International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

J.1205

(01/2022)

SERIES J: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

Smart TV operating system

Smart television operating system – Hardware abstract layer application programming interface

Recommendation ITU-T J.1205

Recommendation ITU-T J.1205

Smart television operating system – Hardware abstract layer application programming interface

Summary

Recommendation ITU-T J.1205 defines the hardware abstract layer application programming interface (API) of a smart TV operating system (TVOS) to enable integrated broadcast and broadband (IBB)-capable cable set-top box (STB) and TV to apply to broadcasting services and IP-based interactive services provided by cable television operators and third-party providers.

The TVOS hardware abstract layer (HAL) consists of multiple hardware abstraction functional interface modules. These modules implement abstraction and encapsulation of different hardware capabilities and provide the upper-layer software with interfaces used to invoke the corresponding hardware capabilities.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T J.1205	2022-01-13	9	11.1002/1000/14841

Keywords

Abstract layer, API, hardware, smart TV operating system, TVOS.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents/software copyrights, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the appropriate ITU-T databases available via the ITU-T website at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Page

Introduction

This Recommendation is the fifth in a series on a smart television operating system (TVOS). The Recommendations for this smart TVOS cover functional requirements, architecture, and security and application programming interfaces (APIs):

Smart television operating system – Functional requirements [ITU-T J.1201]

Smart television operating system – Architecture [ITU-T J.1202]

Smart television operating system – Specification [b-ITU-T J.1203]

Smart television operating system – Security framework [b-ITU-T J.1204]

Smart television operating system – Hardware abstract layer application programming interface (ITU-T J.1205)

Recommendation ITU-T J.1205

Smart television operating system – Hardware abstract layer application programming interface

1 Scope

This Recommendation specifies the hardware abstract layer API of a smart TV operating system over integrated broadcast and broadband cable networks. The TVOS hardware abstract layer (HAL) API provides abstraction and encapsulation of the TVOS hardware platform capability, uses a unified abstraction and encapsulation model for hardware devices of the same type and offers unified invocation interfaces for the upper-layer software to access and control the hardware platform capability.

This Recommendation is based on the requirements and architecture found in [ITU-T J.1201] and [ITU-T J.1202].

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T J.1201] Recommendation ITU-T J.1201 (2022), *Smart television operating system – Functional requirements*.

[ITU-T J.1202] Recommendation ITU-T J.1202 (2022), *Smart television operating system – Architecture*.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 integrated broadcast and broadband DTV service IBB [b-ITU-T J.205]: A service that simultaneously provides an integrated experience of broadcasting and interactivity relating to media content, data and applications from multiple sources, where the interactivity is sometimes associated with broadcasting programmes.

3.1.2 smart television operating system (TVOS) [ITU-T J.1201]: A system software running on an integrated broadcast and broadband-capable (IBB-capable) cable set top box (STB) and television (TV) that is capable of managing hardware, software and data resources of the IBB-capable cable STB and TV, supporting and controlling the application software execution.

3.2 Terms defined in this Recommendation

None.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API	Application Programming Interface
AV	Audio Video
CA	Certification Authority
DCAS	Downloadable Conditional Access System
DVB	Digital Video Broadcasting
ES	Elementary Stream
HAL	Hardware Abstract Layer
MPEG	Moving Picture Experts Group
OSD	On-Screen Display
PCR	Program Clock Reference
PID	Packet Identifier
PTS	Presentation Time Stamp
STB	Set-Top Box
TEE	Trusted Execution Environment
TS	Transport Stream

5 Conventions

In this Recommendation:

The phrase "is required to" indicates a requirement which must be strictly followed and from which no deviation is permitted if conformity with this document is to be claimed.

The phrase "is recommended" indicates a requirement which is recommended but which is not absolutely required. Thus, this requirement needs not be present to claim conformity.

The phrase "is prohibited from" indicates a requirement which must be strictly followed and from which no deviation is permitted if conformity with this document is to be claimed.

The phrase "can optionally" indicates an optional requirement which is permissible, without implying any sense of being recommended. This term is not intended to imply that the vendor's implementation must provide the option and the feature can be optionally enabled by the network operator/service provider. Rather, it means the vendor may optionally provide the feature and still claim conformity with this Recommendation.

In the body of this document and its annexes, the words *shall*, *shall not*, *should*, and *may* sometimes appear, in which case they are to be interpreted, respectively, as *is required to*, *is prohibited from*, *is recommended*, and *can optionally*. The appearance of such phrases or keywords in an appendix or in material explicitly marked as *informative* are to be interpreted as having no normative intent.

6 Definitions of basic data types and operators

6.1 Definition of data types

See Table 1 for the definition of data types.

Table 1 – Definition of data types

Name of data type	Definition of data type
U8	unsigned char
U16	unsigned short
U32	unsigned int
U64	unsigned long long
S8	signed char
S16	short
S32	int
S64	signed long long
CHAR	char
FLOAT	float
DOUBLE	double
HANDLE	unsigned int

6.2 Definition of relational operators

See Table 2 for the definition of relational operators.

Table 2 – Definition of relational operators

Name of relational operator	Meaning of relational operator
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Unequal to

6.3 Definition of arithmetic operators

See Table 3 for the definition of arithmetic operators.

Table 3 – Arithmetic operators

Name of relational operator	Meaning of relational operator
<<	Shift one bit to left
>>	Shit one bit to right
+	Add
++	Progressively add 1
-	Subtract
--	Progressively subtract 1

7 Interface overview

TVOS hardware abstract layer (HAL) implements abstract packaging of TVOS hardware platform capability, adopts a uniform abstract packaging model for the same type of hardware equipment, and provides a uniform invocation interface for the upper-layer software to access and control hardware platform capability. TVOS HAL hardware abstract module includes two categories: special-purpose hardware abstract interface for media processing and general-purpose hardware abstract interface, as shown in Figure 1.

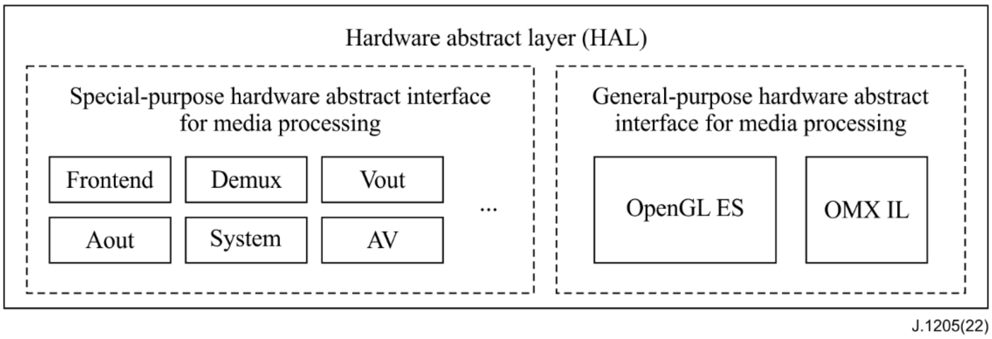
Hardware abstract layer (HAL)
Special-purpose hardware abstract interface for media processing
Frontend
Demux
Vout
Aout
System
AV
...
General-purpose hardware abstract interface for media processing
OpenGL ES
OMX IL
J.1205(22)

Figure 1 – Hardware abstract interface

The special-purpose hardware abstract interface for media processing is an extended definition of TVOS for media service in the field of broadcasting and TV, including 6 submodules, namely, Aout module, Demux module, Frontend module, System module, Vout module and audio video (AV) module. The general-purpose hardware function interface takes in mature hardware abstract interface standards in the industry, including Open OMX IL audio video encoding/decoding module, and OpenGL elementary stream (ES) graphics module, etc. Function definitions of each module are shown in Table 4.

Table 4 – Hardware abstract interface module

S/N	Classification	Module	Description	Remarks
1	Special-purpose hardware abstract interface for media processing	Audio output (Aout) module	It defines the interface of audio output to the audio device, such as left/right channel output, SPDIF output, and HDMI output. In addition, it also provides the functions of audio attribute acquisition and audio manipulation, such as acquiring program clock reference (PCR) attribute of audio stream, audio mute, pause and other attributes	See Annex A for interface details
2		Demultiplexing (Demux) module	It defines the interface to manipulate the demultiplexer and provides the functions of demultiplexing, descrambling and filtering according to the corresponding setting conditions	See Annex B for interface details
3		Frontend module	It defines the interface to operate Tuner and provides the support of multiple standards,	See Annex C for interface details

Table 4 – Hardware abstract interface module

S/N	Classification	Module	Description	Remarks
			such as DVB-C\DVB-S\DVB-T	
4		system module	It defines the interface for system setting, such as the functions of standby, chip ID acquisition, restart and shutdown	See Annex D for interface details
5		Vout module	It defines the interface of video output to the display device, such as 3D video output, resolution ratio and frame rate	See Annex E for interface details
6		Audio video playing (AV) module	It defines the interface for audio video processing and provides the functions of audio video encoding/decoding, transport stream (TS) and ES stream decoding and playing	See Annex F for interface details
9	General-purpose hardware abstract interface	OpenMAX IL module	It defines a series of media encoding and decoding interfaces. A [b-OpenMAX IL 1.1.2] based interface is adopted in this part.	The interface follows [b-OpenMAX IL 1.1.2]
10		OpenGL ES module	It defines a series of 3D graphics interfaces. A [b-OpenGL ES 2.0.25] based interface is adopted in this part.	The interface follows [b-OpenGL ES 2.0.25]

The submodules contained in the general-purpose hardware abstract interface for media processing have similar structures. For example, the structure of Aout module is shown in Figure 2.

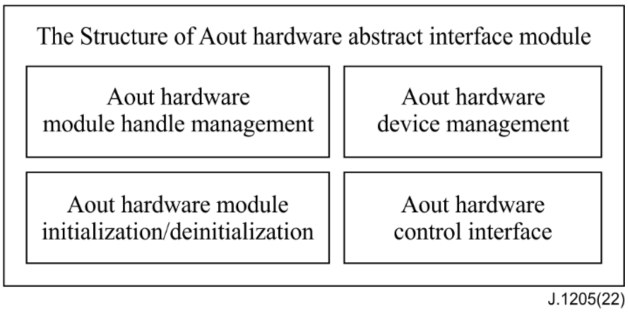
The Structure of Aout hardware abstract interface module
Aout hardware module handle management
Aout hardware device management
Aout hardware module initialization/deinitialization
Aout hardware control interface
J.1205(22)

Figure 2 – Aout module structure

Aout hardware module handle management in Figure 2 adopts the structure defined in Table B.41 to describe the hardware module handle, and provides a uniform interface for the upper-layer application to acquire Aout module handle. Aout hardware module initialization/deinitialization module completes initialization and deinitialization of the audio output hardware. Aout hardware device management implements the uniform management of Aout hardware control interface.

8 Invocation mechanism

TVOS hardware abstract interface module follows the requirements of the smart TV operating system (TVOS) [ITU-T J.1201], and it is implemented by adopting Stub hardware abstract module. Stub hardware abstract module connects a hardware module to several hardware devices and their operation methods in the form of Stub operating function, provides relevant hardware capability invocation method for the upper-layer software and implements operation and control of relevant hardware capabilities through corresponding hardware module ID to Stub operating function pointer.

The principle of Stub hardware abstract model at TVOS HAL layer is shown in Figure 3.

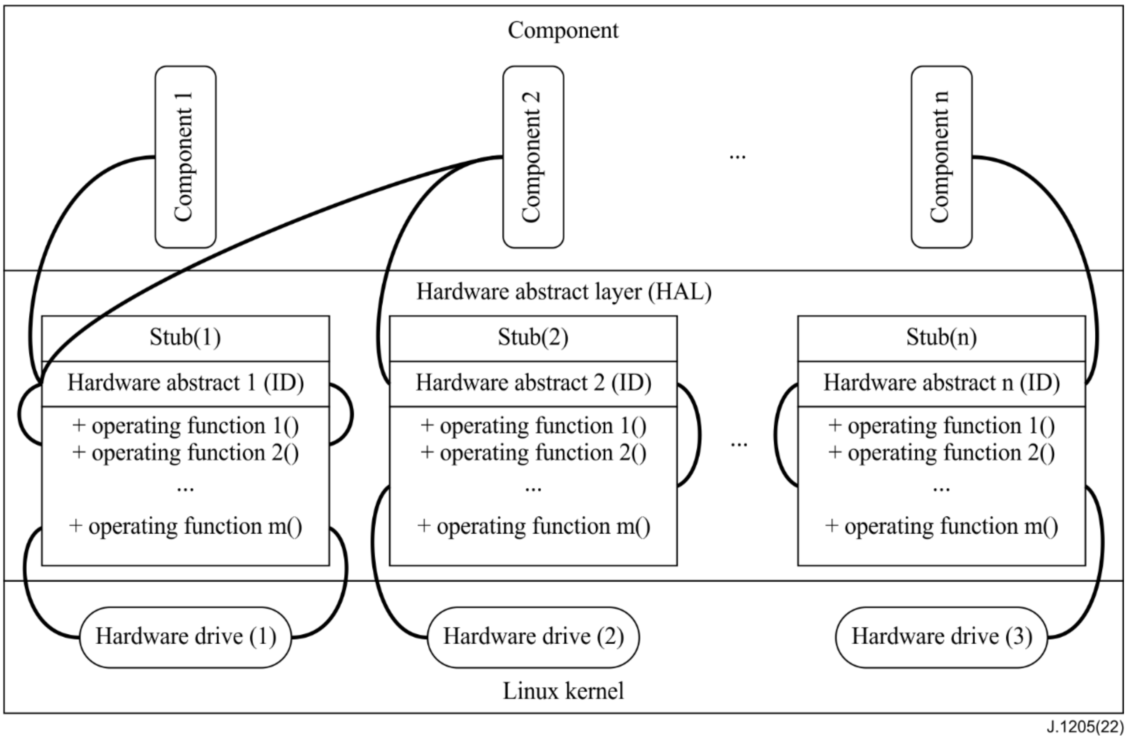

Figure 3 – Principle of Stub hardware abstract model

TVOS hardware abstract interface operates in user space, and abstracts the hardware drive operating in the kernel space. Stub is a concept of proxy and exists in the system in the form of a dynamic library, which provides a series of operating function interfaces for the upper-layer software. The upper-layer software just needs to possess the function pointer to access HAL Stub, without the need of owning the whole HAL Stub. The files in the dynamic library will only be mapped to one process. The upper-layer software acquires and operates HAL Stub through the uniform interface provided by HAL, invoking the operating function to control the hardware.

9 Description of hardware abstract interface

9.1 Special-purpose hardware abstract interface for media processing

9.1.1 Aout module

This clause defines the hardware abstract interface of the audio output module, and the brief list of Aout module interfaces is shown in Table 5.

Table 5 – Aout module interfaces

Interface	Description
aout_close	Close an audio output device
aout_open	Open an audio output device
*aout_init	Initialize Aout instance
*aout_term	Close Aout instance
*aout_get_capability	Get module equipment capability
*aout_open	Open an audio output instance
*aout_close	Close an audio output instance
*aout_set_volume	Set volume of output device
*aout_get_volume	Get volume of output device
*aout_set_digital_mode	Set audio output mode
*aout_get_digital_mode	Get audio output mode
*aout_set_mute	Set mute output
*aout_get_mute	Get mute output setting
*aout_set_channel_mode	Set channel mode. Set the channel for the device. See the definition in Table A.3 AOUT_DEVICE_TYPE_E for the device type, such as SPDIF, HDMI and loudspeaker output.
*aout_get_channel_mode	Set channel mode
*track_get_default_attr	Get default Track parameters
*track_create	Create a Track channel instance. The created Track is in the disabled state
*track_destroy	Destroy a Track channel instance
*track_start	Start Track channel, and Track audio data can be output from Aout
*track_stop	Stop Track channel, and the audio data cached in the channel are emptied
*track_pause	Pause Track channel, and Track audio data stop outputting from aout
*track_resume	Resume Track channel, and the audio data cached in the channel continue to output
*track_flush	Resume Track channel, and the audio data cached in the channel continue to output. Different from resume, flush will first empty the data in cache
*track_set_param	Set Track parameters. The parameters can only be set after the channel is in the halted state
*track_get_params	Get Track parameters
*track_set_mix_params	Set Track sound mixing parameters
*track_get_mix_params	Get Track sound mixing parameters
*track_set_mute	Set Track mute
*track_get_mute	Get Track mute

Table 5 – Aout module interfaces

Interface	Description
*track_set_channel_mode	Set channel mode for one-way track, such as stereo and left & right channel mixing. This does not conflict with the channel mode set in the channel, and it details one-way Track channel model in the channel mode of the channel
*track_get_channel_mode	Get channel mode
*track_set_weight	Set channel weight, such as linear volume or decibel volume
*track_get_weight	Get channel weight, such as linear volume or decibel volume
*track_get_render_position	Get audio frame number output to hardware
*track_get_pts	Get presentation time stamp (PTS) currently played in Track
*track_adjust_speed	Adjust speed of Track playing
*track_get_buf_avail	Get buffer space of Track available
*track_get_latency	Get latency of playable data in the buffer space of Track
*track_write	Write the number of audios to be played into the buffer space of Track. This function is a blocking operation. Prior to writing, be sure to confirm whether there is enough space through track_get_buf_avail, or else the operation will return immediately and fail

9.1.2 Demux module

This clause defines HAL interface of Demux module, and the brief list of Demux module interfaces is shown in Table 6.

Table 6 – Demux module interfaces

Interface	Description
demux_open	Open a Demux module device
demux_close	Close a Demux module device
*dmx_init	Initialize Demux module
*dmx_term	Terminate Demux module
*dmx_set_source_params	Set data source parameters of Demux module
*dmx_get_source_params	Get data source parameters of Demux module
*dmx_disconnect	Disconnect the correlation between Demux and data source (like modem), and they are correlated by default at the startup
*dmx_reconnect	Reconnect the correlation between Demux and the data source (like modem) set currently, and they are correlated by default at the startup
*dmx_get_capability	Get Demux capability
*dmx_get_status	Get Demux status
*dmx_channel_open	Open a channel
*dmx_channel_close	Close a channel and release relevant resources
*dmx_channel_set_pid	Set channel packet identifier (PID)
*dmx_channel_query	Query the channel corresponding to the PID through PID

Table 6 – Demux module interfaces

Interface	Description
*dmx_channel_enable	Enable the channel to receive data
*dmx_channel_disable	Disable the channel to receive data
*dmx_channel_reset	Reset channel
*dmx_channel_get_info	Get channel information
*dmx_channel_set	Set channel parameters
*dmx_channel_get	Get channel configuration
*dmx_channel_get_buf	Get data of the assigned channel
*dmx_channel_release_buf	Release buffer space occupied by the data packet
*dmx_channel_register_callback	Register callback function of corresponding channel
*dmx_channel_add_filter	Add a filter for the channel and set filtering data
*dmx_channel_set_filter	Set filtering conditions of the filter
*dmx_channel_get_filter	Get filtering conditions of the filter
*dmx_channel_destroy_filter	Destroy the filter
*dmx_channel_destroy_all_filter	Destroy all filters of the specified channel
*dmx_channel_enable_filter	Enable the specified filter of the channel to receive data
*dmx_channel_disable_filter	Disable the specified filter in the channel
*dmx_channel_query_filter_by_table_id	Query whether there exists a filter for certain table ID and extension ID
*dmx_channel_query_filter_by_filter_data	Query whether there exist the same filtering conditions for the channel filter
*dmx_descrambler_open	Open a descrambler channel
*dmx_descrambler_open_ex	Open the extension interface of a descrambler channel
*dmx_descrambler_enable	Enable a descrambler
*dmx_descrambler_disable	Disable a descrambler
*dmx_descrambler_close	Close a descrambler
*dmx_descrambler_associate	Associate the descrambler with PID or a channel to be descrambled
*dmx_descrambler_get_associate_info	Get the descrambled PID or channel information associated with the descrambler
*dmx_descrambler_set_even_key	Set even key of descrambler channel
*dmx_descrambler_set_even_iv	Set initial vector of data corresponding to the even key of descrambler channel
*dmx_descrambler_set_odd_key	Set odd key of descrambler channel
*dmx_descrambler_set_odd_iv	Set initial vector of data corresponding to the odd key of descrambler channel
*dmx_set_descrambler_attribute	Set attributes of descrambler channel
*dmx_get_descrambler_attribute	Get attributes of descrambler channel
*dmx_dcas_keyladder_config	Set descrambler configuration corresponding to downloadable conditional access system (DCAS) keyladder
*dmx_dcas_get_nonce	Get DA (Nonce)

Table 6 – Demux module interfaces

Interface	Description
*dmx_avfilter_open	Open a filter to filter AV data
*dmx_avfilter_enable	Enable the filter
*dmx_avfilter_get_esframe	Get frame data
*dmx_avfilter_release_esframe	Release frame data
*dmx_avfilter_disable	Disable the filter to filter audio video
*dmx_avfilter_close	Close the filter to filter audio video
*dmx_pcr_open	Open a PCR filtering channel
*dmx_pcr_close	Close a PCR filtering channel
*dmx_pcr_get	Get PCR
*dmx_tsbuffer_create	Create TS buffer to receive TS data input by network or locally
*dmx_tsbuffer_get	Create a TS buffer space
*dmx_tsbuffer_put	Used to update write pointer of TS data afterTS data input ends
*dmx_tsbuffer_destroy	Destroy the TS buffer space created
*dmx_get_streampath_param	Get streampath parameters required to set keyladder on trusted execution environment (TEE) side

9.1.3 Frontend module

This clause defines HAL interface of Frontend module, and the brief list of Frontend module interfaces is shown in Table 7.

Table 7 – Frontend module interfaces

Interface	Description
frontend_open	Open a Frontend module device
frontend_close	Close a Frontend module device
*frontend_init	Initialize Frontend
*frontend_term	Deinitialize a Frontend instance
*frontend_open	Open a Frontend instance
*frontend_close	Close a Frontend instance
*frontend_get_scan_info	Get current scanning information of Frontend
*frontend_sat_config_lnb	Set LNB
*frontend_get_lnb_pwr_status	Get power supply status of LNB
*frontend_start_scan	Start frequency locking or blind scanning
*frontend_abort	Abort frequency locking or blind scanning
*frontend_register_callback	Register callback function
*frontend_config_callback	Configure callback function
*frontend_lock	Synchronous frequency locking
*frontend_get_bert	Get bit error rate
*frontend_get_signal_quality	Get signal quality

Table 7 – Frontend module interfaces

Interface	Description
*frontend_get_signal_strength	Get signal strength
*frontend_get_atvsignalinfo	Get signal information
*frontend_get_connect_status	Get frequency locking information of signal
*frontend_get_info	Get all Frontend information
*frontend_get_capability	Get Frontend capability
*frontend_get_channel_num	Get channel number
*frontend_get_channel_info	Get channel information
*frontend_config_channel	Set channel informaiton
*frontend_atv_get_lock_status	Get frequency locking status
*frontend_atv_fineTune	Fine tune Tuner frequency

9.1.4 System module

This clause defines HAL interface of the system module, and the brief list of system module interfaces is shown in Table 8.

Table 8 – System module interfaces

Interface	Description
system_open	Open a system module device
system_close	Close a system module device
*system_init	System initialization
*system_term	System deinitialization
*system_switch_standby	Switch to standby mode interface
*system_get_chip_id	Get chip ID information
*system_sys_reboot	System reboot
*system_sys_halt	System halt

9.1.5 Vout module

This clause defines HAL interface of the video output module, and the brief list of Vout module interfaces is shown in Table 9.

Table 9 – Vout module interfaces

Interface	Description
vout_open	Open a video output module device
vout_close	Close a video output device
*vout_init	Initialize video output module
*vout_term	Terminate video output module
*vout_open_channel	Open a video output instance
*vout_close_channel	Close a video output instance
*vout_get_capability	Get module equipment capability

Table 9 – Vout module interfaces

Interface	Description
*vout_evt_config	Configure parameters of a Vout event
*vout_get_evt_config	Get configuration parameters of a Vout event
*vout_outputchannel_mute	Mute output function of output channel
*vout_outputchannel_unmute	Unmute output function of output channel
*vout_display_set	Set display parameters
*vout_display_get	Get display parameters
*vout_vbi_cgms_start	Start CGMS
*vout_vbi_cgms_stop	Stop CGMS
*vout_vbi_microvision_setup	Set up microvision image
*vout_vbi_microvision_enable	Enable/disable microvision image
*vout_set_hdcp_params	Set HDCP parameters
*vout_get_hdcp_status	Get HDCP checkout status
*vout_get_edid	Get original data of EDID
*vout_set_bg_color	Set background colour of video window
*vout_get_bg_color	Get background colour of video window
*vout_set_3dmode	Set 3D mode
*vout_get_3dmode	Get 3D mode of DISP
*vout_set_3d_lr_switch	Set right switch of 3D output
*vout_autodetect3dformat	Automatically detect 3D mode
*vout_window_create	Create a display window
*vout_window_destroy	Destroy a display window
*vout_window_set	Set window parameters
*vout_window_get	Get window parameters
*vout_window_set_input_rect	Set the rectangle of video input window
*vout_window_get_input_rect	Get the rectangle of video input window
*vout_window_set_output_rect	Set the rectangle of video output window
*vout_window_get_output_rect	Get the rectangle of video output window
*vout_window_set_video_rect	Set the rectangle of video content window
*vout_window_get_video_rect	Get the rectangle of video content window
*vout_window_get_status	Get video window status
*vout_window_freeze	Freeze video playing
*vout_window_unfreeze	Unfreeze video playing
*vout_window_mute	Mute window output
*vout_window_unmute	Unmute window output
*vout_window_set_mute_color	Set background colour after window output is muted
*vout_window_enable_filmmode	Enable or disable film mode
*vout_window_set_colortemperature	Set colour temperature of the window
*vout_window_set_zorder	Set up-down order of the window
*vout_window_enable_panorama	Set panorama mode

Table 9 – Vout module interfaces

Interface	Description
*vout_window_queue_frame	Queue video frame
*vout_window_dequeue_frame	Dequeue video display frame
*vout_window_reset	Reset display window
*vout_window_get_virtual_size	Get virtual window size
*vout_window_get_playinfo	Get play information
*vout_window_attach_input	Attach video input in the window
*vout_window_detach_input	Detach video input in the window

9.1.6 AV module

This clause defines HAL interface of the AV module, and the brief list of AV module interfaces is shown in Table 10.

