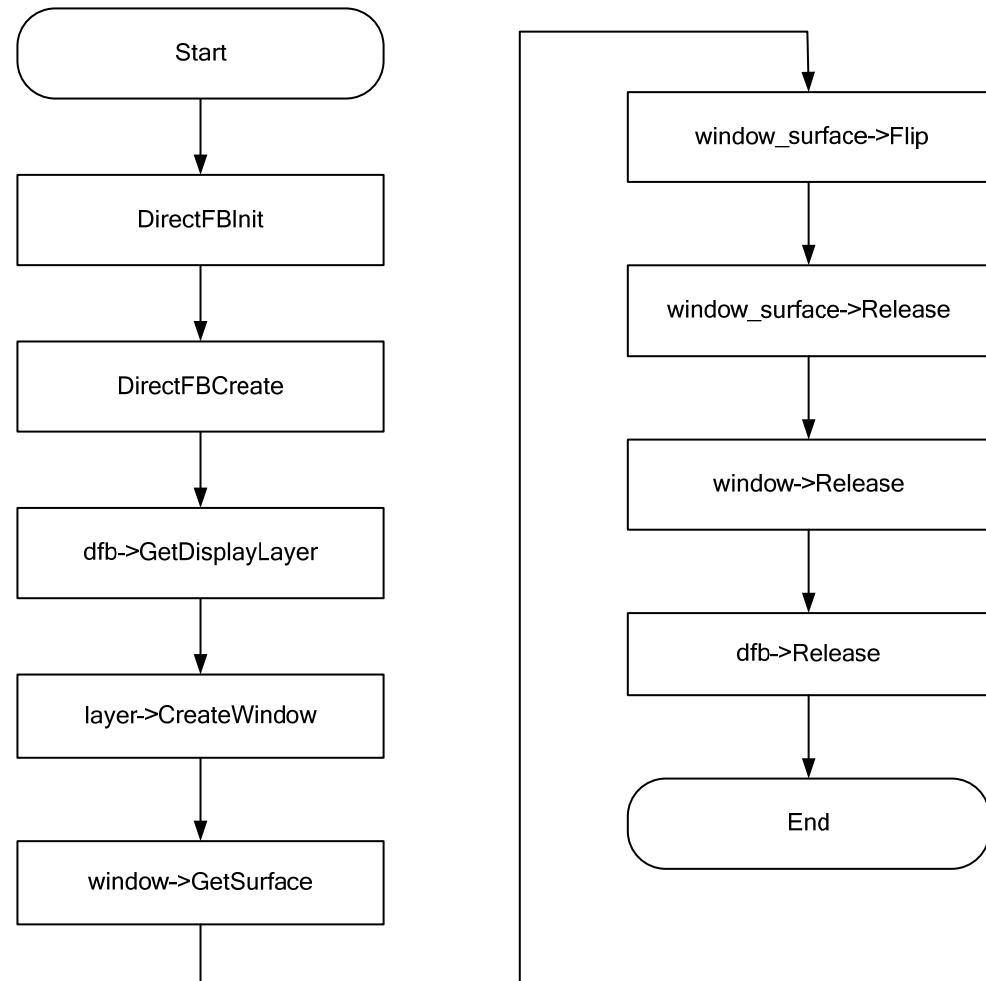
### Scenario

Create and display a window by using the DirectFB.

### Working Process

Figure 40-3 shows the working process of creating a window.

**Figure 40-3** Creating a window



The programming process is as follows:

- Step 1** Initialize the display module by running `cd root directory of the SDK` and `cd Develop/sample/fb`, run `sample_fb`, and mount it to the background.
- Step 2** Load the configuration files.
- Step 3** Create a DirectFB object.
- Step 4** Obtain the display surface.



**Step 5** Create a window. The allocated surface is the main surface.

**Step 6** Display the window.

**Step 7** Destroy the surface.

**Step 8** Destroy the window.

**Step 9** Destroy the created DirectFB object.

----End

## Notes

Ensure that the display module is initialized before running the DirectFB.