Table 10 – AV module interfaces

Interface	Description
av_open	Open an AV module device
av_close	Close an AV module device
*av_init	Initialize AV module
*av_term	Terminate a player instance
*av_create	Create a player instance
*av_destroy	Destroy a player instance
*av_get_capability	Get AV player capability
*av_evt_config	Configure parameters of an AV event
*av_get_evt_config	Get parameters of an AV event
*av_get_status	Get AV playing status
*av_get_config	Get AV configuration
*av_set_config	Set AV configuration
*av_start	Start AV decoding
*av_stop	Stop AV decoding
*av_pause	Pause audio video decoding; live stream does not support such operation
*av_resume	Resume audio video decoding; live stream does not support such operation
*av_reset	Reset audio video buffer to a point in time
*av_start_video	Start video decoding
*av_pause_video	Pause video decoding
*av_freeze_video	Video decoding continues, without display update
*av_resume_video	Resume video decoding, display
*av_stop_video	Stop video decoding
*av_clear_video	Clear data in buffer displayed in the video

Table 10 – AV module interfaces

Interface	Description
*av_start_audio	Start audio decoding
*av_pause_audio	Pause audio decoding
*av_resume_audio	Resume audio decoding
*av_stop_audio	Stop audio decoding
*av_decode_iframe	Decode Iframe
*av_release_iframe	Release resources to decode an Iframe
*av_injecter_open	Open an injector instance
*av_injecter_close	Close an injector instance
*av_injecter_attach	Attach an injector to the decoder
*av_injecter_detach	Detach injector association
*av_inject_data	Inject data to be decoded
*av_inject_get_freebuf	Get address pointer of the free buffer to be injected in the buffer zone next time and the size of continuous free buffer
*av_inject_write_complete	Invoke av_inject_get_freebuf yo get free buffer add invoke this function for the decoder after copying the data
*av_inject_get_buf_status	Get buffer status of memory injection mode
*av_inject_get_setting	get injector parameter setting
*av_inject_reset_buf	Reset injection buffer
*av_inject_set_pcm_params	Set PCM parameters
*av_inject_set_es_params	Set ES parameters
*av_inject_es_data	Memory mode: inject ES data
*av_inject_abort	Abort data injection

9.2 General-purpose hardware abstract interface

9.2.1 OpenGL ES module

For further study based on [b-OpenGL ES 2.0.25].

9.2.2 OMX IL modules

For further study based on [b-OpenMAX IL 1.1.2].

Annex A

Aout module

(This annex forms an integral part of this Recommendation.)

This annex defines hardware abstract interface of the audio output module. The definitions of the basic data types and operators of the module are given in clause 6.

A.1 Definition of constants

The definition of constants is shown in Table A.1.

Table A.1 – Constant definition

Constant	Description
const AOUT_HARDWARE_MODULE_ID = "audio_output"	audio output module ID
const AOUT_HARDWARE_AOUT0 = "aout0"	audio output device ID
const AOUT_ID_NUM = "4"	Number of device IDs
const AOUT_VOL_DB_MAX = "12"	Maximum of decibel volume range of audio equipment
const AOUT_VOL_DB_ZERO = "0"	Zero of decibel volume range of audio equipment
const AOUT_VOL_DB_MIN = "-80"	Minimum of decibel volume range of audio equipment
const TRACK_VOL_MIN = "0"	Minimum of Track linear volume
const TRACK_VOL_MAX = "100"	Maximum of Track linear volume
const TRACK_VOL_DB_MAX = "0"	Maximum of Track linear volume
const TRACK_VOL_DB_MIN = "-70"	Minimum of Track linear volume

A.2 Enumeration definition

A.2.1 Definition of audio output

The enumeration definition of the audio output ID is shown in Table A.2.

Table A.2 – Audio output ID enumeration definition (AOUT_ID_E)

Type name	Value	Description
AOUT_ID_0	1	Audio output ID 0
AOUT_ID_1	1 << 1	Audio output ID 1
AOUT_ID_2	1 << 2	Audio output ID 2
AOUT_ID_3	1 << 3	Audio output ID 3
AOUT_ID_4	1 << 4	Audio output ID 4
AOUT_ID_5	1 << 5	Audio output ID 5
AOUT_ID_6	1 << 6	Audio output ID 6
AOUT_ID_7	1 << 7	Audio output ID 7
AOUT_ID_8	1 << 8	Audio output ID 8
AOUT_ID_9	1 << 9	Audio output ID 9

A.2.2 Definition of audio output device type

The enumeration definition of the audio output device type is shown in Table A.3.

Table A.3 – The enumeration definition of the audio output device type (AOUT_DEVICE_TYPE_E)

Type name	Value	Description
AOUT_DEVICE_NONE	0	Non-valid value
AOUT_DEVICE_RCA	1	Audio output of left and right channels
AOUT_DEVICE_SPDIF	1 << 1	Audio output of SPDIF
AOUT_DEVICE_HDMI	1 << 2	HDMI Audio output
AOUT_DEVICE_SPEAKER	1 << 3	Speaker output
AOUT_DEVICE_ARC	1 << 4	ARC output
AOUT_DEVICE_ALL	(U32) 0xffffffff	All output devices

A.2.3 Audio output mode

The audio output mode is shown in Table A.4.

Table A.4 – Audio output mode (AOUT_DIGITAL_OUTPUT_MODE_E)

Type name	Value	Description
AOUT_DIGITAL_OUTPUT_MODE_PCM	0	PCM decode output
AOUT_DIGITAL_OUTPUT_MODE_RAW	1	Source code output
AOUT_DIGITAL_OUTPUT_MODE_AUTO	2	Automatic output, negotiated according to HDMI EDID, priority HBR(DD+/DTSHD) >LBR(DD/DTS) > PCM

A.2.4 Audio data format

The audio data format is shown in Table A.5.

Table A.5 – The audio data format (AOUT_DATA_FORMAT_E)

Type name	Value	Description
AOUT_DATA_FORMAT_LE_PCM_16_BIT	0	16 bits linear PCM, using little-endian byte order
AOUT_DATA_FORMAT_EXTENSIONS	(U32)0x10000000	Reserved for subsequent extension
AOUT_DATA_FORMAT_BUTT	(U32)0x7fffffff	The enumeration maximum of Audio data format

A.2.5 Channel mode

The Channel mode is shown in Table A.6.

Table A.6 – The Channel mode (AOUT_CHANNEL_MODE_E)

Type name	Value	Description
TRACK_CHANNEL_MONO	0	Mixed sound
TRACK_CHANNEL_RIGHT	1	Right channel
TRACK_CHANNEL_LEFT	2	Left channel
TRACK_CHANNEL_STER	3	Stereo

A.2.6 Audio channel mode enumeration

The audio channel mode enumeration is shown in Table A.7.

Table A.7 – Audio channel mode enumeration (TRACK_CHANNEL_MODE_E)

Type name	Value	Description
TRACK_MODE_STEREO	0	Stereo
TRACK_MODE_DOUBLE_MONO	1	Output after mixing of left and right channels
TRACK_MODE_DOUBLE_LEFT	2	Left and right channels output left channel data
TRACK_MODE_DOUBLE_RIGHT	3	Left and right channels output right channel data
TRACK_MODE_EXCHANGE	4	Left and right channels data exchange output
TRACK_MODE_ONLY_RIGHT	5	Only output right channel data
TRACK_MODE_ONLY_LEFT	6	Only output left channel data
TRACK_MODE_MUTED	7	Mute
TRACK_MODE_BUTT	8	Audio channel mode enumeration maximum

A.3 Data structure definition

The data structure involved in the audio output module includes:

A.3.1 Setting of the channel weight parameter structure

Setting of the channel weight parameter structure is shown in Table A.8.

Table A.8 – Setting of the channel weight parameter structure (TRACK_GAIN_ATTR_S)

Attribute name	Type	Description
bLinearMode	BOOL	Volume mode, true means linear volume, false means decibel volume
s32Gain	S32	Linear volume: 0～100; decibel volume: -70～0

A.3.2 Aout Track create parameter structure

Aout Track create parameter structure is shown in Table A.9.

Table A.9 – Aout Track create parameter structure (AOUT_TRACK_PARAMS_S)

Attribute name	Type	Description
enFormat	Enumerated value	Data format, see Table A.5, AOUT_DATA_FORMAT_E definition
bPtsSync	BOOL	Whether PTS synchronization is supported
u32SampleRate	U32	Sampling frequency
u32Channels	U32	Soundtracks
u32BufferSize	U32	Data buffer size

A.3.3 Aout Track mixing parameter structure

Aout Track mixing parameter structure is shown in Table A.10.

Table A.10 – Aout Track mixing parameter structure (AOUT_TRACK_MIX_PARAMS_S)

Attribute name	Type	Description
s32IntGain	S32	The integer part of the mixing volume value. The range thereof is shown in Table A.1 [AOUT_VOL_DB_MIN, AOUT_VOL_DB_MAX], step size (1.0dB)
s32DecGain	S32	The decimal part of the mixing volume value, range (0～7), step size (0.125dB)
u32Dummy	U32	Reserved for future use

A.3.4 Audio output module initialization parameter structure

Audio output module initialization parameter structure is shown in Table A.11.

Table A.11 – Audio output module initialization parameter structure (AOUT_INIT_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved for future use

A.3.5 Audio output configuration parameter structure

Audio output configuration parameter structure is shown in Table A.12.

Table A.12 – Audio output configuration parameter structure (AOUT_SETTINGS_S)

Attribute name	Type	Description
enOutputDevice	Enumerated value	The enumeration definition is shown in Table A.3. AOUT_DEVICE_TYPE_E
u32Dummy	U32	Reserved for future use

A.3.6 Audio output module capability structure

Audio output module capability structure is shown in Table A.13.

Table A.13 – Audio output module capability structure (AOUT_CAPABILITY_S)

Attribute name	Type	Description
u8SupportedIdNum	U8	The maximum number of audio devices supported
enOutputDevice [AOUT_ID_NUM]	Structure	Audio output device is shown in Table A.3 AOUT_DEVICE_TYPE_E, definition
u8SupportedTrackNum [AOUT_ID_NUM]	U8	The maximum number of Track channels

A.3.7 Aout module termination parameter structure

Aout Module termination parameter structure is shown in Table A.14.

Table A.14 – Aout module termination parameter structure (AOUT_TERM_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved for future use

A.3.8 Audio output example open parameter structure

Audio output example open parameter structure is shown in Table A.15.

Table A.15 – Audio output example open parameter structure (AOUT_OPEN_PARAMS_S)

Attribute name	Type	Description
enId	Enumerated value	Audio output ID is shown in Table A.2, AOUT_ID_E, definition of constants
stSettings	Structure	Audio output settings are shown in Table A.12 AOUT_SETTINGS_S, definition of structure

A.3.9 Audio output example closing parameter structure

Audio output example closing parameter structure is shown in Table A.16.

Table A.16 – Audio output example closing parameter structure (AOUT_CLOSE_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved for future use

A.3.10 Audio output module structure

Audio output module structure is shown in Table A.17.

Table A.17 – Audio output module structure (AOUT_MODULE_S)

Attribute name	Type	Description
stCommon	Structure	See Table B.41, hw_module_t Structure definition

A.4 Definition of Callback function

None

A.5 Calling method

The hardware abstract interface calling method of the Aout module is shown in Figure A.1.

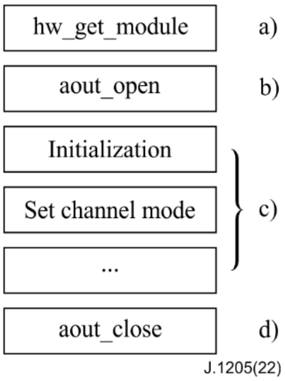

Figure A.1 – Aout module hardware abstract interface calling method

Figure A.1:

a) Call the hw_get_module() interface to get the HAL Stub of the Aout module：
hw_get_module(AOUT_HARDWARE_MODULE_ID, &g_aout_module).

b) Call aout_open(g_aout_module,&pstDevice) to get the device handle of the Aout module:
aout_open (g_aout_mod ule,&pstDevice).

c) Control the Aout hardware through a series of interface functions provided by the device handle.

d) After finishing hardware manipulation, call the aout_close () interface to close the Aout device to avoid resource leakage.

A.6 Definition of interfaces

A.6.1 "Close the Aout device" interface

The prototype: static inline int aout_close (AOUT_DEVICE_S* pstDevice);

Function: Turn off an audio output device.

Input parameter: pstDevice: The handle of the audio output device.

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.2 "Open the Aout device" interface

The prototype: static inline int aout_open (const struct hw_module_t* pstModule, AOUT_DEVICE_S** ppstDevice);

Function: Turn on an audio output device.

Input parameter: pstModule Aout module handle.

Output parameter: ppstDevice audio output device handle.

Return: 0: correct; non-zero: error.

A.6.3 "Initialize the Aout device" interface

The prototype: S32 (*aout_init) (struct _AOUT_DEVICE_S* pstDev, const AOUT_INIT_PARAMS_S* pstInitParams);

Function: Initialize the Aout instance. The other functions cannot run correctly before this function is called.

Input parameter: pstDev Aout device handle

pstInitParams initialization parameters.

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.4 "Terminate the Aout device" interface

The prototype S32 (*aout_term) (struct _AOUT_DEVICE_S* pstDev, const AOUT_TERM_PARAMS_S* pstTermParams);

Function: Close the Aout instance

Input parameter: pstDev Aout device handle

pstTermParams Terminate module parameters. This parameter is currently left unused for the sake of subsequent extension.

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.5 "Get Aout device capability" interface

The prototype: S32 (*aout_get_capability)(struct _AOUT_DEVICE_S* pstDev, AOUT_CAPABILITY_S* pstCapability);

Function: Get module device capabilities

Input parameter: pstDev Aout device handle

Output parameters: pstCapability Audio device capability

Return: 0: correct; non-zero: error.

A.6.7 "Open the Aout device instance" interface

The prototype: S32 (*aout_open)(struct _AOUT_DEVICE_S* pstDev, HANDLE* phAout, const AOUT_OPEN_PARAMS_S* pstOpenParams);

Function: Open an audio output instance

Input parameter: pstDev Aout device handle

pstOpenParams Parameters when opening the audio instance

Output parameters: phAout Audio instance handle

Return: 0: correct; non-zero: error.

A.6.8 "Close the Aout device instance" interface

The prototype: S32 (*aout_close)(struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, const AOUT_CLOSE_PARAMS_S* pstCloseParams);

Function: Close an audio output instance

Input parameter: pstDev Aout device handle

hAout Audio instance handle

pstCloseParams Parameters when the audio instance handle is closed. Currently it is unused for future extension.

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.9 "Set the volume of the Aout device" interface

The prototype: S32 (*aout_set_volume) (struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, AOUT_DEVICE_TYPE_E enOutputDevice, S32 s32IntGain, S32 s32DecGain);

Function: Set the output device volume

Input parameter: pstDev Aout device handle

hAout Audio instance handle

enOutputDevice Audio output device type, such as left and right channels, etc.

s32IntGain The integer part of the value, value range [AOUT_VOL_DB_MIN, AOUT_VOL_DB_MAX], step size 1db

s32DecGain The decimal part of the volume value, value range (0~7), step size (0.125 dB)

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.10 "Get Aout device volume" interface

The prototype: S32 (*aout_get_volume) (struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, AOUT_DEVICE_TYPE_E enOutputDevice, S32* ps32IntGain, S32* ps32DecGain);

Function: Get output device volume

Input parameter: pstDev Aout device handle

hAout Audio instance handle

enOutputDevice Audio output device type, such as left and right channels, etc.

Output parameters: ps32IntGain The integer part of the value, the value range [AOUT_VOL_DB_MIN, AOUT_VOL_DB_MAX], the step size is 1db

ps32DecGain The decimal part of the volume value, the value range (0~7), step size is (0.125 dB)

Return: 0: correct; non-zero: error.

A.6.11 "Set the Aout device output mode" interface

The prototype: S32 (*aout_set_digital_mode)(struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, AOUT_DEVICE_TYPE_E enOutputDevice, AOUT_DIGITAL_OUTPUT_MODE_E enMode);

Function: Set audio output mode.

Input parameter: pstDev Aout device handle

hAout Audio instance handle

enOutputDevice Audio output device type, such as left and right channels, etc.

enMode Audio output mode, is shown in Table A.4 AOUT_DIGITAL_OUTPUT_MODE_E enumeration definition for values.

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.12 "Get Aout device output mode" interface

The prototype: S32 (*aout_get_digital_mode) (struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, AOUT_DEVICE_TYPE_E enOutputDevice, AOUT_DIGITAL_OUTPUT_MODE_E* penMode)

Function: Get audio output mode

Input parameter: pstDev Aout device handle

hAout Audio instance handle

enOutputDevice Audio output device type

Output parameters: penMode Audio output mode

Return: 0: correct; non-zero: error.

A.6.13 "Set of Aout device muted" interface

The prototype: S32 (*aout_set_mute) (struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, AOUT_DEVICE_TYPE_E enOutputDevice, BOOL bMute);

Function: Set output mute.

Input parameter: pstDev Aout device handle

hAout Audio instance handle

enOutputDevice Audio output device type, such as left and right channels, etc.

bMute true mute; false, not mute.

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.14 "Get mute attribute of Aout" interface

The prototype: S32 (*aout_get_mute) (struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, AOUT_DEVICE_TYPE_E enOutputDevice, BOOL* pbMute);

Function: Get output mute setting

Input parameter: pstDev Aout device handle

hAout Audio instance handle

enOutputDevice Audio output device type, such as left and right channels, etc.

Output parameters: pbMute true, muted; false, not muted.

Return: 0: correct; non-zero: error.

A.6.15 "Set the Aout channel mode" interface

The prototype: S32 (*aout_set_channel_mode) (struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, AOUT_DEVICE_TYPE_E enOutputDevice, AOUT_CHANNEL_MODE_E enMode);

Function: Set the channel mode. Set the sound channel at the device level. The device type is defined in Table A.3 AOUT_DEVICE_TYPE_E, such as SPDIF, HDMI, speaker output, etc.

Input parameter: pstDev Aout device handle

hAout Audio instance handle

enOutputDevice Audio output device type, such as left and right channels, etc.

enMode Channel mode, such as left channel, right channel, stereo, etc.

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.16 "Get the Aout channel mode" interface

The prototype: S32 (*aout_get_channel_mode) (struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, AOUT_DEVICE_TYPE_E enOutputDevice, AOUT_CHANNEL_MODE_E* penMode);

Function: Set channel mode

Input parameter: pstDev Aout device handle

hAout Audio instance handle

enOutputDevice Audio output device type, such as left and right channels, etc.

Output parameters: penMode Channel mode, such as left channel, right channel, stereo, etc.

Return: 0: correct; non-zero: error.

A.6.17 "Get Aout Track default attribute" interface

The prototype: S32 (*track_get_default_attr) (struct _AOUT_DEVICE_S* pstDev, AOUT_TRACK_PARAMS_S* pstParams);

Function: Get the default Track parameters.

Input parameter: pstDev Aout device handle

Output parameters: pstParams Track parameters

Return: 0: correct; non-zero: error.

A.6.18 "Create Track channel instance" interface

The prototype: S32 (*track_create) (struct _AOUT_DEVICE_S* pstDev, HANDLE hAout, HANDLE* phTrack, const AOUT_TRACK_PARAMS_S* pstParams);

Function: Create a Track channel instance, and the created Track is in the disabled state.

a) The audio of multiple tracks are mixed and output to the same aout's enOutputDevice (RCA/SPDIF/HDMI, etc.).

b) Source and Track are bound to the scenario (attach), Source can dynamically set Track's AOUT_TRACK_PARAMS_S

c) For playback scenario (non-attach), AOUT_TRACK_PARAMS_S of Track must be set correctly, if you need to modify parameters, you must stop Track first

Input parameter: pstDev Aout device handle

hAout Audio instance handle

pstParams Track creation parameters

Output parameters: phTrack Track handle

Return: 0: correct; non-zero: error.

A.6.19 "Delete the Track channel instance" interface

The prototype: S32 (*track_destroy) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack);

Function: Delete a Track channel instance.

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.20 "Start Track channel" interface

The prototype: S32 (*track_start) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack);

Function: Start the Track channel, Track audio data can be output from aout

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.21 "Stop Track Channel" interface

The prototype: S32 (*track_stop) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack)

Function: Stop the Track channel. The audio data in the channel buffer will be cleared

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.22 "Pause Track channel" interface

The prototype: S32 (*track_pause) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack)

Function: Pause the Track channel, and stop the output of Track audio data from aout

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.23 "Restore Track Channel" interface

The prototype: S32 (*track_resume) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack)

Function: Restore the Track channel. The audio data buffered by the channel will continue to be output

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.24 "Flush Track channel" interface

The prototype: S32 (*track_flush) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack)

Function: Restore the Track channel. The difference from "resume" is that "flush" will first clear the data in the buffer.

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.25 "Set Track attribute" interface

The prototype: S32 (*track_set_params) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, const AOUT_TRACK_PARAMS_S* pstParams);

Function: Set Track attributes. The attributes can be set only after the channel is in the stop state

Input parameter: pstDev Aout device handle

hTrack Track handle

pstParams Track parameters

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.26 "Get Track attribute" interface

The prototype: S32 (*track_get_params) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, AOUT_TRACK_PARAMS_S* pstParams)

Function: Get Track attributes

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: pstParams Track parameters

Return: 0: correct; non-zero: error.

A.6.27 "Set mixing parameter" interface

The prototype: S32 (*track_set_mix_params) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, const AOUT_TRACK_MIX_PARAMS_S* pstParams);

Function: Set Track mixing parameters.

Input parameter: pstDev Aout device handle

hTrack Track handle

pstParams Mixing parameters

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.28 "Get Track mixing parameters" interface

The prototype: S32 (*track_get_mix_params) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, AOUT_TRACK_MIX_PARAMS_S* pstParams)

Function: Get Track mixing parameters

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: pstParams Mixing parameters

Return: 0: correct; non-zero: error.

A.6.29 "Set Track mute" interface

The prototype: S32 (*track_set_mute) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, BOOL bMute)

Function: Set Track to mute

Input parameter: pstDev Aout device handle

hTrack Track handle

bMute true, muted; false, not muted

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.30 "Get Track mute" interface

The prototype: S32 (*track_get_mute) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, BOOL *pbMute)

Function: Get Track Mute

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: pbMute true, muted; false, not muted

Return: 0: correct; non-zero: error.

A.6.31 "Set channel mode" interface

The prototype: S32 (*track_set_channel_mode) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, TRACK_CHANNEL_MODE_E enMode);

Function: Set the channel mode for the track, such as stereo, mixing of left and right channels, etc., which does not conflict with the set channel mode in the Channel. It is a refinement of a Track mode in the Channel channel mode.

Input parameter: pstDev Aout device handle

hTrack Track handle

enMode Audio channel mode

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.32 "Get audio channel mode" interface

The prototype: S32 (*track_get_channel_mode) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, TRACK_CHANNEL_MODE_E *penMode)

Function: Get channel mode

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: penMode Audio channel mode

Return: 0: correct; non-zero: error.

A.6.33 "Set channel weight" interface

The prototype: S32 (*track_set_weight) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, const TRACK_GAIN_ATTR_S stTrackGainAttr);

Function: Set the channel weight based on, for example, linear volume or decibel volume

Input parameter: pstDev Aout device handle

hTrack Track handle

stTrackGainAttr Channel weight

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.34 "Get channel weight" interface

The prototype: S32 (*track_get_weight) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, TRACK_GAIN_ATTR_S* pstTrackGainAttr);

Function: Get the channel weight based on, for example, linear volume or decibel volume.

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: pstTrackGainAttr Channel weight

Return: 0: correct; non-zero: error.

A.6.35 "Get the number of audio frames that has been output to the hardware" interface

The prototype: S32 (*track_get_render_position) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, U32* pu32Frames);

Function: Get the number of audio frames that have been output to the hardware

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: pu32Frames The number of audio frames that have been output to the hardware

Return: 0: correct; non-zero: error.

A.6.36 "Get the PTS currently being played on Track" interface

The prototype: S32 (*track_get_pts) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, S64* ps64Pts

Function: Get the PTS currently being played on Track

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: ps64Pts Return PTS size

Return: 0: correct; non-zero: error.

A.6.37 "Speed control for Track playback" interface

The prototype: S32 (*track_adjust_speed) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, S32 s32Speed)

Function: Adjust the speed of Track playback

Input parameter: pstDev Aout device handle

hTrack Track handle

s32Speed Speed (-100~100) ms

Output parameters: None.

Return: 0: correct; non-zero: error.

A.6.38 "Get remaining space of Track cache" interface

The prototype: S32 (*track_get_buf_avail) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, U32* pu32Bytes);

Function: Obtain the remaining space of Track cache

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: pu32Bytes Return the number of remaining bytes

Return: 0: correct; non-zero: error.

A.6.39 "Get duration of Track cache playable data" interface

The prototype: S32 (*track_get_latency) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, U32* pu32Latency)

Function: Get duration of Track cache playable data

Input parameter: pstDev Aout device handle

hTrack Track handle

Output parameters: pu32Latency Return to play time

Return: 0: correct; non-zero: error.

A.6.40 "Write data to Track" interface

The prototype: S32 (*track_write) (struct _AOUT_DEVICE_S* pstDev, HANDLE hTrack, const void* pvBuffer, U32 u32Bytes)

Function: Write the number of audios to be played into the Track cache. This function is a blocking operation. You must confirm whether there is enough space through track_get_buf_avail before writing, otherwise it will immediately return "Fail".

Input parameter: pstDev Aout device handle

hTrack Track handle

pvBuffer Data pointer to be written

u32Bytes Length of data to be written

Output parameters: None.

Return: 0: correct; non-zero: error.

Annex B

Demux module

(This annex forms an integral part of this Recommendation.)

This annex defines the hardware abstraction layer interfaces of the Demux module. The definitions of the basic data types and operators are given in clause 6.

B.1 Constant definition

The definition of constants is shown in Table B.1.

Table B.1 – Constant definition

Constant	Description
const DEMUX_HARDWARE_MODULE_ID = "dmx"	Demux module ID
const DEMUX_HARDWARE_DEMUX0 = "demux0"	Demux device ID
const DEMUX_HEADER_VERSION = "1"	Demux version number
const DMX_NUMBER_OF_DMX_ID = "24"	Demux device ID number
const DMX_FILE_NAME_LENGTH = "255"	Maximum length of file name
const DMX_CHANNEL_CALLBACK_MAX = "8"	The maximum number of callback functions registered for each channel

B.2 Enumeration definition

B.2.1 Enumeration definition of Channel type

Enumeration definition of Channel type is shown in Table B.2.

Table B.2 – Enumeration definition of Channel type (DMX_CHANNEL_TYPE_E)

Type name	Value	Description
DMX_VIDEO_CHANNEL	0	Demux video channel, only output video ES data
DMX_AUDIO_CHANNEL	1	Demux audio channel, only output audio ES data
DMX_PES_CHANNEL	2	Demux PES data filtering channel
DMX_SECTION_CHANNEL	3	Demux PES data filtering channel
DMX_POST_CHANNEL	4	Demux whole package upload channel, configured to receive complete TS packages of a certain PID
DMX_PCR_CHANNEL	5	Demux PCR packet upload channel, configured to receive PCR TS packets of a certain PID
DMX_CHANNEL_BUTT	6	Demux channel type enumeration maximum

B.2.2 Enumeration definition of Descrambler type

Enumeration definition of Descrambler type is shown in Table B.3.

Table B.3 – Enumeration definition of Descrambler type (DMX_DESCRAMBLER_TYPE_E)

Type name	Value	Description
DMX_CA_NORMAL_DESCRAMBLER	0	Ordinary certification authority (CA) descrambling
DMX_CA_ADVANCE_DESCRAMBLER	1	High security CA descrambling
DMX_CA_BUTT	2	Enumeration maximum of descrambler type

B.2.3 Enumeration definition of Demux device ID

Enumeration definition of Demux device ID is shown in Table B.4.

Table B.4 – Enumeration definition of Demux device ID (DMX_ID_E)

Type name	Value	Description
DMX_ID_0	0	Demux ID 0
DMX_ID_1	1	Demux ID 1
DMX_ID_2	2	Demux ID 2
DMX_ID_3	3	Demux ID 3
DMX_ID_4	4	Demux ID 4
DMX_ID_5	5	Demux ID 5
DMX_ID_6	6	Demux ID 6
DMX_ID_7	7	Demux ID 7
DMX_ID_8	8	Demux ID 8
DMX_ID_9	9	Demux ID 9
DMX_ID_10	10	Demux ID 10
DMX_ID_11	11	Demux ID 11
DMX_ID_12	12	Demux ID 12
DMX_ID_13	13	Demux ID 13
DMX_ID_14	14	Demux ID 14
DMX_ID_15	15	Demux ID 15
DMX_ID_16	16	Demux ID 16
DMX_ID_17	17	Demux ID 17
DMX_ID_18	18	Demux ID 18
DMX_ID_19	19	Demux ID 19
DMX_ID_20	20	Demux ID 20
DMX_ID_21	21	Demux ID 21
DMX_ID_22	22	Demux ID 22
DMX_ID_23	23	Demux ID 23
DMX_ID_BUTT	24	Enumeration maximum of Demux device ID

B.2.4 Enumeration definition of Demux event

Enumeration definition of Demux event is shown in Table B.5.

Table B.5 – Enumeration definition of Demux event (DMX_EVT_E)

Type name	Value	Description
DMX_ALL_EVT	0	All events
DMX_EVT_DATA_AVAILABLE	1	Available data event
DMX_EVT_DATA_ERROR	2	Data error event
DMX_EVT_CRC_ERROR	4	CRC check failure event
DMX_EVT_BUF_OVERFLOW	8	Cache overflow
DMX_EVT_SCRAMBLED_ERROR	16	Scrambling error
DMX_EVT_CHANNEL_TIMEOUT	32	Channel timeout
DMX_EVT_BUTT	33	Demux event defines the maximum value of the enumeration

B.2.5 Enumeration definition of Notification function call type

Enumeration definition of Notification function call type is shown in Table B.6.

Table B.6 – Enumeration definition of Notification function call type (DMX_NOTIFY_TYPE_E)

Type name	Value	Description
DMX_NOTIFY_DATA	0	The notification function will be accompanied by data
DMX_NOTIFY_NODATA	1	The notification function will not be accompanied by data
DMX_NOTIFY_BUTT	2	Notification function call type enumeration maximum

B.2.6 Enumeration definition of Channel status

Enumeration definition of Channel status is shown in Table B.7.

Table B.7 – Enumeration definition of Channel status (DMX_CHANNEL_STATUS_E)

Type name	Value	Description
DMX_CHANNEL_ENABLE	0	Channel enable
DMX_CHANNEL_DISABLE	1	Channel disable
DMX_CHANNEL_RESET	2	Channel reset

B.2.7 Enumeration definition of Demux callback function operation type

Enumeration definition of Demux callback function operation type is shown in Table B.8.

Table B.8 – Enumeration definition of Demux callback function operation type (DMX_CFG_CALLBACK_E)

Type name	Value	Description
DMX_CALLBACK_ENABLE	0	Open callback function
DMX_CALLBACK_DISABLE	1	Close callback function
DMX_CALLBACK_REMOVE	2	Delete callback function

B.2.8 Enumeration definition of Demux report data method

Enumeration definition of Demux report data method is shown in Table B.9.

Table B.9 – Enumeration definition of Demux report data method (DMX_FILTER_REPEAT_MODE_E)

Type name	Value	Description
DMX_FILTER_REPEAT_MODE_REPEATED	0	Report data repeatedly
DMX_FILTER_REPEAT_MODE_ONE_SHOT	1	Only report data once
DMX_FILTER_REPEAT_MODE_BUTT	2	The enumeration maximum of Demux report data method

B.2.9 Enumeration definition of Descrambler associated type

Enumeration definition of Descrambler associated type is shown in Table B.10.

Table B.9 – Enumeration definition of Descrambler associated type (DMX_DESC_ASSOCIATE_MODE_E)

Type name	Value	Description
DMX_DESCRAMBLER_ASSOCIATE_WITH_PIDS	0	Associate with PID
DMX_DESCRAMBLER_ASSOCIATE_WITH_CHANNEL	1	Associate with channel
DMX_DESCRAMBLER_ASSOCIATE_BUTT	2	The enumeration maximum value of Descrambler associated type

B.2.10 Enumeration definition of Descrambling encryption type

Enumeration definition of Descrambling encryption type is shown in Table B.11.

Table B.11 – Enumeration definition of Descrambling encryption type (DMX_DESC_TYPE_E)

Type name	Value	Description
DMX_DESC_TYPE_CSA2	0	CAS2.0 encryption algorithm
DMX_DESC_TYPE_CSA3	1	CAS3.0 encryption algorithm
DMX_DESC_TYPE_DES_CI	2	DES CIPLUS encryption algorithm
DMX_DESC_TYPE_DES_CBC	3	DES CBC encryption algorithm
DMX_DESC_TYPE_DES_IPTV	4	DES IPTV encryption algorithm
DMX_DESC_TYPE_TDES_ECB	5	TDES ECB encryption algorithm

Table B.11 – Enumeration definition of Descrambling encryption type (DMX_DESC_TYPE_E)

Type name	Value	Description
DMX_DESC_TYPE_TDES_CBC	6	TDES CBC encryption Algorithm
DMX_DESC_TYPE_TDES_IPTV	7	TDES IPTV encryption algorithm
DMX_DESC_TYPE_AES_ECB	8	Spe Aes Ecb encryption algorithm
DMX_DESC_TYPE_AES_CBC	9	AES CBC encryption algorithm
DMX_DESC_TYPE_AES_IPTV	10	AES IPTV of SPE encryption algorithm
DMX_DESC_TYPE_AES_NS	11	AES NS-Mode encryption algorithm
DMX_DESC_TYPE_AES_CI	12	SPE AES CIPLUS encryption algorithm
DMX_DESC_TYPE_SMS4_ECB	13	SMS4 ECB encryption Algorithm
DMX_DESC_TYPE_SMS4_CBC	14	SMS4 CBC encryption algorithm
DMX_DESC_TYPE_SMS4_IPTV	15	SMS4 IPTV encryption algorithm
DMX_DESC_TYPE_SMS4_NS	16	SMS4 NS-Mode encryption algorithm
DMX_DESC_TYPE_BUTT	17	The enumeration maximum value of descrambling encryption type

B.2.11 Enumeration definition of Entropy reduction mode

Enumeration definition of Entropy reduction mode is shown in Table B.12.

Table B.12 – Enumeration definition of Entropy reduction mode (DMX_CA_ENTROPY_E)

Type name	Value	Description
DMX_CA_ENTROPY_REDUCTION_CLOSE	0	Entropy reduction turned off
DMX_CA_ENTROPY_REDUCTION_OPEN	1	Entropy reduction turned on
DMX_CA_ENTROPY_REDUCTION_BUTT	2	The enumeration maximum value of Entropy reduction mode

B.2.12 Enumeration definition of Descrambling style

Enumeration definition of Descrambling style is shown in Table B.13.

Table B.13 – Enumeration definition of Descrambling style (DMX_DESC_STYLE_E)

Type name	Value	Description
DMX_DESC_STYLE_NONE	0	Does not support descrambling
DMX_DESC_STYLE_SOLE	1	The same PID can only descramble one way
DMX_DESC_STYLE_SHARE	2	The same PID, one descrambler can descramble multiple channels at the same time, but cannot open multiple descramblers
DMX_DESC_STYLE_BIUNIQUE	3	The same PID, multiple descramblers are descrambling multiple channels at the same time. The descramblers are one to one with channels
DMX_DESC_STYLE_BUTT	4	Enumerated maximum value of descrambling type

B.2.13 Enumeration definition of Data source type

Enumeration definition of Data source type is shown in Table B.14.

Table B.14 – Enumeration definition of Data source type (DMX_SOURCE_TYPE_E)

Type name	Value	Description
DMX_SOURCE_TUNER	0	Tuner From Tuner
DMX_SOURCE_FILE	1	From file
DMX_SOURCE_MEM	2	From memory
DMX_SOURCE_PVR	3	From PVR
DMX_SOURCE_NONE	4	Free Demux
DMX_SOURCE_BUTT	5	Enumeration maximum value of data source type

B.2.14 Enumeration definition of CA encryption algorithm

Enumeration definition of CA encryption algorithm is shown in Table B.15.

Table B.15 – Enumeration definition of CA encryption algorithm (DMX_KL_ALG_E)

Type name	Value	Description
DMX_KL_ALG_TDES	0	3DES algorithm
DMX_KL_ALG_AES	1	AES algorithm
DMX_KL_ALG_BUTT	2	Enumerated maximum value of CA encryption algorithm

B.2.15 Enumeration definition of getting Demux channel data packet structure

Enumeration definition of getting Demux channel is shown in Table B.16.

Table B.16 – Enumeration definition of getting Demux channel (DMX_CHANNEL_DATA_TYPE_E)

Type name	Value	Description
DMX_DATA_TYPE_WHOLE	0	This section of data contains complete data packets, and each packet of SECTION is complete
DMX_DATA_TYPE_HEAD	1	This piece of data contains the start of the data packet, but it is not necessarily a complete packet and is only used for PES data
DMX_DATA_TYPE_BODY	2	This piece of data contains the content of the data packet, does not contain the start, may have a node, and is only used for PES data
DMX_DATA_TYPE_TAIL	3	This section of data contains the end of the data packet, which is configured to indicate the end of the identifiable packet and is only used for PES data
DMX_DATA_TYPE_BUTT	4	Get the enumerated maximum value of the Demux channel data packet structure

B.2.16 Enumeration definition of Video stream type

Enumeration definition of Video stream type is shown in Table B.17.

Table B.17 – Enumeration definition of Video stream type (DMX_VID_STREAM_TYPE_E)

Type name	Value	Description
DMX_VID_STREAM_TYPE_UNKNOWN	-1	Unknown format
DMX_VID_STREAM_TYPE_MPEG2	0	MPEG2 format
DMX_VID_STREAM_TYPE_MPEG4	1	MPEG4 DIVX4 DIVX5 format
DMX_VID_STREAM_TYPE_AVS	2	AVS format
DMX_VID_STREAM_TYPE_AVSPLUS	3	AVS+ format
DMX_VID_STREAM_TYPE_H263	4	H263 format
DMX_VID_STREAM_TYPE_H264	5	H264 format
DMX_VID_STREAM_TYPE_REAL8	6	REAL8 format
DMX_VID_STREAM_TYPE_REAL9	7	REAL9 format
DMX_VID_STREAM_TYPE_VC1	8	VC1format
DMX_VID_STREAM_TYPE_VP6	9	VP6 format
DMX_VID_STREAM_TYPE_VP6F	10	VP6F format
DMX_VID_STREAM_TYPE_VP6A	11	VP6A format
DMX_VID_STREAM_TYPE_MJPEG	12	MJPEG format

Table B.17 – Enumeration definition of Video stream type (DMX_VID_STREAM_TYPE_E)

Type name	Value	Description
DMX_VID_STREAM_TYPE_SORENSON	13	SORENSON format
DMX_VID_STREAM_TYPE_DIVX3	14	DIVX3 format
DMX_VID_STREAM_TYPE_RAW	15	RAW format
DMX_VID_STREAM_TYPE_JPEG	16	JPEG format
DMX_VID_STREAM_TYPE_VP8	17	VP8 format
DMX_VID_STREAM_TYPE_VP9	18	VP9 format
DMX_VID_STREAM_TYPE_MSMPEG4V1	19	MSMPEG4V1 format
DMX_VID_STREAM_TYPE_MSMPEG4V2	20	MSMPEG4V2 format
DMX_VID_STREAM_TYPE_MSVIDEO1	21	MSVIDEO1 format
DMX_VID_STREAM_TYPE_WMV1	22	WMV1 format
DMX_VID_STREAM_TYPE_WMV2	23	WMV2 format
DMX_VID_STREAM_TYPE_RV10	24	RV10 format
DMX_VID_STREAM_TYPE_RV20	25	RV20 format
DMX_VID_STREAM_TYPE_SVQ1	26	SVQ1 format
DMX_VID_STREAM_TYPE_SVQ3	27	SVQ3 format
DMX_VID_STREAM_TYPE_H261	28	H261 format
DMX_VID_STREAM_TYPE_VP3	29	VP3 format
DMX_VID_STREAM_TYPE_VP5	30	VP5 format
DMX_VID_STREAM_TYPE_CINEPAK	31	CINEPAK format
DMX_VID_STREAM_TYPE_INDEO2	32	INDEO2 format
DMX_VID_STREAM_TYPE_INDEO3	33	INDEO3 format
DMX_VID_STREAM_TYPE_INDEO4	34	INDEO4 format
DMX_VID_STREAM_TYPE_INDEO5	35	IINDEO5 format
DMX_VID_STREAM_TYPE_MJPEGB	36	MJPEGB format
DMX_VID_STREAM_TYPE_MVC	37	MVC format
DMX_VID_STREAM_TYPE_HEVC	38	HEVC format
DMX_VID_STREAM_TYPE_DV	39	DV format
DMX_VID_STREAM_TYPE_BUTT	40	Enumerated maximum value of video stream type

B.2.17 Enumeration definition of Audio stream type

Enumeration definition of Audio stream type is shown in Table B.18.

Table B.18 – Enumeration definition of Audio stream type (DMX_AUD_STREAM_TYPE_E)

Type name	Value	Description
DMX_AUD_STREAM_TYPE_UNKNOWN	-1	Unknown format
DMX_AUD_STREAM_TYPE_MP2	0	MP2 format
DMX_AUD_STREAM_TYPE_MP3	1	MP3 format
DMX_AUD_STREAM_TYPE_AAC	2	AAC format

Table B.18 – Enumeration definition of Audio stream type (DMX_AUD_STREAM_TYPE_E)

Type name	Value	Description
DMX_AUD_STREAM_TYPE_AC3	3	AC3 format
DMX_AUD_STREAM_TYPE_DTS	4	DTS format
DMX_AUD_STREAM_TYPE_VORBIS	5	VORBIS format
DMX_AUD_STREAM_TYPE_DVAUDIO	6	DVAUDIO format
DMX_AUD_STREAM_TYPE_WMAV1	7	WMAV1 format
DMX_AUD_STREAM_TYPE_WMAV2	8	WMAV2 format
DMX_AUD_STREAM_TYPE_MACE3	9	MACE3 format
DMX_AUD_STREAM_TYPE_MACE6	10	MACE6 format
DMX_AUD_STREAM_TYPE_VMDAUDIO	11	VMDAUDIO format
DMX_AUD_STREAM_TYPE_SONIC	12	SONIC format
DMX_AUD_STREAM_TYPE_SONIC_LS	13	SONIC LS format
DMX_AUD_STREAM_TYPE_FLAC	14	FLAC format
DMX_AUD_STREAM_TYPE_MP3ADU	15	MP3ADU format
DMX_AUD_STREAM_TYPE_MP3ON4	16	MP3ON4 format
DMX_AUD_STREAM_TYPE_SHORTEN	17	SHORTEN format
DMX_AUD_STREAM_TYPE_ALAC	18	ALAC format
DMX_AUD_STREAM_TYPE_WESTWOOD_SND1	19	WESTWOOD_SND1 format
DMX_AUD_STREAM_TYPE_GSM	20	GSM format
DMX_AUD_STREAM_TYPE_QDM2	21	QDM2 format
DMX_AUD_STREAM_TYPE_COOK	22	COOK format
DMX_AUD_STREAM_TYPE_TRUESPEECH	23	TRUESPEECH format
DMX_AUD_STREAM_TYPE_TTA	24	TTA format
DMX_AUD_STREAM_TYPE_SMACKAUDIO	25	SMACKAUDIO format
DMX_AUD_STREAM_TYPE_QCELP	26	QCELP format
DMX_AUD_STREAM_TYPE_WAVPACK	27	WAVPACK format
DMX_AUD_STREAM_TYPE_DSICINAUDIO	28	DSICINAUDIO format
DMX_AUD_STREAM_TYPE_IMC	29	IMC format
DMX_AUD_STREAM_TYPE_MUSEPACK7	30	MUSEPACK7 format
DMX_AUD_STREAM_TYPE_MLP	31	MLP format
DMX_AUD_STREAM_TYPE_GSM_MS	32	GSM_MS format
DMX_AUD_STREAM_TYPE_ATRAC3	33	ATRAC3 format
DMX_AUD_STREAM_TYPE_VOXWARE	34	VOXWARE format
DMX_AUD_STREAM_TYPE_APE	35	APE format
DMX_AUD_STREAM_TYPE_NELLYMOSER	36	NELLYMOSER format
DMX_AUD_STREAM_TYPE_MUSEPACK8	37	MUSEPACK8 format
DMX_AUD_STREAM_TYPE_SPEEX	38	SPEEX format
DMX_AUD_STREAM_TYPE_WMAVOICE	39	WMAVOICEE format
DMX_AUD_STREAM_TYPE_WMAPRO	40	WMAPRO format
DMX_AUD_STREAM_TYPE_WMALOSSLESS	41	WMALOSSLESS format

Table B.18 – Enumeration definition of Audio stream type (DMX_AUD_STREAM_TYPE_E)

Type name	Value	Description
DMX_AUD_STREAM_TYPE_ATRAC3P	42	ATRAC3P format
DMX_AUD_STREAM_TYPE_EAC3	43	EAC3 format
DMX_AUD_STREAM_TYPE_SIPR	44	SIPR format
DMX_AUD_STREAM_TYPE_MP1	45	MP11 format
DMX_AUD_STREAM_TYPE_TWINVQ	46	TWINVQ format
DMX_AUD_STREAM_TYPE_TRUEHD	47	TRUEHDHD format
DMX_AUD_STREAM_TYPE_MP4ALS	48	MP4ALSS format
DMX_AUD_STREAM_TYPE_ATRAC1	49	ATRAC1 format
DMX_AUD_STREAM_TYPE_BINKAUDIO_RDFT	50	BINKAUDIO_RDFT format
DMX_AUD_STREAM_TYPE_BINKAUDIO_DCT	51	BINKAUDIO_DCT format
DMX_AUD_STREAM_TYPE_DRA	52	DRA format
DMX_AUD_STREAM_TYPE_PCM	53	PCM format
DMX_AUD_STREAM_TYPE_PCM_BLURAY	54	PCM_BLURAY format
DMX_AUD_STREAM_TYPE_ADPCM	55	ADPCM format
DMX_AUD_STREAM_TYPE_AMR_NB	56	AMR_NBB format
DMX_AUD_STREAM_TYPE_AMR_WB	57	AMR_WB format
DMX_AUD_STREAM_TYPE_AMR_AWB	58	AMR_AWB format
DMX_AUD_STREAM_TYPE_RA_144	59	RA_144 format
DMX_AUD_STREAM_TYPE_RA_288	60	RA_288 format
DMX_AUD_STREAM_TYPE_DPCM	61	DPCM format
DMX_AUD_STREAM_TYPE_G711	62	G711 format
DMX_AUD_STREAM_TYPE_G722	63	G722 format
DMX_AUD_STREAM_TYPE_G7231	64	G7231 format
DMX_AUD_STREAM_TYPE_G726	65	G726 format
DMX_AUD_STREAM_TYPE_G728	66	G728 format
DMX_AUD_STREAM_TYPE_G729AB	67	G729AB format
DMX_AUD_STREAM_TYPE_MULTI	68	MULTI format
DMX_AUD_STREAM_TYPE_BUTT	69	Enumerated maximum value of audio stream type

B.3 Definition of the data structure

B.3.1 Filter data structure

Filter data structure is shown in Table B.19.

Table B.19 – Filter data structure (DMX_FILTER_DATA_S)

Attribute name	Type	Description
enFilterRepeatMode	Enumerated value	Data repetition mode, definition of enumeration type, check definition of DMX_FILTER_REPEAT_MODE_E in Table B.9

Table B.19 – Filter data structure (DMX_FILTER_DATA_S)

Attribute name	Type	Description
u32FilterSize	U32	Effective length of filtered data
pu8Match	pointer	Filter matching byte, bitwise comparison, pointer type U8
pu8Mask	pointer	Pointer to the matching mask, pointer type U8
pu8Notmask	pointer	Pointer to non-matching mask, pointer type U8

B.3.2 Data source structure

Data source structure is shown in Table B.20.

Table B.20 – Data source structure (DMX_SOURCE_PARAMS_S)

Attribute name	Type	Description
enSourceType	Enumerated value	Data source type. The enumeration type definition is shown in Table B.14 DMX_SOURCE_TYPE_E

DMX_SOURCE_U	Consortium	Consortium, which is defined as follows：
		union _DMX_SOURCE_U
		{
		HANDLE hSource; //**<CNcomment: enSourceType == DMX_SOURCE_TUNER*/
		CHAR u8FileName[DMX_FILE_NAME_LENGTH + 1]; //**<CNcomment: enSourceType == DMX_SOURCE_FILE*/
		HANDLE hInjecter; /**<CNcomment: enSourceType == DMX_SOURCE_MEM*/
		U32 u32PlaychnId; //**<CNcomment: enSourceType == DMX_SOURCE_PVR*/
		U32 u32Dummy; //**<CNcomment: enSourceType == DMX_SOURCE_NONE*/
		} DMX_SOURCE_U;

B.3.3 Descrambler associated parameter structure

Descrambler associated parameter structure is shown in Table B.21.

Table B.21 – Descrambler associated parameter structure (DMX_DESC_ASSOCIATE_PARAMS_S)

Attribute name	Type	Description
enMode	Enumerated value	Descrambler associated type， the definition of the enumeration type is shown in Table B.10 DMX_DESC_ASSOCIATE_MODE_E
DMX_ASSOCIATE_U	Consortium	Consortium, which is defined as follows： union _DMX_ASSOCIATE_U { U32 u32ChannelId; /**< enMode == DMX_DESCRAMBLER_ASSOCIATE_WITH_CHANNEL */

Table B.21 – Descrambler associated parameter structure (DMX_DESC_ASSOCIATE_PARAMS_S)

Attribute name	Type	Description
		U16 u16Pid; /**< enMode == DMX_DESCRAMBLER_ASSOCIATE_WITH_PIDS */ } DMX_ASSOCIATE_U;

B.3.4 Descrambler attribute structure

Descrambler attribute structure is shown in Table B.22.

Table B.22 – Descrambler attribute structure (DMX_DESCRAMBLER_ATTR_S)

Attribute name	Type	Description
enCaType	Enumerated value	Whether the descrambler uses high-security CA. The definition of the enumeration type is shown in Table B.3 DMX_DESCRAMBLER_TYPE_E
enDescramblerType	Enumerated value	Descrambler descrambling protocol type. The definition of the enumeration type is shown in Table B.11 DMX_DESC_TYPE_E
enEntropyReduction	Enumerated value	Entropy reduction mode, CAS2.0 is effective

B.3.5 DCAS keyladder setting structure

DCAS keyladder setting structure is shown in Table B.23.

Table B.23 – DCAS keyladder setting structure (DMX_DCAS_KEYLADDER_SETTING_S)

Attribute name	Type	Description
enKLAlg	Enumerated value	Key ladder algorithm. The definition of the enumeration is shown in Table B.15 DMX_KL_ALG_E
u32CAVid	U32	Verdor_SysID
pu8EK2	pointer	EK3(K2) key, 16 bytes, pointer type U8
pu8EK1	pointer	EK2(K1) key, 16 bytes, pointer type U8
pu8EvenKey	pointer	EK1(CW) Even key, 16 bytes, pointer type U8
pu8OddKey	pointer	EK1(CW) odd key, 16 bytes, pointer type U8

B.3.6 DCAS Nonce setting structure

DCAS Nonce setting structure is shown in Table B.24.

Table B.24 – DCAS Nonce setting structure (DMX_DCAS_NONCE_SETTING_S)

Attribute name	Type	Description
enKLAlg	Enumerated value	Key ladder algorithm. The definition of the enumeration is shown in Table B.15 DMX_KL_ALG_E
u32CAVid	U32	Verdor_SysID
pu8EK2	pointer	EK3(K2) key, 16 bytes, pointer type U8
pu8Nonce	pointer	Nonce value, 16 bytes, pointer type U8

B.3.7 Set the channel configuration parameter structure

Set the channel configuration parameter structure is shown in Table B.25.

Table B.25 – Set the channel configuration parameter structure (DMX_CHANNEL_SETTING_S)

Attribute name	Type	Description
enNotifyType	Enumerated value	The type of notification function call. The definition of the enumeration is shown in Table B.6 DMX_NOTIFY_TYPE_E
u32IsCRC	U32	Whether to open CRC check when receiving data, default: 1,0: close CRC check. 1: open CRC check, if CRC check fails or there is no CRC, the corresponding data is discarded
u32AddData	U32	Additional data. Will be used in some CAs
u32Tag	U32	Flag bit value, this parameter is optional

B.3.8 Channel information structure

Channel information structure is shown in Table B.26.

Table B.26 – Channel information structure (DMX_CHANNEL_INFO_S)

Attribute name	Type	Description
enDemuxId	Enumerated value	Identify which Demux belongs to
enType	Enumerated value	The type of data received, PES or SECTION. The definition of the enumeration is shown in Table B.2 DMX_CHANNEL_TYPE_E
enStatus	Enumerated value	The current working status of the channel， the definition of the enumeration is shown in Table B.7 DMX_CHANNEL_STATUS_E
u32CallbackNum	U32	Number of callback functions
u32FilterNum	U32	Number of filters
u32PacketCount	U32	The number of data packets, no precise number is required, it is mainly configured to query whether there is data coming up, for error checking
u16Pid	U16	PID to receive data
u32Len	U32	The maximum received data length, and how much memory Demux allocates to receive data

B.3.9 Section data structure

Section data structure is shown in Table B.27.

Table B.27 – Section data structure (DMX_SECTION_DATA_S)

Attribute name	Type	Description
u32Length	U32	Data length
pData	pointer	Data pointer, pointer type U8

B.3.10 Callback function data structure

Callback function data structure is shown in Table B.28.

Table B.28 – Callback function data structure (DMX_CALLBACK_DATA_S)

Attribute name	Type	Description
pstSectionData	pointer	ection data, if it is zero, it means the callback function does not carry data, pointer type DMX_SECTION_DATA_S
u32ChannelId	U32	Current channel number
enEvt	Enumerated value	Current channel events. The definition of the enumeration is shown in Table B.5, DMX_EVT_E
enDemuxId	Enumerated value	Demux ID of the current channel. The definition of the enumeration is shown in Table B.4 DMX_ID_E
u32FilterId	U32	The filter ID of the current channel, if not supported, return 0xFFFFFFF

B.3.11 Demux registration callback function structure

Demux registration callback function structure is shown in Table B.29.

Table B.29 – Demux registration callback function structure (DMX_REG_CALLBACK_PARAMS_S)

Attribute name	Type	Description
pfnCallback	Callback function pointer	Callback function that needs to be registered
u32IsDisable	U32	Whether to enable the callback: 0 enable; 1 disable
enEvt	Enumerated value	emux event, which identifies which events this callback is valid for

B.3.12 Demux initialization parameter structure

Demux initialization parameter structure is shown in Table B.30.

Table B.30 – Demux initialization parameter structure (DMX_INIT_PARAMS_S)

Attribute name	Type	Description
u32TaskPriority	U32	Task priority, unified management by the platform integrator

B.3.13 Demux channel termination parameter structure

Demux channel termination parameter structure is shown in Table B.31.

Table B.31 – Demux channel termination parameter structure (DMX_TERM_PARAM_S)

Attribute name	Type	Description
u8Dummy	U8	Reserved parameters

B.3.14 Demux channel open parameter structure

Demux channel open parameter structure is shown in Table B.32.