## Sample



For details about the code, see the **SDK/sample/directfb/DirectFB-examples-1.0.0\src** directory.

```
int main( int argc, char *argv[] )  
{  
    IDirectFB          *dfb;  
    IDirectFBDisplayLayer *layer;  
    IDirectFBWindow      *window;  
    IDirectFBSurface     *window_surface;  
    DFBWindowDescription desc;  
    int err;  
    int quit = 0;  
  
    DFBCHECK(DirectFBInit( &argc, &argv ));  
    DFBCHECK(DirectFBCreate( &dfb ));  
    DFBCHECK(dfb->GetDisplayLayer( dfb, DLID_PRIMARY, &layer ));  
    layer->SetCooperativeLevel( layer, DLSCL_ADMINISTRATIVE );  
  
    desc.flags = ( DWDESC_POSX | DWDESC_POSY |  
                   DWDESC_WIDTH | DWDESC_HEIGHT );  
    desc.posx  = 0;  
    desc.posy  = 0;  
    desc.width = 1280;  
    desc.height = 720;  
  
    DFBCHECK( layer->CreateWindow( layer, &desc, &window ) );  
    window->GetSurface( window, &window_surface );  
  
    window_surface->Flip( window_surface, NULL, vsync );  
  
    while (!quit)
```



```
{  
    sleep(30);  
  
}  
  
window_surface->Release( window_surface );  
window->Release( window );  
layer->Release( layer );  
dfb->Release( dfb );  
  
return 0;  
}
```

## 40.4.2 Displaying an Image

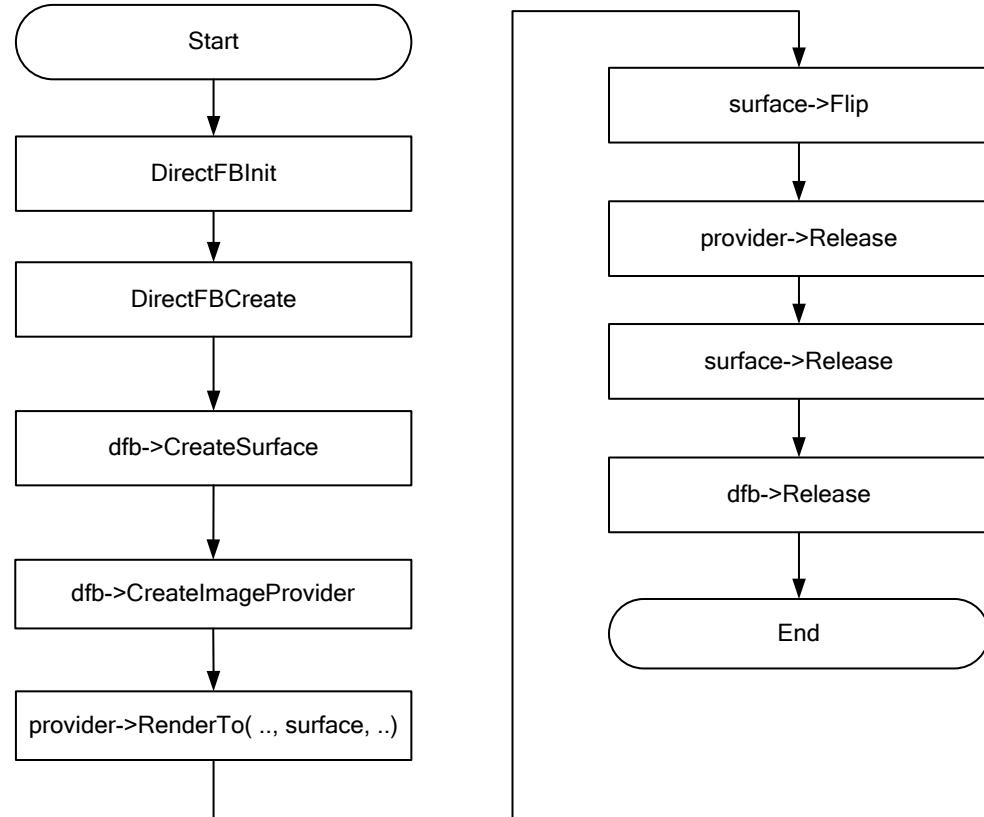
### Scenario

Display an image by using the DirectFB.