Table B.32 – Demux channel open parameter structure (DMX_CHANNEL_OPEN_PARAM_S)

Attribute name	Type	Description
u32Pid	U32	PID to receive data
u32Len	U32	The maximum received data length is configured to tell Demux how much memory should be allocated to receive data. If the cache length passed in by the application is too small, it must be guaranteed not to crash
enType	Enumerated value	Type of data received. The definition of the enumeration is shown in Table B.2 DMX_CHANNEL_TYPE_E
u32FirstIndex	U32	Determine which actual physical channel on the specified Demux starts to find the channels that can be opened, and then you can reserve the first few channels
stChannelSettings	Structure	For the current Channel settings, the definition of the structure in Table B.25 DMX_CHANNEL_SETTING_

B.3.15 Demux channel close parameter structure

Demux channel close parameter structure is shown in Table B.33.

Table B.33 – Demux channel close parameter structure (DMX_CHANNEL_CLOSE_PARAMS_S)

Attribute name	Type	Description
u8Dummy	U8	Reserved parameters

B.3.16 Demux capability structure

Demux capability structure is shown in Table B.34.

Table B.34 – Demux capability structure (DMX_CAPABILITY_S)

Attribute name	Type	Description
u32DMXNum	U32	Number of Demux
u32ChannelNumArr [DMX_ID_BUTT]	U32	The number of channels supported by each Demux
u32FilterNumArr [DMX_ID_BUTT]	U32	The number of filters supported by each Demux
u32DescramblerNumArr [DMX_ID_BUTT]	U32	The number of descramblers supported by each Demux
enDescStyle	Enumerated value	Descrambler style， the definition of the enumeration is shown in Table B.13 DMX_DESC_STYLE_E
enAdvDescStyle	Enumerated value	High security descrambler style， the definition of the enumeration is shown in Table B.13 DMX_DESC_STYLE_E

B.3.17 Demux status structure

Demux status structure is shown in Table B.35.

Table B.35 – Demux status structure (DMX_STATUS_S)

Attribute name	Type	Description
u32FreeChannelNum	U32	Number of free channels
u32FreeFilterNum	U32	Number of free filters
u32FreeDescramblerNum	U32	Number of free descramblers
u32TsPacketCount	U32	Number of TS packages
u32IsConnect	U32	Whether to connect

B.3.18 Get Demux channel data packet structure

Get Demux channel data packet structure is shown in Table B.36.

Table B.36 – Get Demux channel data packet structure (DMX_CHANNEL_DATA_S)

Attribute name	Type	Description
pu8Data	pointer	U8 Data pointer, pointer type U8
u32Size	U32	Data length
enDataType	Enumerated value	Type of data packet， the definition of the enumeration is shown in Table B.16 DMX_CHANNEL_DATA_TYPE_E

B.3.19 Demux module filter data open parameter structure

Demux module filter data open parameter structure is shown in Table B.37.

Table B.37 – Demux module filter data open parameter structure (DMX_PARSER_FILTER_OPEN_PARAM_S)

Attribute name	Type	Description
u32Pid	U32	PID of the channel receiving data
enType	Enumerated value	Type of data received. The definition of the enumeration is shown in Table B.2 DMX_CHANNEL_TYPE_E

DMX_AV_STREAM_TYPE_U	Consortium	union _DMX_AV_STREAM_TYPE_U
		{
		DMX_VID_STREAM_TYPE_E enVIDEsTpye;
		DMX_AUD_STREAM_TYPE_E enAUDEsType;
		} DMX_AV_STREAM_TYPE_U;

B.3.20 Demux ES frame information structure

Demux ES frame information structure is shown in Table B.38.

Table B.38 – Demux ES frame information structure (DMX_ESFRAME_INFO_S)

Attribute name	Type	Description
u64BufferAddr	U64	Data address
u32Lenght	U32	Frame length
u64Timestamp	U64	Timestamp

B.3.21 Stream data structure

Stream data structure is shown in Table B.39.

Table B.39 – Stream data structure (DMX_STREAM_DATA_S)

Attribute name	Type	Description
u32Length	U32	Data length
pu8Data	pointer	Data pointer

B.3.22 Demux module structure

Demux module structure is shown in Table B.40.

Table B.40 – Demux module structure (DMX_MODULE_S)

Attribute name	Type	Description
stCommon	Structure	The definition of structure is shown in Table B.41, hw_module_t

B.3.23 Hardware module structure

Hardware module structure is shown in Table B.41

Table B.41 – Hardware module structure (hw_module_t)

Attribute name	Type	Description
tag	U32	Data length
module_api_version	U16	The version number of the module interface
hal_api_version	U16	The version number of the HAL interface
id	pointer	Module ID
name	pointer	Module name
author	pointer	Module author
methods	Structure pointer	Hardware module method structure

B.4 Definition of Callback function

B.4.1 Demux channel callback function

The prototype : typedef S32 (*DMX_CALLBACK_PFN) (const DMX_CALLBACK_DATA_S* const pstData)

Function: channel callback function.

Input parameter: pstData Callback function parameter.

Output parameters ： None.

Return: 0: correct; non-zero: error.

B.5 Calling method

The hardware abstract interface calling method of the Demux module is shown in Figure B.1.

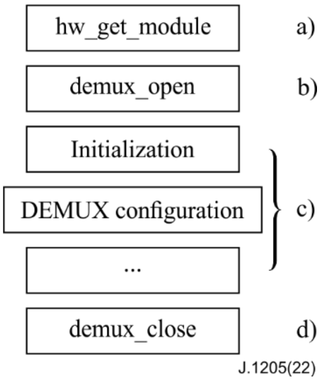

Figure B.1 – Demux module hardware abstract interface calling method

Figure B.1

a) Call the hw_get_module() interface to get the HAL Stub of the Demux module: hw_get_module(DEMUX_HARDWARE_MODULE_ID, &g_dmx_module)

b) Call Demux_open(g_dmx_module,&pstDevice) to get the device handle of the Demux module: demux_open(g_dmx_module,&pstDevice).

c) Control Demux hardware through a series of interface functions provided by the device handle.

d) After finishing the hardware manipulation, call the demux_close() interface to close the Demux device to avoid resource leakage

B.6 Definition of the interface

B.6.1 "Open the Demux module device" interface

The prototype: static inline int demux_open (const struct hw_module_t* pstModule, DEMUX_DEVICE_S** pstDevice);

Function: Open a Demux module device

Input parameter: pstModule Demux module handle.

Output parameter: pstDevice Demux device handle.

Return: 0: correct; non-zero: error

B.6.2 "Close the Demux module device" interface

The prototype: static inline int demux_close (DEMUX_DEVICE_S* pstDevice);

Function: Close a Demux module device.

Input parameter: pstDevice Demux device handle.

Output parameters: None.

Return: 0: correct; non-zero: error

B.6.3 "Demux module initialization" interface

The prototype: S32 (*dmx_init) (struct _DEMUX_DEVICE_S *pstDev, const DMX_INIT_PARAMS_S * const pstInitParams);

Function: Demux module initialization.

Input parameter: pstDev Demux device handle

pstInitParams initialization parameters

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.4 "Demux module termination" interface

The prototype: S32 (*dmx_term) (struct _DEMUX_DEVICE_S *pstDev, const DMX_TERM_PARAM_S * const pstTermParams);

Function: Demux module is terminated.

Input parameter: pstDev Demux device handle

pstTermParams module termination parameters

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.5 "Set the Demux module data source" interface

The prototype: S32 (*dmx_set_source_params) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId, const DMX_SOURCE_PARAMS_S * pstSourceParams);

Function: Set the data source of the Demux module

Input parameter: pstDev Demux device handle

enDemuxId Demux ID

pstSourceParams source parameters.

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.6 "Get Demux data source parameter" interface

The prototype: S32 (*dmx_get_source_params) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId, DMX_SOURCE_PARAMS_S * const pstSourceParams);

Function: Obtain Demux data source parameters.

Input parameter: pstDev Demux device handle

enDemuxId Demux ID

Output parameters：pstSourceParams Data source parameters.

Return: 0: correct; non-zero: error

B.6.7 "Disconnect the association between Demux and the data source" interface

The prototype: S32 (*dmx_disconnect) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId);

Function: Disconnect the association between dmx and the data source (such as demod), and it is associated by default at startup.

Input parameter: pstDev Demux device handle

enDemuxId DemuxID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.8 "Restore the association between Demux and the data source" interface

The prototype: S32 (*dmx_reconnect) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId)；

Function: Restore the association between dmx and the currently set data source (such as demod), which is associated by default at startup

Input parameter: pstDev Demux device handle

enDemuxId DemuxID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.9 "Get capability of Demux" interface

The prototype: S32 (*dmx_get_capability) (struct _DEMUX_DEVICE_S *pstDev, DMX_CAPABILITY_S * const pstCapability);

Function: Get the ability of Demux

Input parameter: pstDev Demux device handle

Output parameter: pstCapability Demux capability parameter

Return: 0: correct; non-zero: error

B.6.10 "Get status of Demux" interface

The prototype: S32 (*dmx_get_status) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId, DMX_STATUS_S * const pstStatus);

Function: Get the status of Demux.

Input parameter: pstDev Demux device handle

enDemuxId DemuxID

Output parameter: pstStatus Demux status.

Return: 0: correct; non-zero: error

B.6.11 "Open data channel" interface

The prototype: S32 (*dmx_channel_open) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId, U32 * const pu32ChannelId, const DMX_CHANNEL_OPEN_PARAM_S * const pstOpenParams);

Function: Open a data channel. After open, the Channel is in the disabled state. Illegal PID can be passed in when it open, and it can be set later through dmx_channel_set_pid().

Input parameter: pstDev Demux device handle

enDemuxId DemuxID;

pstOpenParams open parameters

Output parameter: pu32ChannelId Channel ID

Return: 0: correct; non-zero: error

B.6.12 "Close data channel" interface

The prototype: S32 (*dmx_channel_close) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const DMX_CHANNEL_CLOSE_PARAMS_S * pstCloseParams);

Function: Close a data channel and release related resources.

Input parameter: pstDev Demux device handle

u32ChannelId Channel ID;

pstCloseParams Close parameter

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.13 "set data channel PID" interface

The prototype: S32 (*dmx_channel_set_pid) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const U16 u16Pid);

Function: Set the PID of the data channel.

Input parameter: pstDev Demux device handle

u32ChannelId Channeled

u16Pid Data channel PID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.14 "Query channel number through PID" interface

The prototype: S32 (*dmx_channel_query) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId, U32 * const pu32ChannelId, const U16 u16Pid);

Function: Query the channel corresponding to the PID through PID

Input parameter: pstDev Demux device handle

enDemuxId DemuxID

u16Pid Channel PID

Output parameter: pu32ChannelId The channel corresponding to the PID.

Return: 0: correct; non-zero: error

B.6.15 "Open data channel to receive data" interface

Function: Open the data channel to receive data.

Input parameter: pstDev Demux device handle

u32ChannelId Channel ID。

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.16 "Stop receiving data via data channel" interface

The prototype: S32 (*dmx_channel_disable) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId);

Function: stop receiving data via the data channel

Input parameter: pstDev Demux device handle

u32ChannelId Channel ID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.17 "Reset data channel" interface

The prototype: S32 (*dmx_channel_reset) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId);

Function: Reset data channel

Input parameter: pstDev   Demux device handle

u32ChannelId   Channel ID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.18 "Get data channel information" interface

The prototype: S32 (*dmx_channel_get_info) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, DMX_CHANNEL_INFO_S * const pstInfo);

Function: Get data channel information

Input parameter: pstDev   Demux device handle

u32ChannelId   Data channel ID

Output parameter: pstInfo   Data channel information

Return: 0: correct; non-zero: error

B.6.19 "Set channel parameter" interface

The prototype: S32 (*dmx_channel_set) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const DMX_CHANNEL_SETTING_S * const pstSettings);

Function: When setting channel parameters and modifying channel parameters, it is recommended to get it out first, then modify the members that need to be modified and then set back. The set function does not modify the members that have not changed.

Input parameter: pstDev   Demux device handle

u32ChannelId   Data channel ID.

Output parameters: pstSettings   configuration parameters.

Return: 0: correct; non-zero: error

B.6.20 "Get the configuration of the channel" interface

The prototype: S32 (*dmx_channel_get) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, DMX_CHANNEL_SETTING_S * const pstSettings);

Function: Get the configuration of the channel.

Input parameter: pstDev   Demux device handle

u32ChannelId   Data channel ID.

Output parameter pstSettings   Channel configuration information.

Return: 0: correct; non-zero: error

B.6.21 "Get data of a specified data channel" interface

The prototype: S32 (*dmx_channel_get_buf) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, U32 u32AcquirePackageNum, U32 * pu32AcquiredNum, DMX_CHANNEL_DATA_S * pstChannelData, const U32 u32TimeoutMs);

Function: Get data of a specified data channel,this function is a synchronous function,it waits until the data is obtained or the timeout returns.For section channels and ECM/EMM channels, each data packet contains a complete section; for PES channels, each data packet contains a complete PES as much as possible, however, if the PES is too large, it may be output in multiple PES packets.Whether the output data is complete is specified by the enDataType field of the data packet structure;For data injection channels, each data packet contains one or more complete TS packets, and the TS packet is 188 bytes long;For audio and video channels, data cannot be obtained through this interface. The audio and video data will be directly sent to the decoder through the internal interface for decoding. Repeated calls of this interface are not allowed. You can request multiple releases at one time, but the releases must be in order, and addresses and lengths of the releases must be consistent with the request. In addition, only after all the data packets are released can the request be made again, otherwise the repeated request error code will be returned.

Input parameter: pstDev   Demux device handle

u32ChannelId   data channel ID;

u32AcquirePackageNum   The total number of packages expected to acquire data

u32TimeoutMs   timeout duration, in milliseconds

Output parameter: pu32AcquiredNum The total number of data packets actually obtained

pstChannelData   data pointer

Return: 0: correct; non-zero: error

B.6.22 "Release the buffer space occupied by the data packet" interface

The prototype: S32 (*dmx_channel_release_buf) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, U32 u32ReleaseNum, DMX_CHANNEL_DATA_S * pstChannelData);

Function: Release the buffer space occupied by data packets.

Input parameter: pstDev   Demux device handle

u32ChannelId   Data channel ID

u32ReleaseNum   release the number of data packets;

pstChannelData   pointer to the packet to be released

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.23 "Register the callback function for the channel" interface

The prototype: S32 (*dmx_channel_register_callback) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const DMX_REG_CALLBACK_PARAMS_S * const pstReg);

Function: Register a callback function for the channel. A channel can register up to DMX_CHANNEL_CALLBACK_MAX callback functions.

Input parameter: pstDev   Demux device handle

u32ChannelId   Data channel ID

pstReg   callback function pointer

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.24 "Configure callback function of the corresponding data channel" interface

The prototype: S32 (*dmx_channel_config_callback) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const DMX_CALLBACK_PFN pfnCallback, const DMX_CFG_CALLBACK_E enCfg);

Function: Configure the callback function of the corresponding data channel.

Input parameter: pstDev   Demux device handle

u32ChannelId   Data channel ID

pfnCallback   callback function pointer

enCfg   configuration parameters

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.25 "Assign filter to data channel" interface

The prototype: S32 (*dmx_channel_add_filter) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, U32 * const pu32FilterId, const DMX_FILTER_DATA_S * const pstFilterData);

Function: Assign filters to data channels and set filter data

Input parameter: pstDev   Demux device handle

u32ChannelId   Data channel ID

Output parameter: pu32FilterId   filter ID;

pstFilterData   filter data pointer

Return: 0: correct; non-zero: error

B.6.26 "Set the filter condition of the filter" interface

The prototype: S32 (*dmx_channel_set_filter) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const U32 u32FilterId, const DMX_FILTER_DATA_S * const pstFilterData);

Function: Set the filter condition of the filter

Input parameter: pstDev   Demux device handle

u32ChannelId   Channel ID

u32FilterId   filter ID

pstFilterData   Filter data.

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.27 "Get the filter condition of the filter" interface

The prototype: S32 (*dmx_channel_get_filter) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const U32 u32FilterId, DMX_FILTER_DATA_S * const pstFilterData);

Function: Get the filter conditions of the filter

Input parameter: pstDev   Demux device handle

u32ChannelId   Data channel ID

u32FilterId   Filter ID.

Output parameter: pstFilterData   Filter data.

Return: 0: correct; non-zero: error

B.6.28 "Delete filter" interface

The prototype: S32 (*dmx_channel_destroy_filter) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const U32 u32FilterId);

Function: Delete filter.

Input parameter: pstDev Demux device handle

u32ChannelId Data channel ID

u32FilterId Filter ID.

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.29 "Delete all filters of the specified data channel" interface

The prototype: S32 (*dmx_channel_destroy_all_filter) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId);

Function: Delete all filters of the specified data channel.

Input parameter: pstDev Demux device handle

u32ChannelId Data channel ID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.30 "Open the designated filter of the data channel to start receiving data" interface

The prototype: S32 (*dmx_channel_enable_filter) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const U32 u32FilterId);

Function: Open the specified filter of the data channel to start receiving data.

Input parameter: pstDev Demux device handle

u32ChannelId Data channel ID

u32FilterId filter ID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.31 "Prohibit the specified filter in the data channel" interface

The prototype: S32 (*dmx_channel_disable_filter) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, const U32 u32FilterId);

Function: Disable the specified filter in the data channel.

Input parameter: pstDev Demux device handle

u32ChannelId Data channel ID

u32FilterId filter ID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.32 "Query filter based on table ID or extended table ID" interface

The prototype: S32 (*dmx_channel_query_filter_by_table_id) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, U32 * const pu32FilterId, const U8 u8TableId, const U16 u16ExtId);

Function: Query whether there is a filter for a table ID and an extended table ID

Input parameter: pstDev Demux device handle

u32ChannelId Data channel ID

u8TableId table ID

u16ExtId extension table ID

Output parameter: pu32FilterId filter ID

Return: 0: correct; non-zero: error

B.6.33 "Query filter condition by filter data" interface

The prototype: S32 (*dmx_channel_query_filter_by_filter_data) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32ChannelId, U32 * const pu32FilterId, const DMX_FILTER_DATA_S * const pstFilterData);

Function: Whether the channel filters have the same filter condition.

Input parameter: pstDev Demux device handle

u32ChannelId Data channel ID

pstFilterData filter data

Output parameter: pu32FilterId filter ID

Return: 0: correct; non-zero: error

B.6.34 "Assign descrambling channel" interface

The prototype: S32 (*dmx_descrambler_open) (struct _DEMUX_DEVICE_S *pstDev, const U32 enDemuxId, U32 * const pu32DescId, const DMX_DESC_ASSOCIATE_MODE_E enMode);

Function: Assign a descrambling channel.

Input parameter: pstDev Demux device handle

enDemuxId DemuxID

enMode Descrambler association type.

Output parameter: pu32DescId descrambler ID

Return: 0: correct; non-zero: error

B.6.35 "Assign descrambling channel extension interface" interface

The prototype: S32 (*dmx_descrambler_open_ex) (struct _DEMUX_DEVICE_S *pstDev, const U32 enDemuxId, U32 * const pu32DescId, const DMX_DESCRAMBLER_ATTR_S *pstDesramblerAttr);

Function: Assign a descrambling channel extension interface

Input parameter: pstDev Demux device handle

enDemuxId DemuxID

pstDesramblerAttr Descrambler parameters

Output parameter: pu32DescId   descrambler ID

Return: 0: correct; non-zero: error

B.6.36 "Enable descrambler" interface

The prototype: S32 (*dmx_descrambler_enable) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId);

Function: Enable a descrambler

Input parameter: pstDev   Demux device handle

u32DescId   Descrambler ID.

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.37 "Disable descrambler" interface

The prototype: S32 (*dmx_descrambler_disable) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId);

Function: Disable a descrambler

Input parameter: pstDev   Demux device handle

u32DescId   Descrambler ID.

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.38 "Close the descrambler" interface

The prototype: S32 (*dmx_descrambler_close) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId);

Function: Close a descrambler.

Input parameter: pstDev   Demux device handle

u32DescId   Descrambler ID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.39 "Associate a descrambler" interface

The prototype: S32 (*dmx_descrambler_associate) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, const DMX_DESC_ASSOCIATE_PARAMS_S * const pstParams);

Function: Associate the descrambler and the PID or channel to be descrambled

Input parameter: pstDev   Demux device handle

u32DescId   Descrambler ID

pstParams   associated parameters

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.40 "Get associated information of the descrambler" interface

The prototype: S32 (*dmx_descrambler_get_associate_info) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, DMX_DESC_ASSOCIATE_PARAMS_S * const pstParams);

Function: Get the descrambling PID or channel associated with the descrambler

Input parameter: pstDev Demux device handle

u32DescId Descrambler ID

Output parameters: pstParams related parameters

Return: 0: correct; non-zero: error

B.6.41 "Set even key of the descrambling channel" interface

The prototype: S32 (*dmx_descrambler_set_even_key) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, const U8 * const pu8Key, const U32 u32Len, const U32 u32Option);

Function: set even key of the descrambling channel

Input parameter: pstDev Demux device handle

u32DescId descrambler ID;

pu8Key even key

u32Len key length

u32Option reserved parameters.

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.42 "Set initial vector corresponding to the descrambler even key" interface

The prototype: S32 (*dmx_descrambler_set_even_iv) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, const U8 * const pu8IV, const U32 u32Len);

Function: Set the initial vector of the data corresponding to the even key of the descrambling channel.

Input parameter: pstDev Demux device handle

u32DescId Descrambler ID

pu8IV IV vector

u32Len IV vector length.

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.43 "Set the odd key of the descrambler" interface

The prototype: S32 (*dmx_descrambler_set_odd_key) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, const U8 * const pu8Key, const U32 u32Len, const U32 u32Option);

Function: Set the odd key of the descrambling channel

Input parameter: pstDev Demux device handle

u32DescId Descrambler ID

pu8Key odd key

u32Len key length

u32Option reserved parameters

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.44 "Set the initial vector corresponding to the odd key of the descrambler" interface

The prototype: S32 (*dmx_descrambler_set_odd_iv) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, const U8 * const pu8IV, const U32 u32Len);

Function: Set the initial vector of the data corresponding to the odd key of the descrambling channel.

Input parameter: pstDev Demux device handle

u32DescId Descrambler ID

pu8IV IV vector

u32Len IV vector length

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.45 "Set descrambler attribute" interface

The prototype: S32 (*dmx_set_descrambler_attribute) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, DMX_DESCRAMBLER_ATTR_S *pstAttr);

Function: Set the attribute of the descrambling channel

Input parameter: pstDev Demux device handle

u32DescId Descrambler ID

pstAttr descrambling channel attributes.

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.46 "Get descrambler attribute" interface

The prototype: S32 (*dmx_get_descrambler_attribute) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, DMX_DESCRAMBLER_ATTR_S *pstAttr);

Function: Get the attribute of the descrambling channel

Input parameter: pstDev Demux device handle

u32DescId Descrambler ID

Output parameter: pstAttr Descramble channel attributes

Return: 0: correct; non-zero: error

B.6.47 "Set DCAS keyladder TS descrambler parameter" interface

The prototype: S32 (*dmx_dcas_keyladder_config) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, const DMX_DCAS_KEYLADDER_SETTING_S *pstDcasKLConfig);

Function: Set DCAS keyladder TS descrambler parameters

Input parameter: pstDev Demux device handle

u32DescId Descrambler ID

pstDcasKLConfig DCAS keyladder configuration structure pointer.

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.48 "Get DA (nonce)" interface

The prototype: S32 (*dmx_dcas_get_nonce) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32DescId, const DMX_DCAS_NONCE_SETTING_S *pstDcasNonceConfig, U8 *pu8DANonce);

Function: Get DA (nonce)

Input parameter: pstDev Demux device handle

u32DescId Descrambler ID

pstDcasNonceConfig Nonce configuration parameter.

Output parameter: pu8DANonce DA value

Return: 0: correct; non-zero: error

B.6.49 "Open the filter" interface

The prototype: S32 (*dmx_avfilter_open) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId, U32 * const pu32AVFilterId, const DMX_PARSER_FILTER_OPEN_PARAM_S * const pstFilterOpenPara);

Function: Open a filter to filter AV data

Input parameter: pstDev Demux device handle

enDemuxId Demux ID;

pstFilterOpenPara filter configuration parameters

Output parameter: pu32AVFilterId filter ID

Return: 0: correct; non-zero: error

B.6.50 "Enable filter" interface

The prototype: S32 (*dmx_avfilter_enable) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32AVFilterId);

Function: Enable filter

Input parameter: pstDev Demux device handle

u32AVFilterId filter ID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.51 "Get frame data" interface

The prototype: S32 (*dmx_avfilter_get_esframe) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32AVFilterId, DMX_ESFRAME_INFO_S *pstFrameInfo);

Function: Get frame data

Input parameter: pstDev Demux device handle

u32AVFilterId filter ID

Output parameter: pstFrameInfo obtained frame data.

Return: 0: correct; non-zero: error

B.6.52 "Release frame data" interface

The prototype: S32 (*dmx_avfilter_release_esframe) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32AVFilterId, DMX_ESFRAME_INFO_S *pstFrameInfo);

Function: Release frame data.

Input parameter: pstDev Demux device handle

u32AVFilterId filter ID

pstFrameInfo frame data to be released

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.53 "Disable the filter configured to filter audio and video" interface

The prototype: S32 (*dmx_avfilter_disable) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32AVFilterId);

Function: Disable the filter configured to filter audio and video

Input parameter: pstDev Demux device handle

u32AVFilterId filter ID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.54 "Close filter" interface

The prototype: S32 (*dmx_avfilter_close) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32AVFilterId);

Function: Close filter.

Input parameter: pstDev Demux device handle

u32AVFilterId filter ID

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.55 "Open PCR channel" interface

The prototype: S32 (*dmx_pcr_open) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId, U32 * const pu32PcrId, const U32 u32Pid);

Function: Open a PCR filter channel.

Input parameter: pstDev Demux device handle

enDemuxId DemuxID

u32Pid PCR channel Pid

Output parameter: pu32PcrId PCRID

Return: 0: correct; non-zero: error

B.6.56 "Close PCR channel" interface

The prototype: S32 (*dmx_pcr_close) (struct _DEMUX_DEVICE_S *pstDev, const U32 u32PcrId);

Function: Close a PCR channel.

Input parameter: pstDev Demux device handle

u32PcrId PCRID。

Output parameters: None.

Return: 0: correct; non-zero: error

B.6.57 "Get PCR" interface

The prototype: S32 (*dmx_pcr_get) (struct _DEMUX_DEVICE_S* pstDev, const U32 u32PcrId, U64* pu64StcTime);

Function: Get PCR

Input parameter: pstDev Demux device handle

u32PcrId PCRID。

Output parameter: pu64StcTime Stc time

Return: 0: correct; non-zero: error

B.6.58 "Create TS buffer" interface

The prototype: S32 (*dmx_tsbuffer_create) (struct _DEMUX_DEVICE_S *pstDev, const DMX_ID_E enDemuxId, U32 * const pu32TsBufferId);

Function: Create TS buffer to receive TS data input from the network or locally

Input parameter: pstDev Demux device handle

enDemuxId DemuxID

Output parameter: pu32TsBufferId TS buffer handle

Return: 0: correct; non-zero: error

B.6.59 "Get TS buffer" interface

The prototype: S32 (*dmx_tsbuffer_get) (struct _DEMUX_DEVICE_S *pstDev, U32 u32TsBufferId, U32 u32Size, U32 u32TimeoutMs, DMX_STREAM_DATA_S *pstStreamData);

Function: Create a TS buffer space

Input parameter: pstDev Demux device handle

u32TsBufferId TS buffer handle

u32Size the length of the data to be input

u32TimeoutMs wait timeout time

Output parameter: pstStreamData data buffer structure

Return: 0: correct; non-zero: error

B.6.60 "Write data to TS buffer" interface

The prototype: S32 (*dmx_tsbuffer_put) (struct _DEMUX_DEVICE_S *pstDev, U32 u32TsBufferId, U32 u32ValidDataLen);

Function: TS data input is completed, configured to update TS data write pointer

Input parameter: pstDev Demux device handle

u32TsBufferId TS buffer handle

u32ValidDataLen valid data length

Output parameters：None.

Return: 0: correct; non-zero: error

B.6.61 "Delete TS buffer" interface

The prototype: S32 (*dmx_tsbuffer_destroy) (struct _DEMUX_DEVICE_S *pstDev, U32 u32TsBufferId);

Function: destroy the TS buffer space created

Input parameter: pstDev Demux device handle

u32TsBufferId TS buffer ID

Output parameters: None.

Return: 0: correct; non-zero: error

B.6.62 "Get streampath data" interface

The prototype: S32 (*dmx_get_streampath_param) (struct _DEMUX_DEVICE_S *pstDev, DMX_ID_E enDemuxId, TEE_KLAD_BYTE *streamPath, int *streamPathLength);

Function: Get the streampath parameters needed to set the keyladder on the TEE side

Input parameter: pstDev Demux device handle

enDemuxId Demux ID

Output parameter: streamPath streamPath parameter pointer, depends on the realization of each manufacturer, can output demuxID;

streamPathLength The length of streamPath.

Return: 0: correct; non-zero: error

Annex C

Frontend module

(This annex forms an integral part of this Recommendation.)

This annex defines the hardware abstraction layer interface of the Frontend module. The definitions of the basic data types and operators are given in clause 6.

C.1 Constant definition

The definition of constants is shown in Table C.1.

Table C.1 – Constant definition

Constant	Description
Type of business	
const FRONTEND_HARDWARE_MODULE_ID = " frontend"	Frontend module ID
const FRONTEND_HARDWARE_FRONTEND0 = " frontend0"	Frontend equipment ID
const FRONTEND_DEVICENAME_LENGTH = ＂32＂	Tuner ID maximum length
const FRONTEND_CALLBACK_MAX = ＂8＂	The maximum number of callback functions registered by each Frontend
const FRONTEND_FE_NUM_MAX = ＂8＂	The maximum number of Tuner

C.2 Enumeration definition

C.2.1 Enumeration definition of Frontend state

Enumeration definition of Frontend state is shown in Table C.2.

Table C.2 – Enumeration definition of Frontend state (FRONTEND_FE_STATUS_E)

Type name	Value	Description
FRONTEND_STATUS_UNKNOW	0	Unknown state. After the device is initialized but there is no instance, it is in an unknown state
FRONTEND_STATUS_UNLOCKED	1	Unlocked, it will return to this state after many attempts to lock it. The bottom layer is still trying
FRONTEND_STATUS_SCANNING	2	Scanning, try to change to TUNER_STATUS_UNLOCKED if it cannot be locked (determine whether it is locked after scanning)
FRONTEND_STATUS_LOCKED	3	Locked
FRONTEND_STATUS_NOSIGNAL	4	No signal, can be the same with TUNER_STATUS_UNLOCKED
FRONTEND_STATUS_DISCONNECTED	5	TS stream data has been disconnected

Table C.2 – Enumeration definition of Frontend state (FRONTEND_FE_STATUS_E)

Type name	Value	Description
FRONTEND_STATUS_IDLE	6	The initial state. In this state, the frequency will not be actively locked (similar to the idle state)
FRONTEND_STATUS_BLINDSCANING	7	The demodulator is in the blind scan state, the application needs to wait for the blind scan to complete, or send a frequency lock message to force the blind scan to exit
FRONTEND_STATUS_BLINDSCAN_COMPLETE	8	Demodulator blind scan completed
FRONTEND_STATUS_BLINDSCAN_QUIT	9	User quit
FRONTEND_STATUS_BLINDSCAN_FAIL	10	Scan failed
FRONTEND_STATUS_MOTOR_MOVING	11	Motor moving
FRONTEND_STATUS_MOTOR_STOP	12	Motor stop
FRONTEND_STATUS_BUTT	13	Enumerated maximum value of Frontend state

C.2.2 Enumeration definition of Front-end type

Enumeration definition of Front-end type is shown in Table C.3.

Table C.3 – Enumeration definition of Front-end type (FRONTEND_FE_TYPE_E)

Type name	Value	Description
FRONTEND_FE_SATELLITE1	1	Satellite 1
FRONTEND_FE_SATELLITE2	2	Satellite 2
FRONTEND_FE_CABLE1	4	Cable1 Cable1
FRONTEND_FE_CABLE2	8	Cable2 Cable2
FRONTEND_FE_TERRESTRIAL1	16	TERRESTRIAL1 TERRESTRIAL1
FRONTEND_FE_TERRESTRIAL2	32	TERRESTRIAL2 TERRESTRIAL2
FRONTEND_FE_ATV1	64	ATV1 ATV1
FRONTEND_FE_DTMB1	128	DTMB1 DTMB1
FRONTEND_FE_ISDBT1	256	ISDB-T ISDB-T
FRONTEND_FE_ATSCT	512	ATSC-T ATSC-T
FRONTEND_FE_J83B	1024	J83B J83B
FRONTEND_FE_BUTT	1025	Enumerated maximum value of Front-end type

C.2.3 Enumeration definition of Satellite frequency search method

Enumeration definition of Satellite frequency search method is shown in Table C.4.

Table C.4 – Enumeration definition of Satellite frequency search method （FRONTEND_SEARCH_MODE_E）

Type name	Value	Description
FRONTEND_SEARCH_MOD_NORMAL	0	Default
FRONTEND_SEARCH_MOD_BLIND	1	Frequency step and search method are unknown
FRONTEND_SEARCH_MOD_COLD_START	2	Known search method
FRONTEND_SEARCH_MOD_WARM_START	3	Known frequency step and search method
FRONTEND_SEARCH_MOD_SAT_BLIND_MANUAL	4	Set the starting frequency, polarization and high and low local oscillators. See the definition of Table C.71 FRONTEND_SAT_BLINDSCAN_PARA_S

C.2.4 Enumeration definition of FRONTEND_IQ_IVT_E

Enumeration definition of FRONTEND_IQ_IVT_E is shown in Table C.5.

Table C.5 – Enumeration definition of FRONTEND_IQ_IVT_E (FRONTEND_IQ_IVT_E)

Type name	Value	Description
FRONTEND_IQ_AUTO	0	Automatic mode
FRONTEND_IQ_AUTO_NORMAL_FIRST	1	Normal first
FRONTEND_IQ_FORCE_NORMAL	2	Force Normal
FRONTEND_IQ_FORCE_SWAPPED	3	Forced swapped
FRONTEND_IQ_BUTT	4	The maximum value of enumeration definition

C.2.5 Enumeration definition of LNB type

Enumeration definition of LNB Type is shown in Table C.6.

Table C.6 – Enumeration definition of LNB type (FRONTEND_LNB_TYPE_E)

Type name	Value	Description
FRONTEND_LNB_SINGLE_FREQUENCY	0	Single frequency
FRONTEND_LNB_DUAL_FREQUENCY	1	Dual frequency
FRONTEND_LNB_UNICABLE	2	Single Cable Tuner
FRONTEND_LNB_TYPE_BUTT	3	Enumerated maximum value of LNB type

C.2.6 Enumeration definition of Satellite signal frequency band

Enumeration definition of Satellite signal frequency band is shown in Table C.7.

Table C.7 – Enumeration definition of Satellite signal frequency band (FRONTEND_LNB_BAND_E)

Type name	Value	Description
FRONTEND_LNB_BAND_C	0	C band
FRONTEND_LNB_BAND_KU	1	Ku band
FRONTEND_LNB_BAND_BUTT	2	Enumerated maximum value of Satellite signal frequency band

C.2.7 Enumeration definition of LNB power switch (satellite, ground)

Enumeration definition of LNB power switch (satellite, ground) is shown in Table C.8.

Table C.8 – Enumeration definition of LNB power switch (satellite, ground) (FRONTEND_LNB_POWER_E)

Type name	Value	Description
FRONTEND_LNB_POWER_OFF	0	Power off
FRONTEND_LNB_POWER_ON	1	Power on the default 13V/18V power supply
FRONTEND_LNB_POWER_ENHANCED	2	Enhanced power supply
FRONTEND_LNB_POWER_AUTO	3	Automatic mode
FRONTEND_LNB_POWER_BUTT	4	Enumerated maximum value of LNB power switch (satellite, ground)

C.2.8 Enumeration definition of Rolloff factor(satellite)

Enumeration definition of Rolloff factor(satellite) is shown in Table C.9.

Table C.9 – Enumeration definition of Rolloff factor(satellite) (FRONTEND_ROLLOFF_E)

Type name	Value	Description
FRONTEND_ROLLOFF_35	0	ROLLOFF_35 ROLLOFF_35
FRONTEND_ROLLOFF_25	1	ROLLOFF_25 ROLLOFF_25
FRONTEND_ROLLOFF_20	2	ROLLOFF_20 ROLLOFF_2
FRONTEND_ROLLOFF_BUTT	3	Enumerated maximum value of Roll-off coefficient (satellite)

C.2.9 Enumeration definition of Polarization method(satellite)

Enumeration definition of Polarization method(satellite)is shown in Table C.10.

Table C.10 – Enumeration definition of Polarization method(satellite) (FRONTEND_POLARIZATION_E)

Type name	Value	Description
FRONTEND_PLR_HORIZONTAL	0	Horizontal polarization
FRONTEND_PLR_VERTICAL	1	Vertical polarization
FRONTEND_PLR_LEFT	2	Left polarization
FRONTEND_PLR_RIGHT	3	Right polarization
FRONTEND_PLR_AUTO	4	Automatic polarization
FRONTEND_PLR_BUTT	5	Enumerated maximum value of Polarization method (satellite)

C.2.10 Enumeration definition of 12 V control(satellite)

Enumeration definition of 12 V control(satellite) is shown in Table C.11.

Table C.11 – Enumeration definition of 12 V control(satellite) (FRONTEND_LNB_12V_E)

Type name	Value	Description
FRONTEND_LNB_12V_OFF	0	Turn off
FRONTEND_LNB_12V_ON	1	Turn on
FRONTEND_LNB_12V_AUTO	2	Automatic mode
FRONTEND_LNB_12V_BUTT	3	Enumerated maximum value of 12 V control(satellite)

C.2.11 Enumeration definition of 0/12 V switch

Enumeration definition of 0/12 V switch is shown in Table C.12.

Table C.12 – Enumeration definition of 0/12V switch (FRONTEND_SWITCH_0_12V_E)

Type name	Value	Description
FRONTEND_SWITCH_0_12V_NONE	0	No switch state
FRONTEND_SWITCH_0_12V_0	1	O V state
FRONTEND_SWITCH_0_12V_12	2	12 V state
FRONTEND_SWITCH_0_12V_BUTT	3	Enumerated maximum value of 0/12 V switch

C.2.12 Enumeration definition of 22 kHz output control(satellite)

Enumeration definition of 22 kHz output control(satellite) is shown in Table C.13.

Table C.13 – Enumeration definition of 22 kHz output control(satellite) (FRONTEND_LNB_22K_E)

Type name	Value	Description
FRONTEND_LNB_22K_OFF	0	22 kHz Off
FRONTEND_LNB_22K_ON	1	22 kHz On
FRONTEND_LNB_22K_AUTO	2	22 kHz Automatic
FRONTEND_LNB_22K_BUTT	3	Enumerated maximum value of 22 kHz output control(satellite)

C.2.13 Enumeration definition of 22K switch control

Enumeration definition of 22K switch control is shown in Table C.14.

Table C.14 – Enumeration definition of 22K switch control (FRONTEND_SWITCH_22K_E)

Type name	Value	Description
FRONTEND_SWITCH_22K_NONE	0	No switch state
FRONTEND_SWITCH_22K_OFF	1	0 kHz port
FRONTEND_SWITCH_22K_ON	2	22 kHz port
FRONTEND_SWITCH_22K_BUTT	3	Enumerated maximum value of 22K switch control

C.2.14 Enumeration definition of Tone burst switch control

Enumeration definition of Tone burst switch control is shown in Table C.15.

Table C.15 – Enumeration definition of Tone burst switch control (FRONTEND_SWITCH_TONEBURST_E)

Type name	Value	Description
FRONTEND_SWITCH_TONEBURST_NONE	0	No switch state
FRONTEND_SWITCH_TONEBURST_0	1	0 port
FRONTEND_SWITCH_TONEBURST_1	2	1 port
FRONTEND_SWITCH_TONEBURST_BUTT	3	Enumerated maximum value of Tone burst switch control

C.2.15 Enumeration definition of DiSEqC switch port

Enumeration definition of DiSEqC switch port is shown in Table C.16.

Table C.16 – Enumeration definition of DiSEqC switch port (FRONTEND_DISEQC_SWITCH_PORT_E)

Type name	Value	Description
FRONTEND_DISEQC_SWITCH_NONE	0	No switch
FRONTEND_DISEQC_SWITCH_PORT_1	1	Port 1
FRONTEND_DISEQC_SWITCH_PORT_2	2	Port 2
FRONTEND_DISEQC_SWITCH_PORT_3	3	Port 3
FRONTEND_DISEQC_SWITCH_PORT_4	4	Port 4
FRONTEND_DISEQC_SWITCH_PORT_5	5	Port 5
FRONTEND_DISEQC_SWITCH_PORT_6	6	Port 6
FRONTEND_DISEQC_SWITCH_PORT_7	7	Port 7
FRONTEND_DISEQC_SWITCH_PORT_8	8	Port 8
FRONTEND_DISEQC_SWITCH_PORT_9	9	Port 9
FRONTEND_DISEQC_SWITCH_PORT_10	10	Port 10
FRONTEND_DISEQC_SWITCH_PORT_11	11	Port 11
FRONTEND_DISEQC_SWITCH_PORT_12	12	Port 12
FRONTEND_DISEQC_SWITCH_PORT_13	13	Port 13
FRONTEND_DISEQC_SWITCH_PORT_14	14	Port 14
FRONTEND_DISEQC_SWITCH_PORT_15	15	Port 15
FRONTEND_DISEQC_SWITCH_PORT_16	16	Port 16
FRONTEND_DISEQC_SWITCH_PORT_BUTT	17	Enumerated maximum value of DiSEqC switch port

C.2.16 Enumeration definition of Diseqc protocol(satellite)

Enumeration definition of Diseqc protocol(satellite)is shown in Table C.17.

Table C.17 – Enumeration definition of Diseqc protocol(satellite) (FRONTEND_LNB_DISEQC_CMD_E)

Type name	Value	Description
FRONTEND_DISEQC_COMMAND	0	DiSEqC (1.2/2) command
FRONTEND_DISEQC_TONE_BURST_UNMODULATED	5	TONE_BURST_UNMODULATED TONE_BURST_UNMODULATED
FRONTEND_DISEQC_TONE_BURST_MODULATED	6	TONE_BURST_MODULATED TONE_BURST_UNMODULATED
FRONTEND_DISEQC_TONE_BURST_SEND_0_UNMODULATED	5	TONE_BURST_UNMODULATED TONE_BURST_UNMODULATED
FRONTEND_DISEQC_TONE_BURST_SEND_0_MODULATED	6	TONE_BURST_MODULATED TONE_BURST_UNMODULATED
FRONTEND_DISEQC_BUTT	7	Enumerated maximum value of Diseqc protocol(satellite)

C.2.17 Enumeration definition of Diseqc version(satellite)

Enumeration definition of Diseqc version(satellite)is shown in Table C.18.

Table C.18 – Enumeration definition of Diseqc version(satellite) (FRONTEND_DISEQC_VER_E)

Type name	Value	Description
FRONTEND_DISEQC_VER_NONE	0	Not support
FRONTEND_DISEQC_VER_1_0	1	DiSEqC V1.0
FRONTEND_DISEQC_VER_1_1	2	DiSEqC V1.1
FRONTEND_DISEQC_VER_1_2	3	DiSEqC V1.2
FRONTEND_DISEQC_VER_1_X	4	DiSEqC V1.x all V1
FRONTEND_DISEQC_VER_2_0	5	DiSEqC V2.0
FRONTEND_DISEQC_VER_2_1	6	DiSEqC V2.1
FRONTEND_DISEQC_VER_2_2	7	DiSEqC V2.2
FRONTEND_DISEQC_VER_2_X	8	DiSEqC V2.x all V2
FRONTEND_DISEQC_VER_BUTT	9	Enumerated maximum value of Diseqc version(satellite)

C.2.18 Enumeration definition of LNB command control type(satellite)

Enumeration definition of LNB command control type(satellite)is shown in Table C.19.

Table C.19 – Enumeration definition of LNB command control type (satellite) (FRONTEND_LNB_CMD_TYPE_E)

Type name	Value	Description
FRONTEND_LNB_CMD_SET_PWR	1	Set LNB power switch
FRONTEND_LNB_CMD_SET_POL	2	Set polarization mode
FRONTEND_LNB_CMD_SET_22K	4	Set 22 kHz switch
FRONTEND_LNB_CMD_SET_12V	8	Set the 12 V switch
FRONTEND_LNB_CMD_SET_DISQ	16	Set up digital satellite device control
FRONTEND_LNB_CMD_BUTT	17	Enumerated maximum value of LNB command control type(satellite)

C.2.19 Enumeration definition of Antenna individual setting type

Enumeration definition of Antenna individual setting type is shown in Table C.20.

Table C.20 – Enumeration definition of Antenna individual setting type (FRONTEND_ANTENNA_CMD_TYPE_E)

Type name	Value	Description
FRONTEND_EXTRA_ANTENNA_CONFIG_ALL	0	The definition is shown in Table C.63 FRONTEND_SAT_EXTRA_ANTENNA_CONFIG_S
FRONTEND_SET_LNB_POWER	1	Set LNB power
FRONTEND_SET_SWITCH_22K	2	22 kHz
FRONTEND_DISEQC_SWITCH_4PORT	3	The definition is shown in Table C.54 FRONTEND_DISEQC_SWITCH4PORT_S

Table C.20 – Enumeration definition of Antenna individual setting type (FRONTEND_ANTENNA_CMD_TYPE_E)

Type name	Value	Description
FRONTEND_DISEQC_SWITCH_16PORT	4	The definition is shown in Table C.55 FRONTEND_DISEQC_SWITCH16PORT_S
FRONTEND_MOTOR_SET_COORDINATE	5	The definition is shown in Table C.56 FRONTEND_COORDINATE_S
FRONTEND_MOTOR_STORE_POSITION	6	The definition is shown in Table C.57 FRONTEND_MOTOR_POSITION_S
FRONTEND_MOTOR_GOTO_POSITION	7	The definition is shown in Table C.57 FRONTEND_MOTOR_POSITION_S
FRONTEND_MOTOR_LIMIT	8	The definition is shown in Table C.58 FRONTEND_MOTOR_LIMIT_S
FRONTEND_MOTOR_MOVE	9	The definition is shown in Table C.59 FRONTEND_MOTOR_MOVE_S
FRONTEND_MOTOR_STOP	10	The definition is shown in Table C.18 FRONTEND_DISEQC_VER_E
FRONTEND_MOTOR_CALC_ANGULAR	11	The definition is shown in Table C.60 FRONTEND_MOTOR_CALC_ANGULAR_S
FRONTEND_MOTOR_GOTO_ANGULAR	12	The definition is shown in Table C.61 FRONTEND_MOTOR_USALS_ANGULAR_S
FRONTEND_MOTOR_SET_MANUAL	13	manual setting
FRONTEND_UNICABLE_SCAN_USERBANDS	14	Scan 950-2150 to find the user frequency band
FRONTEND_UNICABLE_EXIT_SCANUSERBANDS	15	Stop scanning user bands
FRONTEND_UNICABLE_GET_USERBANDSINFO	16	Get scan results, get user frequency band information
FRONTEND_ANTENNA_CMD_BUTT	17	Enumerated maximum value of Antenna individual setting type

C.2.20 Enumeration definition of Motor Agreement

Enumeration definition of Motor Agreement is shown in Table C.21.

Table C.21 – Enumeration definition of Motor Agreement (FRONTEND_MOTORTYPE_E)

Type name	Value	Description
FRONTEND_MOTOR_NONE	0	Not use motor
FRONTEND_MOTOR_DISEQC12	1	DiSEqC1.2 agreement
FRONTEND_MOTOR_USLAS	2	DiSEqC1.3 or USLAS agreement
FRONTEND_MOTOR_BUTT	3	Enumerated maximum value of Motor Agreement

C.2.21 Enumeration definition of DiSEqC Motor limit setting

Enumeration definition of DiSEqC Motor limit setting is shown in Table C.22.

Table C.22 – Enumeration definition of DiSEqC Motor limit setting (FRONTEND_MOTOR_LIMIT_E)

Type name	Value	Description
FRONTEND_MOTOR_LIMIT_OFF	0	Limit off
FRONTEND_MOTOR_LIMIT_EAST	1	Limit east
FRONTEND_MOTOR_LIMIT_WEST	2	Limit west
FRONTEND_MOTOR_LIMIT_BUTT	3	Enumerated maximum value of DiSEqC Motor limit setting

C.2.22 Enumeration definition of DiSEqC Motor Moving direction

Enumeration definition of DiSEqC Motor Moving direction is shown in Table C.23.

Table C.23 – Enumeration definition of DiSEqC Motor Moving direction (FRONTEND_MOTOR_MOVE_DIR_E)

Type name	Value	Description
FRONTEND_MOTOR_MOVE_DIR_EAST	0	Move east
FRONTEND_MOTOR_MOVE_DIR_WEST	1	Move west
FRONTEND_MOTOR_MOVE_DIR_BUTT	2	Enumerated maximum value of DiSEqC Motor Moving direction

C.2.23 Enumeration definition of Unicable switch port

Enumeration definition of Unicable switch port is shown in Table C.24.

Table C.24 – Enumeration definition of Unicable switch port (FRONTEND_UNICABLE_SATPOSITION_E)

Type name	Value	Description
FRONTEND_UNICABLE_SATPOSN_A	0	Port A
FRONTEND_UNICABLE_SATPOSN_B	1	Port B
FRONTEND_UNICABLE_SATPOSN_BUT	2	Enumerated maximum value of switch port

C.2.24 Enumeration definition of Forward error correction rate

Enumeration definition of Forward error correction rate is shown in Table C.25.

Table C.25 – Enumeration definition of Forward error correction rate (FRONTEND_FEC_RATE_E)

Type name	Value	Description
FRONTEND_FEC_AUTO	0	Automatic error correction
FRONTEND_FEC_1_2	1	1/2 error correction
FRONTEND_FEC_2_3	2	2/3 error correction

Table C.25 – Enumeration definition of Forward error correction rate (FRONTEND_FEC_RATE_E)

Type name	Value	Description
FRONTEND_FEC_3_4	3	3/4 error correction
FRONTEND_FEC_3_5	4	3/5 error correction
FRONTEND_FEC_4_5	5	4/5 error correction
FRONTEND_FEC_5_6	6	5/6 error correction
FRONTEND_FEC_6_7	7	6/7 error correction
FRONTEND_FEC_7_8	8	7/8 error correction
FRONTEND_FEC_8_9	9	8/9 error correction
FRONTEND_FEC_9_10	10	9/10 error correction
FRONTEND_FEC_ANNEX_B	11	ANNEX_B error correction
FRONTEND_FEC_1_3	12	1/3 error correction
FRONTEND_FEC_1_4	13	1/4 error correction
FRONTEND_FEC_2_5	14	2/5 error correction
FRONTEND_FEC_BUTT	15	Enumerated maximum value of Forward error correction rate

C.2.25 Enumeration definition of Modulation

Enumeration definition of Modulation is shown in Table C.26.

Table C.26 – Enumeration definition of Modulation (FRONTEND_MODULATION_E)

Type name	Value	Description
FRONTEND_MOD_AUTO	0	Automatic modulation
FRONTEND_MOD_QAM16	1	QAM16 modulation
FRONTEND_MOD_QAM32	2	QAM32 modulation
FRONTEND_MOD_QAM64	3	QAM64 modulation
FRONTEND_MOD_QAM128	4	QAM128 modulation
FRONTEND_MOD_QAM256	5	QAM256 modulation
FRONTEND_MOD_QPSK	6	QPSK modulation
FRONTEND_MOD_8PSK	7	8PSK modulation
FRONTEND_MOD_BPSK	8	BPSK modulation
FRONTEND_MOD_DVBT	9	DVBT modulation
FRONTEND_MOD_DVBT2	10	DVBT2 modulation
FRONTEND_MOD_ISDBT	11	ISDBT modulation
FRONTEND_MOD_QAM1024	12	QAM4096 modulation
FRONTEND_MOD_QAM4096	13	QAM4096 modulation
FRONTEND_MOD_16APSK	14	16APSK modulation
FRONTEND_MOD_32APSK	15	32APSK modulation
FRONTEND_MOD_8VSB	16	8VSB modulation
FRONTEND_MOD_16VSB	17	16VSB modulation

Table C.26 – Enumeration definition of Modulation (FRONTEND_MODULATION_E)

Type name	Value	Description
FRONTEND_MOD_BUTT	18	Enumerated maximum value of modulation

C.2.26 Enumeration definition of Spectrum inversion control (wired, ground)

Enumeration definition of Spectrum inversion control (wired, ground) is shown in Table C.27.

Table C.27 – Enumeration definition of Spectrum inversion control (wired, ground) (FRONTEND_SPECTRUM_E)

Type name	Value	Description
FRONTEND_SPECTRUM_INVERSION_OFF	0	Turn off spectrum inversion control
FRONTEND_SPECTRUM_INVERSION	1	Turn on spectrum inversion control
FRONTEND_SPECTRUM_INVERSION_AUTO	2	Automatic
FRONTEND_SPECTRUM_INVERSION_UNK	3	UN K method
FRONTEND_SPECTRUM_INVERSION_BUTT	4	Enumerated maximum value of spectrum inversion control (wired, ground)

C.2.27 Enumeration definition of Transmission bandwidth (ground, wired)

Enumeration definition of Transmission bandwidth (ground, wired) is shown in Table C.28.

Table C.28 – Enumeration definition of Transmission bandwidth (terrestrial, wired) (FRONTEND_BAND_WIDTH_E)

Type name	Value	Description
FRONTEND_BANDWIDTH_8_MHZ	0	8MHZ
FRONTEND_BANDWIDTH_7_MHZ	1	7MHZ
FRONTEND_BANDWIDTH_6_MHZ	2	6MHZ
FRONTEND_BANDWIDTH_BUTT	3	Transmission bandwidth (ground, wired)

C.2.28 Enumeration definition of Transmission mode(ground)

Enumeration definition of Transmission mode(ground) is shown in Table C.29.

Table C.29 – Enumeration definition of Transmission mode(ground) (FRONTEND_TRANSMIT_MOD_E)

Type name	Value	Description
FRONTEND_TRANS_MOD_1K	0	1k mode
FRONTEND_TRANS_MOD_2K	1	2k mode
FRONTEND_TRANS_MOD_4K	2	4k mode
FRONTEND_TRANS_MOD_8K	3	8k mode
FRONTEND_TRANS_MOD_16K	4	16k mode
FRONTEND_TRANS_MOD_32K	5	32k mode

Table C.29 – Enumeration definition of Transmission mode(ground) (FRONTEND_TRANSMIT_MOD_E)

Type name	Value	Description
FRONTEND_TRANS_MOD_AUTO	6	automatic mode
FRONTEND_TRANS_MOD_BUTT	7	Enumerated maximum value of Transmission mode (ground)

C.2.29 Enumeration definition of Guard interval (ground, C2)

Enumeration definition of Guard interval (ground, C2) is shown in Table C.30.