### Working Process

Figure 40-4 shows the working process of displaying an image.

**Figure 40-4** Displaying an image





The programming process is as follows:

- Step 1** Initialize the display module by running **cd root directory of the SDK** and **cd Develop/sample/fb**, run sample\_fb, and mount it to the background.
- Step 2** Create a window. If the created surface is set to the main surface, the drawings can be displayed on the surface directly. If not, the drawings are displayed only after being flipped to the main surface.
- Step 3** Create a decoder.
- Step 4** Decode the image and store data to the specified surface.
- Step 5** Display the image.
- Step 6** Destroy the decoder.
- Step 7** Destroy the surface.
- Step 8** Destroy the created DirectFB object.

----End

## Notes

Ensure that the display module is initialized before running the DirectFB.

## Sample



### NOTE

For details about the code, see the **SDK/sample/directfb/DirectFB-examples-1.0.0\src** directory.

```
int main( int argc, char *argv[] )  
{  
    IDirectFB          *dfb;  
    IDirectFBIImageProvider *provider;  
    IDirectFBSurface     *window_surface;  
    DFBSurfaceDescription dsc;  
    int err;  
    int quit = 0;  
  
    DFBCHECK(DirectFBInit( &argc, &argv ));  
    DFBCHECK(DirectFBCreate( &dfb ));  
    dfb->SetCooperativeLevel( dfb, DFSCL_FULLSCREEN );  
    DFBCHECK(dfb->CreateImageProvider( dfb,  
        DATADIR"/tux.png",&provider ));  
    DFBCHECK (provider->GetSurfaceDescription (provider, &dsc));  
    dsc.flags = DSDESC_WIDTH | DSDESC_HEIGHT|DSDESC_CAPS;  
  
    dsc.width = 720;  
    dsc.height = 576;  
    dsc.caps = DSCAPS_PRIMARY | DSCAPS_DOUBLE;
```



```
DFBCHECK(df->CreateSurface( dfb, &dsc, &window_surface ));

DFBCHECK(provider->RenderTo( provider, window_surface, NULL ));
provider->Release( provider );
window_surface->Flip( window_surface, NULL, 0 );

while (!quit)
{
    sleep(30);
}

window_surface->Release( window_surface );
df->Release( dfb );

return 0;
}
```

## 40.5 FAQs

### 40.5.1 What Do I Do If the DirectFB Fails to Be Compiled?

#### Problem Description

The DirectFB fails to be compiled on the server, with the information similar to **checking for pkg-config... no** displayed.

#### Solution

- Check whether the following tools are installed on the server by running the **which** command. If the tools are not installed, you need to install them on the server first.
  - script
  - autoconf
  - gawk
  - pkg-config
- The compiling of the DirectFB is dependent on some SDK component libraries, such as **freetype**, **jpeg**, **png**, and **liz**. You need to first compile the SDK and then the DirectFB.

### 40.5.2 How Do I Compile the DirectFB that Supports Multiple Processes?

#### Problem Description

How do I enable the DirectFB to support multiple processes?



## Solution

Run the following commands:

```
cd source/msp/drv/directfb  
make install;
```

### 40.5.3 How Do I Compile the DirectFB of the Debug Version?

#### Problem Description

How do I compile the DirectFB of the debug version?

## Solution

Run **make menuconfig** under the SDK root directory to configure the DirectFB compilation mode as debugging mode, and then run the following command:

```
cd source/component/directfb  
make;make install;
```

### 40.5.4 How Do I Reduce the Size of the DirectFB Library After Compilation?

#### Problem Description

In the low-cost solution, how do I reduce the size of the DirectFB library?