Table C.30 – Enumeration definition of Guard interval (ground, C2) (FRONTEND_GUARD_INTERVAL_E)

Type name	Value	Description
FRONTEND_GUARDINTERVAL_1_128	0	The guard interval is 1/128 of the character length
FRONTEND_GUARDINTERVAL_1_64	1	The guard interval is 1/64 of the character length
FRONTEND_GUARDINTERVAL_1_32	2	The guard interval is 1/32 of the character length
FRONTEND_GUARDINTERVAL_1_16	3	The guard interval is 1/16 of the character length
FRONTEND_GUARDINTERVAL_19_256	4	The guard interval is 19/256 of the character length
FRONTEND_GUARDINTERVAL_1_8	5	The guard interval is 1/8 of the character length
FRONTEND_GUARDINTERVAL_19_128	6	The guard interval is 19/128 of the character length
FRONTEND_GUARDINTERVAL_1_4	7	The guard interval is 1/4 of the character length
FRONTEND_GUARDINTERVAL_AUTO	8	Adaptive mode
FRONTEND_GUARDINTERVAL_BUTT	9	Enumerated maximum value of Guard interval (ground, C2)

C.2.30 Enumeration definition of TS priority

Enumeration definition of TS priority is shown in Table C.31.

Table C.31 – Enumeration definition of TS priority (FRONTEND_TER_TS_PRIORITY_E)

Type name	Value	Description
FRONTEND_TER_TS_PRIORITY_NONE	0	No priority mode
FRONTEND_TER_TS_PRIORITY_HP	1	High priority mode
FRONTEND_TER_TS_PRIORITY_LP	2	Low priority mode
FRONTEND_TER_TS_PRIORITY_BUTT	3	Enumerated maximum value of TS priority

C.2.31 Enumeration definition of Channel mode

Enumeration definition of TS Channel mode is shown in Table C.32.

Table C.32 – Enumeration definition of TS Channel mode (FRONTEND_TER2_MODE_E)

Type name	Value	Description
FRONTEND_TER2_MODE_BASE	0	Only base signal is supported in the channel
FRONTEND_TER2_MODE_LITE	1	Need to support lite signal in the channel
FRONTEND_TER2_MODE_BUTT	2	Enumerated maximum value of TS Channel mode

C.2.32 Enumeration definition of Physical layer pipe type under T2

Enumeration definition of Physical layer pipe type under T2 is shown in Table C.33.

Table C.33 – Enumeration definition of Physical layer pipe type under T2 (FRONTEND_TER2_PLP_TYPE_E)

Type name	Value	Description
FRONTEND_TER2_PLP_TYPE_COM	0	Common type
FRONTEND_TER2_PLP_TYPE_DAT1	1	Data 1 type
FRONTEND_TER2_PLP_TYPE_DAT2	2	Data 2 type
FRONTEND_TER2_PLP_TYPE_BUTT	3	Enumerated maximum value of Physical layer pipe type under T2

C.2.33 Enumeration definition of Configure callback function status

Enumeration definition of Configure callback function status is shown in Table C.34.

Table C.34 – Enumeration definition of Configure callback function status (FRONTEND_CFG_CALLBACK_E)

Type name	Value	Description
FRONTEND_CALLBACK_ENABLE	0	Enable
FRONTEND_CALLBACK_DISABLE	1	Disable
FRONTEND_CALLBACK_REMOVE	2	Remove

C.2.34 Enumeration definition of LNB power status

Enumeration definition of LNB power status is shown in Table C.35.

Table C.35 – Enumeration definition of LNB power status (FRONTEND_LNB_PWR_STATUS_E)

Type name	Value	Description
FRONTEND_LNB_PWR_STATUS_ON	0	Normal state of turning on
FRONTEND_LNB_PWR_STATUS_OFF	1	Normal state of turning off
FRONTEND_LNB_PWR_STATUS_SHORT_CIRCUIT	2	LNB output circuit is shorted
FRONTEND_LNB_PWR_STATUS_OVER_TEMPERATURE	3	LNB output module temperature is too high

Table C.35 – Enumeration definition of LNB power status (FRONTEND_LNB_PWR_STATUS_E)

Type name	Value	Description
FRONTEND_LNB_PWR_STATUS_LOW_VOLTAGE	4	LNB output voltage is too low
FRONTEND_LNB_PWR_STATUS_OVER_VOLTAGE	5	LNB output voltage is too high
FRONTEND_LNB_PWR_STATUS_BUTT	6	Enumerated maximum value of LNB power status

C.2.35 Enumeration definition of Command to control output stream

Enumeration definition of Command to control output stream is shown in Table C.36.

Table C.36 – Enumeration definition of Command to control output stream (FRONTEND_TSOUT_CMD_TYPE_E)

Type name	Value	Description
FRONTEND_TSOUT_TER2_GET_PLPNUM	0	Get the number of PLP
FRONTEND_TSOUT_TER2_GET_PLP_TYPE	1	Get the type of PLP
FRONTEND_TSOUT_TER2_GET_PLP_GROUPID	2	Get the PLP group ID, type U8
FRONTEND_TSOUT_TER2_SET_PLP_MODE	3	Set the physical layer pipeline read and write mode
FRONTEND_TSOUT_TER2_SET_PLPID	4	Incoming PLPID, type U8
FRONTEND_TSOUT_TER2_SET_COMMON_PLPID	5	Set the shared physical layer pipe ID, type U8
FRONTEND_TSOUT_TER2_SET_COMPLP_COMB	6	Set whether the shared physical layer pipeline and the data physical layer pipeline need to be combined flags
FRONTEND_TSOUT_SAT2_GET_ISINUM	7	Get the number of PLP, type U8
FRONTEND_TSOUT_SAT2_GET_ISIID	8	Get ISI ID
FRONTEND_TSOUT_SAT2_SET_ISIID	9	Set ISI ID
FRONTEND_TSOUT_ISDBT_GET_TMCC_INFO	10	Get TMCC information of ISDB-T signal
FRONTEND_TSOUT_CMD_END	11	Enumerated maximum value of Command to control output stream

C.2.36 Enumeration definition of Tuner blind scan event

Enumeration definition of Tuner blind scan event is shown in Table C.37.

Table C.37 – Enumeration definition of Tuner blind scan event (FRONTEND_SAT_BLINDSCAN_EVT_E)

Type name	Value	Description
FRONTEND_SAT_BLINDSCAN_EVT_STATUS	0	State change
FRONTEND_SAT_BLINDSCAN_EVT_PROGRESS	1	Progress change
FRONTEND_SAT_BLINDSCAN_EVT_NEWRESULT	2	New frequency
FRONTEND_SAT_BLINDSCAN_EVT_BUTT	3	Enumerated maximum value of Tuner blind scan event

C.2.37 Enumeration definition of ATV system information

Enumeration definition of ATV system information is shown in Table C.38.

Table C.38 – Enumeration definition of ATV system information (FRONTEND_ATV_SYSTEM_E)

Type name	Value	Description
FRONTEND_ATV_SYSTEM_PAL_BG	0	PAL BG TV system
FRONTEND_ATV_SYSTEM_PAL_DK	1	TV system TV system
FRONTEND_ATV_SYSTEM_PAL_I	2	PAL I TV system
FRONTEND_ATV_SYSTEM_PAL_M	3	PAL M TV system
FRONTEND_ATV_SYSTEM_PAL_N	4	PAL N TV system
FRONTEND_ATV_SYSTEM_SECAM_BG	5	SECAM BG TV system
FRONTEND_ATV_SYSTEM_SECAM_DK	6	SECAM DK TV system
FRONTEND_ATV_SYSTEM_SECAM_L_PRIME	7	SECAM L PRIME TV system
FRONTEND_ATV_SYSTEM_SECAM_LL	8	SECAM LL TV system
FRONTEND_ATV_SYSTEM_NTSC_M	9	NTSC M TV system
FRONTEND_ATV_SYSTEM_BUTT	10	Enumerated maximum value of ATV system information

C.2.38 Enumeration definition of ATV Search mode

Enumeration definition of ATV Search mode is shown in Table C.39.

Table C.39 – Enumeration definition of ATV Search mode (FRONTEND_ATV_SIF_BW_E)

Type name	Value	Description
FRONTEND_ATV_SIF_BW_WIDE	0	Auto search mode
FRONTEND_ATV_SIF_BW_NORMAL	1	Normal playback mode
FRONTEND_ATV_SIF_BW_NARROW	2	Narrow mode
FRONTEND_ATV_SIF_BW_BUTT	3	Enumerated maximum value of ATV Search mode

C.2.39 Enumeration definition of ATV working mode

Enumeration definition of ATV working mode is shown in Table C.40.

Table C.41 – Enumeration definition of ATV working mode (FRONTEND_ATV_CONNECT_WORK_MODE_E)

Type name	Value	Description
FRONTEND_CONNECT_WORK_MODE_NORMAL	0	Normal working mode
FRONTEND_CONNECT_WORK_MODE_CHAN_SCAN	1	RF searching mode
FRONTEND_CONNECT_WORK_MODE_BUTT	2	Enumerated maximum value of ATV working mode

C.2.40 Enumeration definition of ATV Frontend lock status

Enumeration definition of ATV Frontend lock status is shown in Table C.41.

Table C.41 – Enumeration definition of ATV Frontend lock status (FRONTEND_ATV_LOCK_STATUS_E)

Type name	Value	Description
FRONTEND_ATV_UNLOCK	0	Unlock status
FRONTEND_ATV_LOCK	1	Lock status
FRONTEND_ATV_BUTT	2	Enumerated maximum value of ATV Frontend lock status

C.2.41 Enumeration definition of DTMB carrier type

Enumeration definition of DTMB carrier type is shown in Table C.42.

Table C.42 – Enumeration definition of DTMB carrier type (FRONTEND_DTMB_CARRIER_MODE_E)

Type name	Value	Description
FRONTEND_DTMB_CARRIER_UNKNOWN	0	Unknown type
FRONTEND_DTMB_CARRIER_SINGLE	1	Single carrier
FRONTEND_DTMB_CARRIER_MULTI	2	Multi-carrier
FRONTEND_DTMB_CARRIER_BUTT	3	Enumerated maximum value of DTMB carrier type

C.2.42 Enumeration definition of DTMB DTMB code rate type

Enumeration definition of DTMB DTMB code rate type is shown in Table C.43.

Table C.43 – Enumeration definition of DTMB DTMB code rate (FRONTEND_DTMB_CODE_RATE_E)

Type name	Value	Description
FRONTEND_DTMB_CODE_RATE_UNKNOWN	0	Unknown type
FRONTEND_DTMB_CODE_RATE_0_DOT_4	1	0.4 code rate
FRONTEND_DTMB_CODE_RATE_0_DOT_6	2	0.6 code rate
FRONTEND_DTMB_CODE_RATE_0_DOT_8	3	0.8 code rate
FRONTEND_DTMB_CODE_RATE_BUTT	4	Enumerated maximum value of DTMB code rate

C.2.43 Enumeration definition of DTMB time domain interleaving type

Enumeration definition of DTMB time domain interleaving type is shown in Table C.44.

Table C.44 – Enumeration definition of DTMB time domain interleaving type (FRONTEND_DTMB_TIME_INTERLEAVE_E)

Type name	Value	Description
FRONTEND_DTMB_TIME_INTERLEAVER_UNKNOWN	0	Unknown type
FRONTEND_DTMB_TIME_INTERLEAVER_240	1	240 type
FRONTEND_DTMB_TIME_INTERLEAVER_720	2	720 type
FRONTEND_DTMB_TIME_INTERLEAVER_BUTT	3	Enumerated maximum value of DTMB time domain interleaving type

C.2.44 Enumeration definition of DTMB guard interval type

Enumeration definition of DTMB guard interval type is shown in Table C.45.

Table C.45 – Enumeration definition of DTMB guard interval type (FRONTEND_DTMB_GUARD_INTERVAL_E)

Type name	Value	Description
FRONTEND_DTMB_GI_UNKNOWN	0	Unknown type
FRONTEND_DTMB_GI_420	1	The guard interval inserts a PN sequence with a length of 420 as the frame header
FRONTEND_DTMB_GI_595	2	The guard interval inserts a PN sequence with a length of 595 as the frame header
FRONTEND_DTMB_GI_945	3	The guard interval inserts a PN sequence with a length of 945 as the frame header
FRONTEND_DTMB_GI_BUTT	4	Enumerated maximum value of DTMB guard interval type

C.2.45 Enumeration definition of Unicable scan event

Enumeration definition of Unicable scan event is shown in Table C.46.

Table C.46 – Enumeration definition of Unicable scan event (FRONTEND_UNICABLE_SCAN_EVT_E)

Type name	Value	Description
FRONTEND_UNICABLE_SCAN_EVT_STATUS	0	State change
FRONTEND_UNICABLE_SCAN_EVT_PROGRESS	1	Progress change
FRONTEND_UNICABLE_SCAN_EVT_BUTT	2	Enumerated maximum value of scan event

C.2.46 Enumeration definition of Unicable scan status

Enumeration definition of Unicable scan status is shown in Table C.47.

Table C.47 – Enumeration definition of Unicable scan status (FRONTEND_UNICABLE_SCAN_EVT_E)

Type name	Value	Description
FRONTEND_UNICABLE_SCAN_STATUS_IDLE	0	idle
FRONTEND_UNICABLE_SCAN_STATUS_SCANNING	1	scanning
FRONTEND_UNICABLE_SCAN_STATUS_FINISH	2	Completed successfully
FRONTEND_UNICABLE_SCAN_STATUS_QUIT	3	User quit
FRONTEND_UNICABLE_SCAN_STATUS_FAIL	4	Scan failed
FRONTEND_UNICABLE_SCAN_STATUS_BUTT	5	Enumerated maximum value of scan status

C.2.47 Enumeration definition of Callback function type

Enumeration definition of Callback function type is shown in Table C.48.

Table C.48 – Enumeration definition of Callback function type (FRONTEND_REG_CALLBACK_TYPE_E)

Type name	Value	Description
FRONTEND_REG_LOCKSTATUS_CALLBACK	0	Set LOCKSTATUS as the callback function. The definition of the Callback function is shown in C.4
FRONTEND_REG_GETSTATUS_CALLBACK	1	Set GETSTATUS as the callback function. The definition of the Callback function is shown in C.4
FRONTEND_REG_BLINDSCAN_CALLBACK	2	Set BLINDSCAN as the callback function. The definition of the Callback function is shown in C.4
FRONTEND_REG_LNB_PWR_STATUS_CALLBACK	3	Set LNB_PWR_STATUS as the callback function. The definition of the Callback function is shown in C.4
FRONTEND_REG_UNICABLE_SCAN_CALLBACK	4	Set UNICABLE_SCAN as the callback function. The definition of the Callback function is shown in C.4
FRONTEND_REG_CALLBACK_TYPE_BUTT	5	Enumerated maximum value of Callback function type

C.2.48 Enumeration definition of the Second-generation frontend

Enumeration definition of the Second-generation frontend is shown in Table C.49.

Table C.49 – Enumeration definition of the Second-generation frontend (FRONTEND_G2_STREAM_TYPE_E)

Type name	Value	Description
FRONTEND_G2_STREAM_TYPE_UNKNOWN	0	Unknown
FRONTEND_G2_STREAM_TYPE_TS	1	MPEG-TS stream
FRONTEND_G2_STREAM_TYPE_GSE	2	GSE stream, GSE refers to general stream

Table C.49 – Enumeration definition of the Second-generation frontend (FRONTEND_G2_STREAM_TYPE_E)

Type name	Value	Description
		encapsulation (Generic Stream Encapsulation）
FRONTEND_G2_STREAM_TYPE_GCS	4	GCS stream, GCE refers to Generic Continuous Stream (Generic Continuous Stream)
FRONTEND_G2_STREAM_TYPE_GFPS	8	GFPS stream, GFPS refers to Generic Fixed-length Packetized Stream
FRONTEND_G2_STREAM_TYPE_BUTT	9	Enumerated maximum value of the Second-generation frontend

C.3 Enumeration definition of data structure

C.3.1 Frontend initialization parameter structure

Frontend initialization parameter structure is shown in Table C.50.

Table C.50 – Frontend initialization parameter structure (FRONTEND_INIT_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved parameters

C.3.2 Frontend deinitialize the parameter structure

Frontend deinitialize the parameter structure is shown in Table C.51.

Table C.51 – Frontend deinitialize the parameter structure (FRONTEND_TERM_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved parameters

C.3.3 Device open parameter structure

Device open parameter structure is shown in Table C.52.

Table C.52 – Device open parameter structure (FRONTEND_OPEN_PARAMS_S)

Attribute name	Type	Description
u32FrontendIndex	U32	Frontend index
enFeType	Enumerated value	Initialize Tuner to this type. The definition of the enumeration type is shown in Table C.3 FRONTEND_FE_TYPE_E
enFecRate	Enumerated value	Error correction rate. The definition of the enumeration type is shown in Table C.25 FRONTEND_FEC_RATE_E

C.3.4 Information (ISDB-T mode) structure

Information (ISDB-T mode) structure is shown in Table C.53.

Table C.53 – Information (ISDB-T mode) structure (FRONTEND_ISDBT_TMCC_INFO_S)

Attribute name	Type	Description
u8EmergencyFlag	U8	Emergency alarm announcement start sign
u8PartialFlag	U8	Partially accepted flag
u8PhaseShiftCorr	U8	Phase offset value
u8IsdbtSystemId	U8	System identification

C.3.5 DiSEqC 1.0/2.0 switch parameter structure

DiSEqC 1.0/2.0 switch parameter structure is shown in Table C.54.

Table C.54 – DiSEqC 1.0/2.0 switch parameter structure (FRONTEND_DISEQC_SWITCH4PORT_S)

Attribute name	Type	Description
enLevel	Enumerated value	Device version. The definition of the enumeration value is shown in Table C.18 FRONTEND_DISEQC_VER_E
enPort	Enumerated value	Gated port number. The definition of the enumeration type is shown in Table C.16 FRONTEND_DISEQC_SWITCH_PORT_E
enPolar	Enumerated value	Polarization mode. The definition of the enumeration is shown in Table C.10 FRONTEND_POLARIZATION_E
enLNB22K	Enumerated value	22 kHz state. The definition of the enumeration is shown in Table C.13 FRONTEND_LNB_22K_E

C.3.6 DiSEqC 1.1/2.1 Switch parameter structure

DiSEqC 1.1/2.1 Switch parameter structure is shown in Table C.55.

Table C.55 – DiSEqC 1.1/2.1 Switch parameter structure (FRONTEND_DISEQC_SWITCH16PORT_S)

Attribute name	Type	Description
enLevel	Enumerated value	Device version. The definition of the enumeration value is shown in Table C.18 FRONTEND_DISEQC_VER_E
enPort	Enumerated value	Gated port number. The definition of the enumeration type is shown in Table C.16 FRONTEND_DISEQC_SWITCH_PORT_E

C.3.7 Local latitude and longitude parameter structure

Local latitude and longitude parameter structure is shown in Table C.56.

Table C.56 – Local latitude and longitude parameter structure (FRONTEND_COORDINATE_S)

Attribute name	Type	Description
u16MyLongitude	U16	longitude
u16MyLatitude	U16	latitude

C.3.8 Antenna storage location parameter structure

Antenna storage location parameter structure is shown in Table C.57.

Table C.57 – Antenna storage location parameter structure (FRONTEND_MOTOR_POSITION_S)

Attribute name	Type	Description
enLevel	Enumerated value	Device version. The definition of the enumeration value is shown in Table C.18 FRONTEND_DISEQC_VER_E
u32Pos	U32	Position number

C.3.9 Antenna Limit setting parameter structure

Antenna Limit setting parameter structure is shown in Table C.58.

Table C.58 – Antenna Limit setting parameter structure (FRONTEND_MOTOR_LIMIT_S)

Attribute name	Type	Description
enLevel	Enumerated value	Device version. The definition of the enumeration value is shown in Table C.18 FRONTEND_DISEQC_VER_E
enLimit	Enumerated value	Limit settings. The definition of the enumeration value is shown in Table C.22 FRONTEND_MOTOR_LIMIT_E的

C.3.10 DiSEqC motor movement parameter structure

DiSEqC motor movement parameter structure is shown in Table C.59.

Table C.59 – iSEqC motor movement parameter structure (FRONTEND_MOTOR_MOVE_S)

Attribute name	Type	Description
enLevel	Enumerated value	Device version. The definition of the enumeration value is shown in Table C.18 FRONTEND_DISEQC_VER_E
enDir	Enumerated value	Moving direction. The definition of the enumeration value is shown in Table C.23 FRONTEND_MOTOR_MOVE_DIR_E
u32RunningSteps	U32	0 means continuous rotation; 1～128 means the number of steps per rotation

C.3.11 Calculate angle structure

Calculate angle structure is shown in Table C.60.

Table C.60 – Calculate angle structure (FRONTEND_MOTOR_CALC_ANGULAR_S)

Attribute name	Type	Description
u16SatLongitude	U16	Satellite longitude
u16Angular	U16	Calculated angle

C.3.12 Angle parameter structure

USALS angle parameter structure is shown in Table C.61.

Table C.61 – USALS angle parameter structure (FRONTEND_MOTOR_USALS_ANGULAR_S)

Attribute name	Type	Description
enLevel	Enumerated value	Device version. The definition of the enumeration value is shown in Table C.18 FRONTEND_DISEQC_VER_E
u32Angular	U32	Angle value

C.3.13 LNB configuration parameter structure

LNB configuration parameter structure is shown in Table C.62.

Table C.62 – LNB configuration parameter structure (FRONTEND_SAT_LNB_CONFIG_S)

Attribute name	Type	Description
enLNBType	Enumerated value	LNB type. The definition of the enumeration value is shown in Table C.6 FRONTEND_LNB_TYPE_E
u32LowLO	U32	LNB low local oscillator frequency, in megahertz (MHz)
u32HighLO	U32	LNB high local oscillator frequency, in megahertz (MHz)
enLNBBand	Enumerated value	LNB band: C or Ku. The definition of the enumeration value is shown in Table C.7 FRONTEND_LNB_BAND_E
u32UNIC_SCRNO	U32	SCR serial number, ranging from 0 to 7
u32UNICIFFreqMHz	U32	SCR intermediate frequency, in megahertz (MHz)
enSatPosn	Enumerated value	Switch port number. The definition of the enumeration value is shown in Table C.24 FRONTEND_UNICABLE_SATPOSITION_E

C.3.14 Satellite antenna parameter structure

Satellite antenna parameter structure is shown in Table C.63.

Table C.63 – Satellite antenna parameter structure (FRONTEND_SAT_EXTRA_ANTENNA_CONFIG_S)

Attribute name	Type	Description
enSwitch22K	Enumerated value	22 kHz switch. The definition of the enumeration value is shown in Table C.14 FRONTEND_SWITCH_22K_E
enToneburst	Enumerated value	Tone burst switch. The definition of the enumeration value is shown in Table C.15 FRONTEND_SWITCH_TONEBURST_E
enSwitch12V	Enumerated value	12 V switch, the definition of the enumeration value is shown in Table C.12 FRONTEND_SWITCH_0_12V_E

Table C.63 – Satellite antenna parameter structure (FRONTEND_SAT_EXTRA_ANTENNA_CONFIG_S)

Attribute name	Type	Description
enDiSEqCLevel	Enumerated value	Device version. The definition of the enumeration value is shown in Table C.18 FRONTEND_DISEQC_VER_E
enPort4	Enumerated value	Four cut one switch. The definition of the enumeration value is shown in Table C.16 FRONTEND_DISEQC_SWITCH_PORT_E
enPort16	Enumerated value	Sixteen cut one switch. The definition of the enumeration value is shown in Table C.16 FRONTEND_DISEQC_SWITCH_PORT_E
enMotorType	Enumerated value	Motor Agreement. The definition of the enumeration value is shown in Table C.21 FRONTEND_MOTORTYPE_E
u32Longitude	U32	Satellite longitude
u32MotoPos	U32	Motor position number, store up to 256 motor positions, limited by the storage capacity of motor hardware

C.3.15 Unicable SCR user frequency band information structure

Unicable SCR user frequency band information structure is shown in Table C.64.

Table C.64 – Unicable SCR user frequency band information structure (FRONTEND_UNICABLE_SCR_UB_S)

Attribute name	Type	Description
u32SCRNo	U32	User band number
s32CenterFreq	S32	User band center frequency

C.3.16 LNB scan information acquisition data structure

Unicable LNB scan information acquisition data structure is shown in Table C.65.

Table C.65 – Unicable LNB scan information acquisition data structure (FRONTEND_UNICABLE_USERBANDS_S)

Attribute name	Type	Description
ppUserBandsinfo	pointer	Pointer to user band storage array
pu32Num	pointer	The size of the array, pointer type U32

C.3.17 Get the ISIID parameter structure

Get the ISIID parameter structure is shown in Table C.66.

Table C.66 – Get the ISIID parameter structure Get the ISIID parameter structure

Attribute name	Type	Description
u8StreamNum	U8	Number of streams
u8IsiID	U8	ID of ISI

C.3.18 LNB control parameter structure

LNB control parameter structure is shown in Table C.67.

Table C.67 – LNB control parameter structure (FRONTEND_SAT_LNB_INFO_S)

Attribute name	Type	Description
enCmdType	Enumerated value	Command type. The definition of the enumeration value is shown in Table C.19 FRONTEND_LNB_CMD_TYPE_E
enLnbPower	Enumerated value	lnb power status. The definition of the enumeration value is shown in Table C.8 FRONTEND_LNB_POWER_E
enPolarization	Enumerated value	Polarization. The definition of the enumeration value is shown in Table C.10 FRONTEND_POLARIZATION_E
enLnb12vState	Enumerated value	12 V control state. The definition of the enumeration value is shown in Table C.11 FRONTEND_LNB_12V_E
enLnb22kState	Enumerated value	22 kHz state. The definition of the enumeration value is shown in Table C.13 FRONTEND_LNB_22K_E
enDiseqcCmd	Enumerated value	DiSEqC status. The definition of the enumeration value is shown in Table C.17 FRONTEND_LNB_DISEQC_CMD_E
bDiseqcNeedResponse	BOOL	Whether DiSEqC response is required (DiSEqC2.x). TRUE: response; FALSE: no response
u32DiseqcDataLen	U32	DiSEqC control data length Bytes
pDiseqcData	pointer	DiSEqC control data, pointer type U8

C.3.19 Satellite blind scan TP information structure

Satellite blind scan TP information structure is shown in Table C.68.

Table C.68 – Satellite blind scan TP information structure (FRONTEND_SAT_BLINDSCAN_TP_INFO_S)

Attribute name	Type	Description
u32Freq	U32	Frequency, TP frequency point, unit kilohertz (kHz)
u32Sym	U32	Symbol rate in kilobits per second (kbps)
enModulation	Enumerated value	Modulation mode, 8PSK, QPSK, etc., corresponding to DVB-S1, S2. The definition of the enumeration type is shown in Table C.26 FRONTEND_MODULATION_E
enPolar	Enumerated value	Polarization type. The definition of the enumeration type is shown in Table C.10 FRONTEND_POLARIZATION_E
enIQInt	Enumerated value	IQ mode. The definition of the enumeration type is shown in Table C.5 FRONTEND_IQ_IVT_E

C.3.20 Satellite blind scan result structure

Satellite blind scan result structure is shown in Table C.69.

Table C.69 – Satellite blind scan result structure (FRONTEND_SAT_BLINDSCAN_DATA_S)

Attribute name	Type	Description
u32CenterFreq	U32	The center frequency (MHz) of the current blind scan, used by

Table C.69 – Satellite blind scan result structure (FRONTEND_SAT_BLINDSCAN_DATA_S)

Attribute name	Type	Description
		the App to calculate the blind scan progress
u32TpCnt	U32	Number of TPs found in the current frequency band
pTpInfo	pointer	Information array pointing to the TP transponder. The definition of the point type is shown in Table C.68 FRONTEND_SAT_BLINDSCAN_TP_INFO_S

C.3.21 Tuner Blind Scan Notification Information Structure

Tuner Blind Scan Notification Information Structure is shown in Table C.70.

Table C.70 – Tuner Blind Scan Notification Information Structure (FRONTEND_SAT_BLINDSCAN_NOTIFY_U)

Attribute name	Type	Description
penStatus	pointer	Blind scan status. The definition of the point type is shown in Table C.2 FRONTEND_FE_STATUS_E
pu16ProgressPercent	pointer	Blind scan progress, pointer type U16
pstResult	pointer	Blind scan results. The definition of the point type is shown in Table C.68 FRONTEND_SAT_BLINDSCAN_TP_INFO_S

C.3.22 Satellite Tuner blind scan parameter structure

Satellite Tuner blind scan parameter structure is shown in Table C.71.

Table C.71 – Satellite Tuner blind scan parameter structure (FRONTEND_SAT_BLINDSCAN_PARA_S)

Attribute name	Type	Description
enPolar	Enumerated value	LNB polarization mode, automatic scan mode setting is invalid. The definition of the enumeration value is shown in Table C.10 FRONTEND_POLARIZATION_E
enLNB22K	Enumerated value	LNB 22 kHz state, for Ku-band dual local oscillator LNB, ON selects high local oscillator, OFF selects low local oscillator, automatic scan mode setting is invalid. The definition of the enumeration value is shown in Table C.13 FRONTEND_LNB_22K_E
u32StartFreq	U32	Blind scan start frequency (intermediate frequency), unit: kHz, automatic scan mode setting is invalid
u32StopFreq	U32	Blind scan end frequency (intermediate frequency), unit: kHz, automatic scan mode setting is invalid
pfnDISEQCSet	Callback function pointer	The callback function is defined as follows: VOID (*pfnDISEQCSet) (const HANDLE hFrontend, const FRONTEND_POLARIZATION_E enPolar, const FRONTEND_LNB_22K_E enLNB22K)
pfnEVTNotify	Callback function pointer	The callback function is defined as follows: VOID (*pfnEVTNotify) (const HANDLE hFrontend, const FRONTEND_SAT_BLINDSCAN_EVT_E enEVT,

Table C.71 – Satellite Tuner blind scan parameter structure (FRONTEND_SAT_BLINDSCAN_PARA_S)

Attribute name	Type	Description
		const FRONTEND_SAT_BLINDSCAN_NOTIFY_U* punNotify)

C.3.23 Satellite signal search (frequency lock or blind scan) parameter structure

Satellite signal search (frequency lock or blind scan) parameter structure is shown in Table C.72.

Table C.72 – Satellite signal search (frequency lock or blind scan) parameter structure (FRONTEND_SAT_SCAN_INFO_S)

Attribute name	Type	Description
u32Freq	U32	Downlink frequency, unit kilohertz (kHz)
u32Sym	U32	Symbol rate in thousands of symbols per second (kSyms/s)
enPolar	Enumerated value	Polarization mode， the definition of the enumeration type is shown in Table C.10 FRONTEND_POLARIZATION_E
u32ScrambleValue	U32	The initial value of the physical layer scrambling code, ranging from 0 to 262141, this value is a special signal when it is not 0, and this value can only be notified by the signal sender; when the frequency is not a special signal, the value must be configured as the default value 0
enModulation	Enumerated value	Modulation mode, 8PSK, QPSK, etc., corresponding to DVB-S1, S2. The definition of the enumeration type is shown in Table C.26 FRONTEND_MODULATION_E
u32StopFreq	U32	Sweep cutoff frequency, only effective in blind scan mode (MHz)
enRolloff	Enumerated value	Roll-off factor options. The definition of the enumeration type is shown in Table C.9 FRONTEND_ROLLOFF_E
enIQInt	Enumerated value	IQ mode. The definition of the enumeration type is shown in Table C.5 FRONTEND_IQ_IVT_E
enFecRate	Enumerated value	The definition of the enumeration type is shown in Table C.25 FRONTEND_FEC_RATE_E
u8ChannelIndex	U8	Channel index is the index of ISI [1, max]. When searching at the beginning, use 0, which can be obtained from the interface later. S2 has only one PLP (physical layer pipe), possibly multiple ISI (input stream identifier)
u32SymOffset	U32	Symbol rate shift
u32FreqOffset	U32	Downlink frequency offset
enSearchMod	Enumerated value	Search mode, such as blind scan. The definition of the enumeration type is shown in Table C.4 FRONTEND_SEARCH_MODE_E
pstBindScanParam	pointer	The definition of the point type is shown in Table C.71 FRONTEND_SAT_BLINDSCAN_PARA_S
pstExAntenna	pointer	If the antenna parameters are set, set the antenna parameters first, and then lock the frequency. the definition of the point type is shown in Table C.63 FRONTEND_SAT_EXTRA_ANTENNA_CONFIG_S

C.3.24 Wired signal search (frequency lock) parameter structure

Wired signal search (frequency lock) parameter structure is shown in Table C.73.

Table C.73 – Wired signal search (frequency lock) parameter structure (FRONTEND_CAB_SCAN_INFO_S)

Attribute name	Type	Description
u32Freq	U32	Search frequency (kHz)
u32Sym	U32	Symbol rate (Syms/s)
enModulation	Enumerated value	Modulation. The definition of the enumeration type is shown in Table C.26 FRONTEND_MODULATION_E
enSpectrum	Enumerated value	Spectrum polarity. The definition of the enumeration type is shown in Table C.27 FRONTEND_SPECTRUM_E
enBandWidth	Enumerated value	Bandwidth. The definition of the enumeration type is shown in Table C.28 FRONTEND_BAND_WIDTH_E
enGuardInterval	Enumerated value	guard_interval. The definition of the enumeration type is shown in Table C.30 FRONTEND_GUARD_INTERVAL_E
u8ChannelIndex	U8	The Channel index of DVB-C2 is the index of ISI [1,max]. At the beginning of the search, use 0, which can be obtained from the interface later. C2 may have multiple PLPs (physical layer pipe), and each PLP has only one ISI (input stream identifier)

C.3.25 Ground signal search (frequency locking) parameter structure

Ground signal search (frequency locking) parameter structure is shown in Table C.74.

Table C.74 – Ground signal search (frequency locking) parameter structure (FRONTEND_TER_SCAN_INFO_S)

Attribute name	Type	Description
u32Freq	U32	Search frequency (kHz)
enBandWidth	Enumerated value	Bandwidth. The definition of the enumeration type is shown in Table C.28 FRONTEND_BAND_WIDTH_E
enModulation	Enumerated value	Mode. The definition of the enumeration type is shown in Table C.26 FRONTEND_MODULATION_E
enSpectrum	Enumerated value	Spectrum polarity. The definition of the enumeration type is shown in Table C.27 FRONTEND_SPECTRUM_E
enCoderate	Enumerated value	Coderate. The definition of the enumeration type is shown in Table C.25 FRONTEND_FEC_RATE_E
enGuardInterval	Enumerated value	Interval. The definition of the enumeration type is shown in Table C.30 FRONTEND_GUARD_INTERVAL_E
enTransmitMod	Enumerated value	The definition of the enumeration type is shown in Table C.29 FRONTEND_TRANSMIT_MOD_E
enTer2ChannelMode	Enumerated value	The definition of the enumeration type is shown in Table C.32 FRONTEND_TER2_MODE_E
enTerTsPriority	Enumerated value	TS priority mode. The definition of the enumeration type is shown in Table C.31 FRONTEND_TER_TS_PRIORITY_E
u8ChannelIndex	U8	The channel index of DVB-T2 is the index of PLP [1,max]

C.3.26 ATV search parameter structure

ATV search parameter structure is shown in Table C.75.

Table C.75 – ATV search parameter structure (FRONTEND_ATV_SCAN_INFO_S)

Attribute name	Type	Description
u32Freq	U32	Search frequency (kHz)
enSystem	Enumerated value	ATV system. The definition of the enumeration type is shown in Table C.38 FRONTEND_ATV_SYSTEM_E
enSifBw	Enumerated value	ATV search bandwidth. The definition of the enumeration type is shown in Table C.39 FRONTEND_ATV_SIF_BW_E
enConnectWorkMode	Enumerated value	ATV working mode. The definition of the enumeration type is shown in Table C.40 FRONTEND_ATV_CONNECT_WORK_MODE_E

C.3.27 ATV signal information structure

ATV signal information structure is shown in Table C.76.

Table C.76 – ATV signal information structure (FRONTEND_ATV_SIGNALINFO_S)

Attribute name	Type	Description
bVifLock	BOOL	Whether the intermediate frequency is locked
bAfcWin	BOOL	Whether it in the AFC window
bCarrDet	BOOL	FM sound carrier detection
s32AfcFreq	S32	AFC frequency value in kilohertz (kHz)

C.3.28 DTMB search parameter structure

DTMB search parameter structure is shown in Table C.77.

Table C.77 – DTMB search parameter structure (FRONTEND_DTMB_SCAN_INFO_S)

Attribute name	Type	Description
u32Freq	U32	Search frequency (kHz)
enBandWidth	Enumerated value	Bandwidth. The definition of the enumeration type is shown in Table C.28 FRONTEND_BAND_WIDTH_E
enModulation	Enumerated value	Mode. The definition of the enumeration type is shown in Table C.26 FRONTEND_MODULATION_E
enSpectrum	Enumerated value	Spectrum polarity. The definition of the enumeration type is shown in Table C.27 FRONTEND_SPECTRUM_E
enCarrierMode	Enumerated value	Carrier type. The definition of the enumeration type is shown in Table C.42 FRONTEND_DTMB_CARRIER_MODE_E
enCoderate	Enumerated value	Code rate type. The definition of the enumeration type is shown in Table C.43 FRONTEND_DTMB_CODE_RATE_E
enTimeInterleave	Enumerated value	Time domain interleaving type. The definition of the enumeration type is shown in Table C.44 FRONTEND_DTMB_TIME_INTERLEAVE_E
enGuardInterval	Enumerated value	Guard time type. The definition of the enumeration type is shown in Table C.45 FRONTEND_DTMB_GUARD_INTERVAL_E
u8ChannelIndex	U8	Channel index

C.3.29 Signal search parameter structure

Signal search parameter structure is shown in Table C.78.

Table C.78 – Signal search parameter structure (FRONTEND_SIGNAL_SCAN_INFO_U)

Attribute name	Type	Description
stSatInfo	Structure	Satellite frequency lock information. The definition of the structure is shown in Table C.72 FRONTEND_SAT_SCAN_INFO_S
stCabInfo	Structure	Wired frequency lock information. The definition of the structure is shown in Table C.73 FRONTEND_CAB_SCAN_INFO_S
stTerInfo	Structure	Ter frequency lock information. The definition of the structure is shown in Table C.74 FRONTEND_TER_SCAN_INFO_S
stAtvInfo	Structure	Analog frequency lock information. The definition of the structure is shown in Table C.75 FRONTEND_ATV_SCAN_INFO_S
stDtmbInfo	Structure	Dtmb frequency lock information. The definition of the structure is shown in Table C.77 FRONTEND_DTMB_SCAN_INFO_S

C.3.30 Frequency point information structure

Frequency point information structure is shown in Table C.79.

Table C.79 – Frequency point information structure (FRONTEND_SCAN_INFO_S)

Attribute name	Type	Description
u16TsIndex	U16	TS stream channel
enFrontendType	Enumerated value	Frequency lock type. The definition of the enumeration is shown in Table C.3 FRONTEND_FE_TYPE_E
unScanInfo	Structure	Frequency lock parameters. The definition of the structure is shown in Table C.78 FRONTEND_SIGNAL_SCAN_INFO_U

C.3.31 Frequency related information structure

Frequency related information structure is shown in Table C.80.

Table C.80 – Frequency related information structure (FRONTEND_FRONTEND_STATUS_S)

Attribute name	Type	Description
u32Strength	U32	Signal strength
u32Quality	U32	Signal quality
fBert	float	Code error rate

C.3.32 Unicable blind scan data notification structure

Unicable blind scan data notification structure is shown in Table C.81.

Table C.81 – Unicable blind scan data notification structure (FRONTEND_UNICABLE_SCAN_NOTIFYDATA_U)

Attribute name	Type	Description
penStatus	pointer	Blind scan status
pu16ProgressPercent	pointer	Blind scan progress

C.3.33 Callback function registration parameter structure

Callback function registration parameter structure is shown in Table C.82.

Table C.82 – Callback function registration parameter structure (FRONTEND_REG_CALLBACK_PARAMS_S)

Attribute name	Type	Description
enCallbackType	Enumerated type	Callback function type, determine the following callback prototype
pCallBack	pointer	Callback function pointer
u32UserData	U32	The data the user wants to return
bDisable	BOOL	Whether to enable the callback

C.3.34 Frontend information structure

Frontend information structure is shown in Table C.83.

Table C.83 – Frontend information structure (FRONTEND_INFO_S)

Attribute name	Type	Description
aszDevName	Structure	Frontend module name
u32CallbackNum	U32	Number of callback functions
u32SourceId	U32	Input source ID
enDiseqcVer	Enumerated value	Supported DISEQC version
u32Generation	U32	Which generation of frontend is supported
enDemuxSetArr [DMX_NUMBER_OF_DMX_ID]	Enumerated value	The number of Demux supported by this frontend. The definition of the enumeration value is shown in Table B.4 DMX_ID_E

C.3.35 Frontend capability (multimode Tuner) structure

Frontend capability (multimode Tuner) structure is shown in Table C.84.

Table C.84 – Frontend capability (multimode Tuner) structure (FRONTEND_CAPABILITY_S)

Attribute name	Type	Description
u32TunerNum	U32	Number of Tuners
au32FeCurType [FRONTEND_FE_NUM_MAX]	U32	Frontend current type
au32FeType [FRONTEND_FE_NUM_MAX]	U32	Frontend type

C.3.36 Second-generation frontend structure

Second-generation frontend structure is shown in Table C.85.

Table C.85 – Second-generation frontend structure (FRONTEND_G2_MODFEC_S)

Attribute name	Type	Description
enFecRate	Enumerated value	Forward error correction code rate. The definition of the enumeration is shown in Table C.25 FRONTEND_FEC_RATE_E
enModulation	Enumerated value	Modulation. The definition of the enumeration is shown in Table C.26 FRONTEND_MODULATION_E

C.3.37 Second-generation frontend channel information structure

Second-generation frontend channel information structure is shown in Table C.86.

Table C.86 – Second-generation frontend channel information structure (FRONTEND_G2_CHANNEL_INFO_S)

Attribute name	Type	Description
u8Isi	U8	Input stream ID
enStreamType	Enumerated value	DVB-C2, DVB-T2, DVB-S2 input stream format. The definition of the enumeration is shown in Table C.49 FRONTEND_G2_STREAM_TYPE_E
bCcm	BOOL	Boolean value
bIssActive	BOOL	Boolean value
bNpdActive	BOOL	Boolean value
enRollOff	Enumerated value	The definition of the enumeration is shown in Table C.9 FRONTEND_ROLLOFF_E
stModFec	Structure	The definition of the structure is shown in Table C.85 FRONTEND_G2_MODFEC_S
bShortFrame	BOOL	Boolean value

C.3.38 Channel configuration parameter structure

Channel configuration parameter structure is shown in Table C.87.

Table C.87 – Channel configuration parameter structure (FRONTEND_G2_REQ_PARAMS_S)

Attribute name	Type	Description
stModFec	Structure	The definition of the structure is shown in Table C.85 FRONTEND_G2_MODFEC_S
bPilots	BOOL	TRUE: pilots, FALSE:no pilots

C.3.39 Channel configuration response parameter structure

Channel configuration response parameter structure is shown in Table C.88.

Table C.88 – Channel configuration response parameter structure (FRONTEND_G2_RPN_PARAMS_S)

Attribute name	Type	Description
u8ModFecAck	U8	Channel configuration parameters

C.3.40 Frontend module structure

Frontend module structure is shown in Table C.89.

Table C.89 – Frontend module structure (FRONTEND_MODULE_S)

Attribute name	Type	Description
common	Structure	The definition of the structure is shown in Table B.41 hw_module_t

C.4 The definition of Callback function

C.4.1 Blind scan status/data callback function callback function

The prototype: typedef void (*FRONTEND_SAT_BLINDSCAN_STATUS_PFN) (const HANDLE hFrontend, const FRONTEND_FE_STATUS_E enBlindscanStatus,

const FRONTEND_SAT_BLINDSCAN_DATA_S* const pstBlindscanData, const U32 u32UserData);

Function: Blind scan status/data callback function

Input parameter: hFrontend Frontend handle

enBlindscanStatus Blind scan status

pstBlindscanData Blind scan data

u32UserData User private data

Output parameters: None.

Return: 0: correct; non-zero: error

C.4.2 Lock status notification callback function

The prototype: typedef void (*FRONTEND_NOTIFY_STATUS_PFN) (const HANDLE hFrontend,

const FRONTEND_FE_STATUS_E enOldStatus,

const FRONTEND_FE_STATUS_E enNewStatus,

const U32 u32UserData);

Function: lock state notification callback function, callback when the lock state changes

Input parameter: hFrontend Frontend handle

enOldStatus old status

enNewStatus new status

u32UserData User private data

Output parameters: None.

Return: 0: correct; non-zero: error

C.4.3 Get frontend status callback function

The prototype: typedef void (*FRONTEND_GET_STATUS_PFN) (const HANDLE hFrontend,

const FRONTEND_FRONTEND_STATUS_S* const pFrontendStatus,

const U32 u32UserData);

Function: Get the frontend status callback function.

Input parameter: hFrontend Frontend handle

pFrontendStatus Frontend status

u32UserData User private data

Output parameters: None.

Return: 0: correct; non-zero: error

C.4.4 Notify LNB\PWR status callback function

The prototype: typedef void (*FRONTEND_NOTIFY_LNB_PWR_STATUS_PFN) (const HANDLE hFrontend, const FRONTEND_LNB_PWR_STATUS_E enOldStatus,

const FRONTEND_LNB_PWR_STATUS_E enNewStatus, const U32 u32UserData);

Function: notify LNB\PWR status callback function

Input parameter: hFrontend Frontend handle

enOldStatus old status

enOldStatus old status

u32UserData User private data

Output parameters: None.

Return: 0: correct; non-zero: error

C.4.5 Frontend search callback function

The prototype:

typedef void (*FRONTEND_UNICABLE_SCAN_PFN)

(const HANDLE hFrontend,

FRONTEND_UNICABLE_SCAN_EVT_E enEVT,

FRONTEND_UNICABLE_SCAN_NOTIFYDATA_U* pData);

Function: FrontEnd Unicable search callback function

Input parameter: hFrontend Frontend handle

enEVT Data type returned

pData Returned data, combined data type

Output parameters: None.

Return: 0: correct; non-zero: error

C.5 Call method

The hardware abstract interface calling method of the Frontend module is shown in Figure C.1.

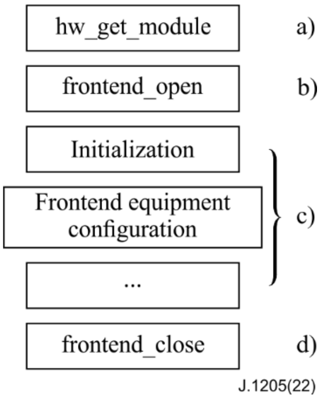

Figure C.1 – Frontend module hardware abstract interface calling method

Figure C.1

a) Call the hw_get_module() interface to get the HAL Stub of the Frontend module:
hw_get_module(FRONTEND_HARDWARE_MODULE_ID, &g_frontend_module)

b) Call frontend_open (g_frontend_module, & pstDevice) to get the device handle of the Frontend module:
frontend_open (g_frontend_module, & pstDevice)

c) Control the Frontend hardware through a series of interface functions provided by the device handle

d) After finishing the hardware manipulation, you should call the frontend_close() interface to close the Frontend device to avoid resource leakage

C.6 Definition of Interface

C.6.1 "Open Frontend device" interface

The prototype: static inline int frontend_open (const struct hw_module_t* pstModule, FRONTEND_DEVICE_S** pstDevice);

Function: Open a Frontend module device

Input parameter: hFrontend Frontend module handle

Output parameter: pstDevice Frontend device handle

Return: 0: correct; non-zero: error

C.6.2 "Close the Frontend device" interface

The prototype: static inline int frontend_close (FRONTEND_DEVICE_S* pstDevice);

Function: Close a Frontend module device

Input parameter: pstDevice Frontend device handle

Output parameter: None.

Return: 0: correct; non-zero: error

C.6.3 "Initialize Frontend device" interface

The prototype: S32 (*frontend_init) (struct _FRONTEND_DEVICE_S*pstDev, const FRONTEND_INIT_

PARAMS_S *const pstInitParams)；

Function: Frontend initialization

Input parameter: pstDev Frontend device handle

pstInitParams initialization parameters

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.4 "Open the Frontend instance" interface

The prototype: S32 (*frontend_open) (struct _FRONTEND_DEVICE_S *pstDev, HANDLE *const phFrontend, const FRONTEND_OPEN_PARAMS_S *const pOpenParams)；

Function: Open a Frontend instance

Input parameter: pstDev Frontend device handle

phFrontend Frontend example handle

pOpenParams open parameters

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.5 "Close the Frontend instance" interface

The prototype: S32 (*frontend_close) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE frontend_handle)

Function: Close a Frontend instance.

Input parameter: pstDev Frontend device handle

frontend_handle Frontend example handle

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.6 "Deinitialize the Frontend instance" interface

The prototype: S32 (*frontend_term) (struct _FRONTEND_DEVICE_S *pstDev, const FRONTEND_TERM_

PARAMS_S *const pstTermParams);

Function: Deinitialize the Frontend instance

Input parameter: pstDev Frontend device handle

pstTermParams Deinitialize parameters

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.7 "Get Frontend's current scan status" interface

The prototype: S32 (*frontend_get_scan_info) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, FRONTEND_SCAN_INFO_S *const pstScanInfo)；

Function: Get the current scan status of Frontend

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameters: pstScanInfo Scan information

Return: 0: correct; non-zero: error

C.6.8 "Set LNB" interface

The prototype: S32 (*frontend_sat_config_lnb) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE

hFrontend, const FRONTEND_SAT_LNB_INFO_S * const pstLnbInfo);

Function: Set LNB.

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

pstLnbInfo LNB information.

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.9 "Get LNB power status" interface

The prototype: S32 (*frontend_get_lnb_pwr_status) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, FRONTEND_LNB_PWR_STATUS_E *const penLnbPwrStatus);

Function: Get LNB power status.

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameters: penLnbPwrStatus LNB power status

Return: 0: correct; non-zero: error

C.6.10 "Start scan" interface

The prototype: S32 (*frontend_start_scan) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE

hFrontend, const FRONTEND_SCAN_INFO_S * const pstScanParams, const BOOL bSynch, const U32 u32Timeout);

Function: frequency lock or blind scan.

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

pstScanParams Frequency lock parameters, blind scan parameters

bSynch Whether to synchronize frequency lock, blind scan is invalid

u32Timeout Timeout period

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.11 "Stop scan" interface

The prototype: S32 (*frontend_abort) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend);

Function: stop frequency lock or blind scan.

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.12 "Register the callback function with Frontend" interface

The prototype: S32 (*frontend_register_callback) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, const FRONTEND_REG_CALLBACK_PARAMS_S* const pstRegParams)

Function: Register the callback function.

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

pstRegParams Callback function parameters

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.13 "Configure callback function" interface

The prototype: S32 (*frontend_config_callback) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, const void * const pCallback, const FRONTEND_CFG_CALLBACK_E enCallbackCfg);

Function: Configure the callback function.

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

pCallback Callback function pointer

enCallbackCfg Whether the parameters of the callback function works or not

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.14 "Frequency lock" interface

The prototype: S32 (*frontend_lock) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend);

Function: Synchronous frequency lock.

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameters: None.

Return: 0: correct; non-zero: error

C.6.15 "Get code error rate" interface

The prototype: S32 (*frontend_get_bert) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, float *const pfBert);

Function: Get code error rate

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameter: pfBert Code error rate

Return: 0: correct; non-zero: error

C.6.16 "Get signal quality" interface

The prototype: S32 (*frontend_get_signal_quality) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, U32 *const pu32Quality);

Function: Get signal quality

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameter: pu32Quality Signal quality.

Return: 0: correct; non-zero: error

C.6.17 "Get signal strength" interface

The prototype: S32 (*frontend_get_signal_strength) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, U32 *const pu32Strength)

Function: Get signal strength

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameter: pu32Strength signal strength

Return: 0: correct; non-zero: error

C.6.18 "Get signal information" interface

The prototype: S32 (*frontend_get_atvsignalinfo) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, FRONTEND_ATV_SIGNALINFO_S* const pstSignalInfo);

Function: Get signal information

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameter: pstSignalInfo Signal information

Return: 0: correct; non-zero: error

C.6.19 "Get signal frequency lock information" interface

The prototype: S32 (*frontend_get_connect_status) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, FRONTEND_FE_STATUS_E * const penStatus);

Function: Get signal frequency lock information

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameter: penStatus Signal frequency lock information

Return: 0: correct; non-zero: error

C.6.20 "Get Frontend information" interface

The prototype: S32 (*frontend_get_info) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, FRONTEND_INFO_S * const pstInfo);

Function: Get all information of frontend

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameter: pstInfo   Frontend all information

Return: 0: correct; non-zero: error

C.6.21 "Get Frontend Capability" Interface

The prototype: S32 (*frontend_get_capability) (struct _FRONTEND_DEVICE_S *pstDev, FRONTEND_

CAPABILITY_S * const pstCapability);

Function: Get frontend capability

Input parameter: pstDev   Frontend device handle

Output parameter: pstCapability   Frontend capabilities

Return: 0: correct; non-zero: error

C.6.22 "Get channel number" interface

The prototype: S32 (*frontend_get_channel_num) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, U8 * const pu8ChannelNum);

Function: Get the number of channels

Input parameter: pstDev   Frontend device handle

hFrontend   Frontend example handle

Output parameter: pu8ChannelNum   Number of data channels

Return: 0: correct; non-zero: error

C.6.23 "Get channel information" interface

The prototype: S32 (*frontend_get_channel_info) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, const U8 u8ChannelIndex, FRONTEND_G2_CHANNEL_INFO_S * const pstChannelInfo, const U32 u32Timeout);

Function: Get channel information。

Input parameter: pstDev   Frontend device handle

hFrontend   Frontend example handle

u8ChannelIndex   Channel index

u32Timeout   Timeout period

Output parameter: pstChannelInfo   Channel information

Return: 0: correct; non-zero: error

C.6.24 "Set channel information" interface

The prototype: S32 (*frontend_config_channel) (struct _FRONTEND_DEVICE_S *pstDev, const HANDLE hFrontend, const U8 u8ChannelIndex, const FRONTEND_G2_REQ_PARAMS_S * const pstReqParams, FRONTEND_G2_RPN_PARAMS_S * const pstRpnParams, const U32 u32Timeout);

Function: Set channel information

Input parameter: pstDev   Frontend device handle

hFrontend   Frontend example handle

u8ChannelIndex   Channel index

pstReqParams channel setting parameters

u32Timeout timeout

Output parameter: pstRpnParams feedback information

Return: 0: correct; non-zero: error

C.6.25 "Get frequency lock status" interface

The prototype: S32 (*frontend_atv_get_lock_status) (struct _FRONTEND_DEVICE_S *pstDev, HANDLE hFrontend, FRONTEND_ATV_LOCK_STATUS_E* penLockStatus)；

Function: Get the frequency lock status.