## Solution

- Disable unnecessary functions during compilation. You can view the functions that are provided by the DirectFB through **./configure-help**. For example, for the **--with-inputdrivers=keyboard,linuxinput,lirc,ps2mouse** option, you can disable the functions that are not needed in the **Makefile**. If you do not need the cursor function, you can make the following changes:

```
./configure --host=$CROSS --prefix=$INSTALL_DIR --build=x86 --  
$prototype-multi --enable-freetype --$comtype-debug --with-  
gfxdrivers=tde --with-inputdrivers=keyboard,linuxinput,lirc --  
without-setsockopt --disable-osx --disable-x11 --disable-x11vdpau --  
disable-mmx --disable-sse --disable-vnc --disable-mesa --disable-  
devmem --disable-pnm --disable-mpeg2 --disable-bmp
```

- Perform the strip operation on each **.so** library under the **directfb/lib** directory after the library compilation completes.



## 40.5.5 What Do I Do If the Input Event in the DirectFB Affects the Performance of Other Input Event Applications?

### Problem Description

When the DirectFBCreate is called to create the dfb handle, the event reception of other applications (such as remote control) that process the input events may be delayed.

### Solution

During the processing of multiple processes or threads, input devices block each other. Therefore, you are advised to mask the DirectFB function when it is not used in input event management. You can mask the function using the following methods:

- Method 1: Delete corresponding library files under the **inputdrivers** directory.
- Method 2: Mask the unnecessary functions during compilation.

## 40.5.6 What Do I Do If Intermittence Occurs During DirectFB Overlaying or Refreshing?

### Problem Description

Intermittence or flickers occur when the DirectFB draws the display UI.

### Solution

Check whether the hardware acceleration takes effect.

- Check whether **gfxdrivers/libdirectfb\_tde.so** exists.
- Check whether the memory meets requirements. You can verify the issue by increasing the memory size. If the fault is resolved, then you can recalculate the memory size according to the actual scenario.

## 40.5.7 How Do I Set the Environment Variables When the DirectFB Is Running?

### Problem Description

When a DirectFB application is running, how do I set the environment variables such as the layer size and pixel format?

### Solution

- Method 1: Configure the variables using the export mode through the following script:

```
export DFBARGS=depth=32,layer-size=1280x720,pixelformat=ARGB,layer-bg-none,no-cursor,module-dir=$PWD
```
- Method 2: Configure the variables through the configuration file **.directfbrc** or the configuration file of an executable application (if you know the name of the executable application as each executable application uses a configuration file). For example, for the executable application **df\_cursor**, its configuration file is **.directfbrc.df\_cursor**. In this



case, you can run multiple cases with different configurations. The contents of the configuration file are as follows:

```
system=fbdev           #output device
fbdev=/dev/fb0         #layer to be used
wm=default             #default for window system
layer-size=1280x720   #input resolution
depth=32               #pixel depth
pixelformat=ARGB       #pixel format
layer-bg-none          #background without a layer
desktop-buffer-mode=frontonly #buffer type
```

You can place the DirectFB library and the **.directfbrc** files in the **/mnt** directory, and then configure **run.sh** by running the following scripts:

```
export PWD=/mnt
export HOME=$PWD
export LD_LIBRARY_PATH=$PWD/directfb-lib
```

- For the DirectFB-1.4.2 version, run the following command:

```
export DFBARGS=module-dir=$PWD/directfb-lib/directfb-1.4-0
```

- For the DirectFB-1.6.1 version, run the following commands:

```
export DFBARGS=module-dir=$PWD/directfb-lib/directfb-1.6-0
source run.sh or. ./run.sh to parse the .directfbrc under the /mnt
directory
```

- Method 3: Use the executable programs to bring in parameters. For example, run **dr\_cursor** cases.  

```
./df_cursor --dfb:help info
./df_cursor --dfb:desktop-buffer-mode=frontonly single buffer mode
```
- Method 4: Set the variables through the interface DirectFBSetOption in applications. For example, set the adaptation library path:  

```
DirectFBSetOption ( "module-dir" , "/mnt/directfb/directfb/lib/directfb-
1.4-0" );
```

## 40.5.8 What Do I Do to Display BMP Images Properly?

### Problem Description

BMP images cannot be displayed.

### Solution

The DirectFB does not support BMP images. Use images in other formats.

## 40.5.9 What Do I Do to Display Chinese Characters Properly?

### Problem Description

English characters are displayed properly, but Chinese characters are garbled.



## Solution

The DirectFB supports only the UTF8 encoding format. Convert the Chinese characters into the UTF8 encoding format.

### 40.5.10 What Do I Do If the Failure of Running Multiple Processes Occurs? (1)

#### Problem Description

When multiple processes run concurrently, failures occur and the following error information is displayed:

```
opening '/dev/fusion0' and '/dev/fusion/0' failed?
```

## Solution

Run **insmosd hi fusion.ko**. If it does not exist, compile the DirectFB to generate it.

### 40.5.11 What Do I Do If the Failure of Running Multiple Processes Occurs? (2)

#### Problem Description

When two instances are started in sequence, the second one fails to be started.

## Solution

Before running multiple instances, set the DirectFB level to **DFSCL\_NORMAL** as follows:

```
dfb->SetCooperativeLevel( dfb, DFSCL_NORMAL );
```

### 40.5.12 What Do I Do If the Failure of Running Multiple Processes Occurs? (3)

#### Problem Description

When the DirectFB application with multiple processes runs, failures occur.

## Solution

- After the board starts up, if the file system is read-only, the multiple processes are not supported and the device files cannot be generated by loading the **hi fusion.ko** driver.
- Select the multi-process compiling mode during compilation.
- Load the **hi fusion.ko** driver before running the DirectFB case.



## 40.5.13 What Do I Do to If the Drawn Rectangle Cannot Be Displayed?

### Problem Description

The drawn rectangle on the main surface cannot be displayed.

### Solution

Perform the following operation:

```
surface->Flip(surface, NULL, vsync );
```

## 40.5.14 How Do I Use Dual Buffers?

### Problem Description

Dual buffers are commonly used during drawing. The working principle is as follows: A buffer is used for drawing. When the buffer is used to submit images for display, the other buffer starts to be used for drawing. In this way, images are drawn and displayed by using dual buffers alternatively, which improves efficiency. How do I use dual buffers in the DirectFB?

### Solution

Each buffer of the DirectFB is a memory. The buffers of the DirectFB can be allocated in either of the following ways:

- Allocated by users. If the buffers are allocated by users, the memory address is transferred to the DirectFB when a surface is being created. You must set the **DSCAPS\_PREMULTIPLIED** attribute when creating the surface.
- Allocated by the DirectFB. The DirectFB supports triple buffers or dual buffers. You need to set the buffer mode to **DSCAPS\_TRIPLE** or **DSCAPS\_DOUBLE** when creating a surface during programming, and specify a surface memory for storing images.

To minimize the memory size, the surface is from the layer and is obtained by calling GetLayerSurface. Most users adopt this method.

The following is a drawing instance using dual buffers:

```
dfb->CreateSurface(dfb, &sds, &surface);
for(i=0;i<100;i++)
{
    surface->DrawRectangle( surface, 100, 100+i, 200, 100+i );
    sleep(1);
    surface->Flip(surface, NULL, 0 );
    surface->Release(surface);
```



## 40.5.15 How Do I Set the Path of a Third-Party Library?

### Problem Description

When the DirectFB is started, the following error information is displayed:

```
(!) DirectFB/core/system: No system found!
(#) DirectFSError [DirectFBCreate() failed]: No (suitable) implementation
found!
```

### Solution

- Method 1

Ensure that the running path and compilation path of the DirectFB are the same.

- Method 2

Run the following commands before running the DirectFB:

```
cd (lib directory of the DirectFB)
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD
```

- For the DirectFB-1.4.2 version, run the following command:

```
cd directfb-1.4-0
```

- For the DirectFB-1.6.1 version, run the following commands:

```
cd directfb-1.6-0
```

```
export DFBARGS=module-dir=$PWD
```

- Method 3

Set the path by calling the corresponding interface.

For example, if the path of the DirectFB library is **/mnt/directfb/directfb/lib** during running, call DirectFBSetOption ("module-dir", "/mnt/directfb/directfb/lib/directfb-1.4-0") after initializing the DirectFB or DirectFBSetOption ("module-dir", "/mnt/directfb/directfb/lib/directfb-1.6-0");

## 40.5.16 What Do I Do If the DirectFB Fails to Run?

### Problem Description

The DirectFB fails to run and the following information is displayed:

```
(!) Direct/Util: opening '/dev/fb2' failed
--> Operation not permitted
(!) DirectFB/FBDev: Error opening framebuffer device!
```

### Solution

Initialize the display module before running the DirectFB. Run sample\_fb and mount it to the background.

The commands are as follows:

```
cd SDK root directory
```



```
cd ./sample/fb
./sample_fb
ctrl + Z (mount the application to the background)
```

## 40.5.17 What Do I Do If the Memory for the DirectFB Is Insufficient?

### Problem Description

When the DirectFB is running, the memory is insufficient, even the system is suspended.

### Solution

Only the frame buffer memory supports hardware operations. The buffer memory can be adjusted in the following methods:

- Increase the memory size by loading a .ko driver. When loading the frame buffer driver, increase the memory size for the high-definition layer and modify the parameter **vram0\_size**. The details are as follows:

```
rmmmod hifb.ko
insmod /kmod/hi_fb.ko video="hi_fb: vram0_size:7000"
```
- Increase the memory size by compiling the frame buffer driver into the kernel. The details are as follows:

```
cd SDK root directory;
make menuconfig
Msp ---> Graphic Config ---> HD FrameBuffer Size
make linux_clean;make linux_install
Reburn the kernel.
```

## 40.5.18 How Do I Create the Surface to Minimize the Memory Usage?

### Problem Description

In the application of the DirectxFB, how do I create the surface to minimize the memory usage?

### Solution

- Call `layer->EnableCursor` to disable the cursor function when it is not in need.
- Call `layer->GetSurface` (`layer & LayerSurface`) to directly obtain the layer surface. In this case, the displayed address is the actual address to be displayed.



## 40.5.19 What Is the DirectFB Memory Usage Mechanism?

### Problem Description

During the application, no error occurs when CreateSurface() is called. However, errors occur during the lock, blit, or flip operation.

### Solution

- Since no operations are performed on the actual memory address in CreateSurface(), and only the offset information of the memory required by the surface is recorded, creating multiple surfaces has the same effect as creating one surface. During the lock, blit, or flip operation where operation on the surface memory address is required, an internal judgment is made to determine whether the surface offset exceeds the available memory size. If the surface offset exceeds the available memory size, a message indicating insufficient memory size is returned otherwise, the surface address is obtained based on the surface size.
- The size of the offset recorded by the surface created later is the total size of the surfaces created earlier plus the size of the current surface. Whether the memory size is enough is determined based on the offset of the surface to be used.

## 40.5.20 What Is the Memory Type for the Surface in the DirectFB?

### Problem Description

For the three parameters **DSCAPS\_PRIMARY**, **DSCAPS\_SYSTEMONLY**, and **DSCAPS\_VIDEOONLY**, which are allocated by the MMZ and which are allocated by the malloc?

### Solution

There are only two parameters (DSCAPS\_SYSTEMONLY and DSCAPS\_VIDEOONLY) related to the memory. The parameter DSCAPS\_PRIMARY is used to specify whether to bind the created surface to the layer. It has nothing to do with the memory usage.

- System memory: DSCAPS\_SYSTEMONLY
- MMZ memory: DSCAPS\_VIDEOONLY

## 40.5.21 How Do I Check the MMZ Memory that Has Been Used and the Surface Information in the DirectFB?

### Problem Description

How do I check the MMZ memory that has been used and the surface information in the DirectFB?

### Solution

Currently, there is no specific log information that can be used to check the memory usage and surface information. Users need to use the surface based on their scenario, and the information of each surface is consistent with that during creation. As for the memory size that has been used, users need to calculate based on the number of surfaces used.



## 40.5.22 What Are the Hardware Acceleration Specifications of the DirectFB?

### Problem Description

What are the hardware acceleration specifications of the DirectFB?

### Solution

Table 40-1, Table 40-2, and Table 40-3 describes the hardware acceleration specifications.

**Table 40-1** Hardware acceleration specifications 1

Layer	Description	Supported or Not
Dual graphics layers	Display of multiple graphics layers on the screen	No (Only HD graphics layers are supported. If the sources of the HD and SD graphics layers are the same, you can call the SDK interfaces to implement the function.)
Inter-layer alpha	Translucence between graphics layers	Yes
Inter-layer colorkey	Transparency between graphics layers	Yes
Graphics layer enlargement	Output enlargement of graphics layers by using the hardware	Yes
Start position of a graphics layer	Setting of the start position of a graphics layer on the screen	Yes
Palette	Palette not supported	No

**Table 40-2** Hardware acceleration specifications 2

Blit Operation	Description	Operation Attribute	Blit/Stretchblt Supported or Not
DSBLIT_NOFX	No operation is performed.	Single source	Yes
DSBLIT_BLEND_ALPHACHANNEL	Channel alpha is blended.	Dual sources	Yes
DSBLIT_BLEND_COLORALPHA	Pixel alpha is blended.	Dual sources	Yes



Blit Operation	Description	Operation Attribute	Blit/Stretchblt Supported or Not
DSBLIT_COLORIZE	The source color is adjusted by using the red-green-blue (RGB) components of the color.	Single source	Yes
DSBLIT_SRC_COLORKEY	Source colorkey	Dual sources	Yes
DSBLIT_DST_COLORKEY	Destination colorkey	Dual sources	Yes
DSBLIT_SRC_PREMULTIPLY	The source is premultiplied.	Single source	Yes
DSBLIT_DST_PREMULTIPLY	The destination is premultiplied.	Single source	Yes
DSBLIT_DEMULTIPLY	The result is not premultiplied.	Single source	No
DSBLIT_SRC_PREMULTCOLOR	The source color is adjusted by using the color alpha component.	Single source	Yes
DSBLIT_XOR	The XOR operation is performed for the source and destination regions.	Dual sources	Yes
DSBLIT_INDEX_TRANSLATION	Palette transparency (used separately)	Dual sources	No
DSBLIT_ROTATE90	Rotated by 90°	Single source	No
DSBLIT_ROTATE180	Rotated by 180°	Single source	No
DSBLIT_ROTATE270	Rotated by 270°	Single source	No
DSBLIT_COLORKEY_PROTECT	The pixel to be written and source pixel mismatch.	Dual sources	No
DSBLIT_SRC_MASK_ALPHA	The source alpha channel is adjusted by using the alpha channel of the source mask region.	Single source (surface mask is required)	No
DSBLIT_SRC_MASK_COLOR	The source RGB channel is adjusted by using the RGB channel of the source mask region.	Single source (surface mask is required)	No



**Table 40-3** Hardware acceleration specifications 3

Draw Operation	Description	Operation Attribute	Draw/Fill Supported or Not
DSDRAW_NOFX	No operation is performed.	Single source	Yes
DSDRAW_BLEND	Alpha component of the color is used.	Dual sources	Yes
DSDRAW_DST_COLORKEY	Color is written only when the destination color matches the destination colorkey.	Dual sources	No
DSDRAW_SRC_PREMULTIPLY	The color is adjusted and then drawn.	Dual sources	Yes
DSDRAW_DST_PREMULTIPLY	The destination image is premultiplied.	Single source	Yes
DSDRAW_DEMULTIPLY	The color of the destination is not premultiplied.	Single source	No
DSDRAW_XOR	The XOR operation is performed for the destination region and color after they are premultiplied.	Single source	No