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

Output parameter: penLockStatus Frequency lock status.

Return: 0: correct; non-zero: error

C.6.26 "Fine-tune frequency of the tunner" interface

The prototype: S32 (*frontend_atv_fineTune) (struct _FRONTEND_DEVICE_S *pstDev, HANDLE hFrontend, S32 s32Steps);

Function: fine-tune the frequency of the tunner

Input parameter: pstDev Frontend device handle

hFrontend Frontend example handle

s32Steps Fine tuning frequency offset

Output parameters: None.

Return: 0: correct; non-zero: error

Annex D

System module

(This annex forms an integral part of this Recommendation.)

This annex defines the hardware abstraction layer interface of the system module. The definitions of the basic data types and operators are given in clause 6.

D.1 Constant definition

The definition of constants is shown in Table D.1.

Table D.1 – Constant definition

Constant	Description
const SYSTEM_HARDWARE_MODULE_ID = "systemt"	system module ID
const SYSTEM_HARDWARE_SYSTEM0 = "system0"	system device ID
const SYSTEM_CHIP_ID_LENGTH = "256"	Chip ID length
const SYSTEM_STANDBY_WKUP_KEY_MAXNUM = "8"	Configurable maximum number of standby wakeup keys

D.2 Enumeration definition

D.2.1 Enumeration definition of System working mode

Enumeration definition of System working mode is shown in Table D.2.

Table D.2 – Enumeration definition of System working mode (SYSTEN_SYSTEM_MODE_E)

Type name	Value	Description
SYSTEM_SYSTEM_MODE_NORMAL	0	Normal working mode
SYSTEM_SYSTEM_MODE_SLOW	1	Low power operating mode
SYSTEM_SYSTEM_MODE_STANDBY	2	True standby mode
SYSTEM_SYSTEM_MODE_BUTT	3	System working mode enumeration maximum

D.2.2 Enumeration definition of Key type

Enumeration definition of Key type is shown in Table D.3.

Table D.3 – Enumeration definition of Key type (SYSTEM_KEY_TYPE_E)

Type name	Value	Description
KEY_TYPE_IR	0	Infrared remote control button
KEY_TYPE_PANEL	1	Front panel buttons
KEY_TYPE_BUTT	2	Key type enumeration maximum

D.3 Definition of Data structure

D.3.1 System initialization parameter structure

System initialization parameter structure is shown in Table D.4.

Table D.4 – System initialization parameter structure (SYSTEM_INIT_PARAMS_S)

Attribute name	Type	Description
u32MemSize	U32	Total system memory size (Byte)
u32SysMemSize	U32	Operating system and device driver memory size (Byte)
u32DmxMemSize	U32	Demux drive memory size (Byte)
u32AvMemSize	U32	AV memory size (Byte)
u32UsrShareMemSize	U32	User shared memory size (Byte)

D.3.2 System deinitialization the parameter structure

System deinitialization the parameter structure is shown in Table D.5.

Table D.5 – System deinitialization the parameter structure (SYSTEM_TERM_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved parameters

D.3.3 Key value structure

Key value structure is shown in Table D.6.

Table D.6 – Key value structure (SYSTEM_KEY_CODE_S)

Attribute name	Type	Description
u32KeyLowerValue	U32	Low key value
u32KeyUpperValue	U32	High key value

D.3.4 Key information structure

Key information structure is shown in Table D.7.

Table D.7 – Key information structure (SYSTEM_KDB_KEY_DATA_S)

Attribute name	Type	Description
bExist	BOOL	Whether the configuration is valid
enType	Enumerated value	Button type, remote control or panel buttons
stCode	Structure	Key value received

D.3.5 Time information structure

Time information structure is shown in Table D.8.

Table D.8 – Time information structure (SYSTEM_TIME_S)

Attribute name	Type	Description
u64TimeSec	U64	Time value, expressed in uniform time

D.3.6 System wake-up parameter structure

System wake-up parameter structure is shown in Table D.9.

Table D.9 – System wake-up parameter structure (SYSTEM_WAKEUPINFO_S)

Attribute name	Type	Description
stWakeupKey	Structure	Wake-up key information. The definition of structure is shown in Table D.7 SYSTEM_KDB_KEY_DATA_S
stStandbyPeriodTime	Structure	Standby event. The definition of structure is shown in Table D.8 SYSTEM_TIME_S

D.3.7 System standby configuration parameter structure

System standby configuration parameter structure is shown in Table D.10.

Table D.10 – System standby configuration parameter structure (SYSTEM_STANDBY_PARA_S)

Attribute name	Type	Description
enSystemMode	Enumerated value	System working mode
astStandbyKey [SYSTEM_STANDBY_WKUP_KEY_MAXNUM]	Structure	Standby wake key configuration
bAutoWakeup	BOOL	Whether to wake-up automatically
bDispTimeEnable	BOOL	Whether the front panel displays the time during standby
stCurrTime	Structure	System current time
stAlarmTime	Structure	Automatic system wake-up time
bWifiPowerOn	BOOL	Wi-Fi power supply in standby
bCmPowerOn	BOOL	Whether the cable modem is powered during standby
u32Dummy	U32	Reserved parameters

D.3.8 Chip description information structure

Chip description information structure is shown in Table D.11.

Table D.11 – Chip description information structure (SYSTEM_CHIP_ID_S)

Attribute name	Type	Description
u32ActLen	U32	The actual length of the chip ID
au8ChipIdBuf [SYSTEM_CHIP_ID_LENGTH]	U8	Chip ID cache

D.3.9 System module structure

System module structure is shown in Table D.12.

Table D.12 – System module structure (SYSTEM_MODULE_S)

Attribute name	Type	Description
stCommon	Structure	The definition of structure is shown in Table B.41 hw_module_t

D.4 Definition of Callback function

None.

D.5 Call method

The hardware abstract interface calling method of the system module is shown in Figure D.1.

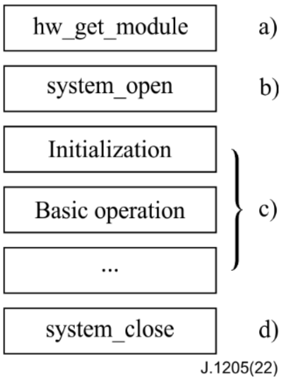

Figure D.1 – System module hardware abstract interface calling method

Figure D.1

a) Call the hw_get_module() interface to get the HAL Stub of the system module: hw_get_module(SYSTEM_HARDWARE_MODULE_ID, &g_system_module)

b) Call system_open (g_system_module,&pstDevice) to get the device handle of the system module: system_open (g_system_module,&pstDevice)

c) Control system hardware through a series of interface functions provided by the device handle

d) After completing the hardware manipulation, call the system_close() interface to close the system device to avoid resource leakage.

D.6 Definition of interface

D.6.1 "Open the system module device" interface

The prototype: static inline int system_open (const struct hw_module_t* pstModule, SYSTEM_DEVICE_S** pstDevice);

Function: Open a system module device

Input parameter: pstModule   system module handle

Output parameter: pstDevice   System device handle.

Return: 0: correct; non-zero: error

D.6.2 "Close the system module device" interface

The prototype: static inline int system_close (SYSTEM_DEVICE_S* pstDevice)；

Function: Close a system module device.

Input parameter: pstDevice system device handle.

Output parameters: None.

Return: 0: correct; non-zero: error

D.6.3 "system device initialization" interface

The prototype: S32 (*system_init) (struct _SYSTEM_DEVICE_S* pstDev, const SYSTEM_INIT_PARAMS_S * const pstInitParams);

Function: system initialization, board-level development kit initialization and allocation of necessary resources, other module functions can only be used after this module is initialized.

Input parameter: pstDev system device handle;

pstInitParams initialization parameters.

Output parameters: None.

Return: 0: correct; non-zero: error

D.6.4 "System deinitialization" interface

The prototype: S32 (*system_term) (struct _SYSTEM_DEVICE_S* pstDev, const SYSTEM_TERM_PARAMS_S * const pstTermParams);

Function: System deinitialization, board-level development kit to initialize and release occupied resources, it should be called after other modules are initialized.

Input parameter: pstDev system device handle;

pstTermParams Termination module parameters.

Output parameters: None.

Return: 0: correct; non-zero: error

D.6.5 "Switch to standby mode interface" interface

The prototype: S32 (*system_switch_standby) (struct _SYSTEM_DEVICE_S* pstDev, const SYSTEM_STANDBY_PARA_S* const pstPara, SYSTEM_WAKEUPINFO_S* pstWakeupInfo);

Function: switch to standby mode interface.

Input parameter: pstDev system device handle;

pstPara standby parameters

Output parameter: pstWakeupInfo standby wake-up parameter

Return: 0: correct; non-zero: error

D.6.6 "Get chip ID information" interface

The prototype: S32 (*system_get_chip_id) (struct _SYSTEM_DEVICE_S* pstDev, SYSTEM_CHIP_ID_S * const pstChipId);

Function: Get chip ID information

Input parameter: pstDev system device handle;

Output parameter: pstChipId chip ID information

Return: 0: correct; non-zero: error

D.6.7 "System restart" interface

The prototype: S32 (*system_sys_reboot) (struct _SYSTEM_DEVICE_S* pstDev, const U32 u32TimeMs)

Function: System restart.

Input parameter: pstDev system device handle;

u32TimeMs system restart time

Output parameters: None.

Return: 0: correct; non-zero: error

D.6.8 "System shutdown" interface

The prototype: S32 (*system_sys_halt) (struct _SYSTEM_DEVICE_S* pstDev, const U32 u32TimeMs);

Function: System shutdown.

Input parameter: pstDev system device handle;

u32TimeMs System shutdown time.

Output parameters: None.

Return: 0: correct; non-zero: error

Annex E

Vout module

(This annex forms an integral part of this Recommendation.)

This annex defines the hardware abstraction layer interface of the video output module. The definitions of the basic data types and operators are given in clause 6.

E.1 Constant definition

The definition of constants is shown in Table E.1.

Table E.1 – Constant definition

Constant	Description
Type of business	
const VOUT_HARDWARE_MODULE_ID = "video_output"	Video output module ID
const VOUT_HARDWARE_VOUT0 = "vout0"	Video output device ID
const VOUT_ALPHA_MAX = "100"	The maximum of transparency
const VOUT_ALPHA_MIN = "0"	The minimum of transparency
const VOUT_BRIGHTNESS_MAX = "100"	The maximum of brightness range
const VOUT_BRIGHTNESS_MIN = "0"	The minimum of brightness range
const VOUT_CONTRAST_MAX = "100"	The maximum of contrast range
const VOUT_CONTRAST_MIN = "0"	The minimum of contrast range
const VOUT_SATURATION_MAX = "100"	The maximum of saturation range
const VOUT_SATURATION_MIN = "0"	The minimum of saturation range
const VOUT_HUE_MAX = "100"	The maximum of hue range
const VOUT_HUE_MIN = "0"	The minimum of hue range
const VOUT_DOF_MAX = "100"	The maximum of depth of field
const VOUT_DOF_MIN = "0"	The minimum of depth of field

E.2 Enumeration definition

E.2.1 Enumeration definition of display channel

Enumeration definition of display channel is shown in Table E.2.

Table E.2 – Enumeration definition of display channel (VOUT_DISPLAY_CHANNEL_E)

Type name	Value	Description
VOUT_DISPLAY_HD0	0	HD0
VOUT_DISPLAY_HD1	1	HD1
VOUT_DISPLAY_HD2	2	HD2
VOUT_DISPLAY_SD0	4	SD0
VOUT_DISPLAY_SD1	8	SD1
VOUT_DISPLAY_SD2	16	SD2

Table E.2 – Enumeration definition of display channel (VOUT_DISPLAY_CHANNEL_E)

Type name	Value	Description
VOUT_DISPLAY_MAX	32	Enumerated maximum value of display channel

E.2.2 Enumeration definition of video output event

Enumeration definition of video output event is shown in Table E.3.

Table E.3 – Enumeration definition of video output event (VOUT_EVT_E)

Type name	Value	Description
VOUT_EVT_BASE	0	Video output event reference value
VOUT_HDMI_EVT_BASE	0	HDMI output event reference value
VOUT_HDMI_EVT_PLUGIN	0	HDMI insertion
VOUT_HDMI_EVT_UNPLUG	1	HDMI unplug
VOUT_HDMI_EVT_EDID_FAIL	2	Failed to get EDID
VOUT_HDMI_EVT_HDCP_FAIL	3	Failed to set HDCP
VOUT_HDMI_EVT_HDCP_SUCCESS	4	Set HDCP successfully
VOUT_EVT_BUTT	5	Enumerated maximum value of video output event

E.2.3 Enumeration definition of video output frame rate type

Enumeration definition of video output frame rate type is shown in Table E.4.

Table E.4 – Enumeration definition of video output frame rate type (VOUT_VID_FRAME_RATE_E)

Type name	Value	Description
VOUT_VID_FRAME_RATE_UNKNOWN	0	Unknown frame rate
VOUT_VID_FRAME_RATE_AUTO	1	Auto setting mode
VOUT_VID_FRAME_RATE_23_976	2	23.976 frames
VOUT_VID_FRAME_RATE_24	4	24 frames
VOUT_VID_FRAME_RATE_25	8	25 frames
VOUT_VID_FRAME_RATE_29_97	16	29.97 frames
VOUT_VID_FRAME_RATE_30	32	30 frames
VOUT_VID_FRAME_RATE_50	64	50 frames
VOUT_VID_FRAME_RATE_59_94	128	59.94 frames
VOUT_VID_FRAME_RATE_60	256	60 frames

E.2.4 Enumeration definition of video output resolution setting

Enumeration definition of video output resolution setting is shown in Table E.5.

Table E.5 – Enumeration definition of video output resolution setting (VOUT_FORMAT_E)

Type name	Value	Description
VOUT_FORMAT_AUTO	0	Auto mode
VOUT_FORMAT_PAL	1	PALD Resolution
VOUT_FORMAT_NTSC	2	NTSC Resolution
VOUT_FORMAT_PALN	3	PALN Resolution
VOUT_FORMAT_PALM	4	PALM Resolution
VOUT_FORMAT_SECAM	5	SECAM Resolution
VOUT_FORMAT_480P	6	480p
VOUT_FORMAT_576P	7	576p
VOUT_FORMAT_HD_720P	8	720p
VOUT_FORMAT_HD_1080I	9	1080i
VOUT_FORMAT_HD_1080P	10	1080p
VOUT_FORMAT_HD_3840X2160	11	3840x2160
VOUT_FORMAT_HD_4096X2160	12	4096x2160
VOUT_FORMAT_SHV_8192X4096	13	8192x4096
VOUT_FORMAT_480I	14	480i
VOUT_FORMAT_576I	15	576i
VOUT_FORMAT_UNKNOWN	16	Unknown Resolution
VOUT_FORMAT_BUTT	17	Enumerated maximum value of video output resolution setting

E.2.5 Enumeration definition of video aspect ratio

Enumeration definition of video aspect ratio is shown in Table E.6.

Table E.6 – Enumeration definition of video aspect ratio (VOUT_ASPECT_RATIO_E)

Type name	Value	Description
VOUT_ASPECT_RATIO_AUTO	0	Automatically choose
VOUT_ASPECT_RATIO_16TO9	1	16:9
VOUT_ASPECT_RATIO_4TO3	2	4:3
VOUT_ASPECT_RATIO_UNKNOWN	3	Unknown aspect ratio, configured to get the stream status to get the unknown aspect ratio
VOUT_NB_OF_ASPECT_RATIO	4	Enumerated maximum value of video aspect ratio

E.2.6 Enumeration definition of video output type

Enumeration definition of video output type is shown in Table E.7.

Table E.7 – Enumeration definition of video output type (VOUT_OUTPUT_TYPE_E)

Type name	Value	Description
VOUT_OUTPUT_TYPE_NONE	0	SD analog composite output
VOUT_OUTPUT_TYPE_COMPOSITE	1	SD analog component output
VOUT_OUTPUT_TYPE_YPBPR	2	SD output
VOUT_OUTPUT_TYPE_SVIDEO	4	DVI
VOUT_OUTPUT_TYPE_DVI	8	DVI
VOUT_OUTPUT_TYPE_HDMI	VOUT_OUTPUT_TYPE_DVI	HDMI
VOUT_OUTPUT_TYPE_SCART	0x10	SCART
VOUT_OUTPUT_TYPE_VGA	0x20	VGA
VOUT_OUTPUT_TYPE_RF	0x40	RF
VOUT_OUTPUT_TYPE_YCBCR	0x80	YCBCR
VOUT_OUTPUT_TYPE_HD_YUV	0x100	YUV
VOUT_OUTPUT_TYPE_HDMI_RGB888	0x200	RGB888
VOUT_OUTPUT_TYPE_HDMI_YCBCR444	0x400	YCBCR444
VOUT_OUTPUT_TYPE_HDMI_YCBCR422	0x800	YCBCR422
VOUT_OUTPUT_TYPE_RGB	0x1000	RGB
VOUT_OUTPUT_TYPE_HD_RGB	0x2000	RGB
VOUT_OUTPUT_TYPE_PANNEL	0x4000	PANNEL
VOUT_OUTPUT_TYPE_ALL	(S32)0xffffffff	Enumerated maximum value of video output type

E.2.7 Enumeration definition of VBI CGMS type

Enumeration definition of VBI CGMS type is shown in Table E.8.

Table E.8 – Enumeration definition of VBI CGMS type (VOUT_VBI_CGMS_TYPE_E)

Type name	Value	Description
VOUT_VBI_CGMS_A	0	CGMS_A permission type
VOUT_VBI_CGMS_B	1	CGMS_B permission type
VOUT_VBI_CGMS_BUTT	2	Enumerated maximum value of VBI CGMS type

E.2.8 Enumeration definition of CGMS_A permission type

Enumeration definition of CGMS_A permission type is shown in Table E.9.

Table E.9 – Enumeration definition of CGMS_A permission type (VOUT_VBI_CGMS_A_COPY_E)

Type name	Value	Description
VOUT_VBI_CGMS_A_COPY_PERMITTED	0	Unlimited copy

Table E.9 – Enumeration definition of CGMS_A permission type (VOUT_VBI_CGMS_A_COPY_E)

Type name	Value	Description
VOUT_VBI_CGMS_A_COPY_ONE_TIME_BEEN_MADE	1	Already copied one time
VOUT_VBI_CGMS_A_COPY_ONE_TIME	2	Can only be copied one time
VOUT_VBI_CGMS_A_COPY_FORBIDDEN	3	Forbid copy
VOUT_VBI_CGMS_A_BUTT	4	Enumerated maximum value of CGMS_A permission type

E.2.9 Enumeration definition of 3D display mode

Enumeration definition of 3D display mode is shown in Table E.10.

Table E.10 – Enumeration definition of 3D display mode (VOUT_3D_FORMAT_E)

Type name	Value	Description
VOUT_3D_FORMAT_2D	0	2D mode
VOUT_3D_FORMAT_FP	1	Frame encapsulation
VOUT_3D_FORMAT_SBS	2	Side by side, left and right half
VOUT_3D_FORMAT_TAB	3	Up and down mode
VOUT_3D_FORMAT_FA	4	Field interleaving
VOUT_3D_FORMAT_LA	5	Line staggered
VOUT_3D_FORMAT_SBS_FULL	6	Side by side, left and right audience
VOUT_3D_FORMAT_L_DEPTH	7	L+DEPTH
VOUT_3D_FORMAT_LBL_LR	8	Line staggered, left eye first
VOUT_3D_FORMAT_LBL_RL	9	Line staggered, right eye first
VOUT_3D_FORMAT_L_DEPTH_GRAPHISC_DEPTH	10	L+depth+Graphics+Graphics-depth
VOUT_3D_FORMAT_BUTT	11	Enumerated maximum value of 3D display mode

E.2.10 Enumeration definition of 3D video playback mode

Enumeration definition of 3D video playback mode is shown in Table E.11.

Table E.11 – Enumeration definition of 3D video playback mode (VOUT_3D_MODE_E)

Type name	Value	Description
VOUT_3D_MODE_2D	0	2D mode playback
VOUT_3D_MODE_2DTO3D	1	2D to 3D mode playback
VOUT_3D_MODE_3D	2	3D mode playback
VOUT_3D_MODE_BUTT	3	Enumerated maximum value of 3D video playback mode

E.2.11 Enumeration definition of video channel delay

Enumeration definition of video channel delay is shown in Table E.12.

Table E.12 – Enumeration definition of video channel delay (VOUT_WINDOW_CHANNEL_E)

Type name	Value	Description
VOUT_WINDOW_HIGHQUALITY	0	High quality access
VOUT_WINDOW_LOWQUALITY	1	Low quality access
VOUT_WINDOW_MAX	2	Enumerated maximum value of video channel delay

E.2.12 Enumeration definition of stop mode

Enumeration definition of stop mode is shown in Table E.13.

Table E.13 – Enumeration definition of stop mode (VOUT_WINDOW_STOP_MODE_E)

Type name	Value	Description
VOUT_WINDOW_STOP_MODE_BLACK	0	Black screen
VOUT_WINDOW_STOP_MODE_FREEZE	1	Still frame

E.2.13 Enumeration definition of video format

Enumeration definition of video format is shown in Table E.14.

Table E.14 – Enumeration definition of video format (VOUT_VIDEO_FORMAT_E)

Type name	Value	Description
VOUT_FORMAT_YUV_SEMIPLANAR_422	0	YUV422
VOUT_FORMAT_YUV_SEMIPLANAR_420	1	YUV420
VOUT_FORMAT_YUV_SEMIPLANAR_400	2	YUV400
VOUT_FORMAT_YUV_SEMIPLANAR_411	3	YUV411
VOUT_FORMAT_YUV_SEMIPLANAR_422_1X2	4	YUV422_1
VOUT_FORMAT_YUV_SEMIPLANAR_444	5	YUV444
VOUT_FORMAT_YUV_SEMIPLANAR_420_UV	6	YUV420, U priority
VOUT_FORMAT_YUV_PACKAGE_UYVY	7	UYVY
VOUT_FORMAT_YUV_PACKAGE_YUYV	8	YUYV
VOUT_FORMAT_YUV_PACKAGE_YVYU	9	YVYU
VOUT_FORMAT_YUV_PLANAR_400	10	YUV400, PLANAR format
VOUT_FORMAT_YUV_PLANAR_411	11	YUV411, PLANAR format
VOUT_FORMAT_YUV_PLANAR_420	12	YUV420, PLANAR format
VOUT_FORMAT_YUV_PLANAR_422_1X2	13	YUV422, 1X2 format
VOUT_FORMAT_YUV_PLANAR_422_2X1	14	YUV422, 2X1 format

Table E.14 – Enumeration definition of video format (VOUT_VIDEO_FORMAT_E)

Type name	Value	Description
VOUT_FORMAT_YUV_PLANAR_444	15	YUV444, PLANAR format
VOUT_FORMAT_YUV_PLANAR_410	16	YUV410, PLANAR format
VOUT_FORMAT_YUV_BUTT	17	Enumerated maximum value of YUV
VOUT_FORMAT_RGB_SEMIPLANAR_444	18	RGB
VOUT_FORMAT_RGB_BUTT	19	Enumerated maximum value of video format

E.2.14 Enumeration definition of video field mode

Enumeration definition of video field mode is shown in Table E.15.

Table E.15 – Enumeration definition of video field mode (VOUT_VIDEO_FIELD_MODE_E)

Type name	Value	Description
VOUT_VIDEO_FIELD_ALL	0	Frame mode
VOUT_VIDEO_FIELD_TOP	1	Top field mode
VOUT_VIDEO_FIELD_BOTTOM	2	Bottom field mode
VOUT_VIDEO_FIELD_BUTT	3	Enumerated maximum value of video field mode

E.2.15 Enumeration definition of 3D frame type

Enumeration definition of 3D frame type is shown in Table E.16.

Table E.16 – Enumeration definition of 3D frame type (VOUT_VIDEO_FRAME_PACKING_TYPE_E)

Type name	Value	Description
VOUT_FRAME_PACKING_TYPE_NONE	0	Non-3D format
VOUT_FRAME_PACKING_TYPE_SIDE_BY_SIDE	1	Left and right mode
VOUT_FRAME_PACKING_TYPE_TOP_AND_BOTTOM	2	Up and down mode
VOUT_FRAME_PACKING_TYPE_TIME_INTERLACED	3	Time-based cross mode Time, one frame for left eye, one frame for right eye
VOUT_FRAME_PACKING_TYPE_FRAME_PACKING	4	Frame mode
VOUT_FRAME_PACKING_TYPE_3D_TILE	5	Tile mode
VOUT_FRAME_PACKING_TYPE_BUTT	6	Enumerated maximum value of 3D frame type

E.2.16 Enumeration definition of window switching mode

Enumeration definition of window switching mode is shown in Table E.17.

Table E.17 – Enumeration definition of window switching mode (VOUT_WINDOW_SWITCH_MODE_E)

Type name	Value	Description
VOUT_WINDOW_SWITCH_MODE_FREEZE	0	Freeze mode
VOUT_WINDOW_SWITCH_MODE_BLACK	1	Black screen
VOUT_WINDOW_SWITCH_MODE_BUTT	2	Enumerated maximum value of window switching mode

E.3 Definition of data structure

E.3.1 Default display setting parameter structure

Default display setting parameter structure is shown in Table E.18.

Table E.18 – Default display setting parameter structure (VOUT_DEFAULT_DISPSETTING_S)

Attribute name	Type	Description
enDispFmt	Enumerated value	Enumerated value of output resolution setting. The definition of enumerated value is shown in Table E.5 VOUT_FORMAT_E
enFrameRate	Enumerated value	Output frame rate setting. The definition of enumeration is shown in Table E.4 VOUT_VID_FRAME_RATE_E
enAspectRatio	Enumerated value	Video aspect ratio setting. The definition of enumeration is shown in Table E.6 VOUT_ASPECT_RATIO_E

E.3.2 Set output coordinate structure

Set output coordinate structure is shown in Table E.19.

Table E.19 – Set output coordinate structure (VOUT_RECT_S)

Attribute name	Type	Description
s32XOffset	S32	Vertex X coordinate
s32YOffset	S32	Vertex Y coordinate
u32Width	U32	Width
u32Height	U32	Height

E.3.3 OSD display area structure

On-screen display (OSD) display area structure is shown in Table E.20.

Table E.20 – OSD display area structure (VOUT_REGION_S)

Attribute name	Type	Description
u32Left	U32	Left margin
u32Top	U32	Top margin
u32Right	U32	Right margin
u32Bottom	U32	Bottom margin

E.3.4 Default display settings structure

Default display settings structure is shown in Table E.21.

Table E.21 – Default display settings structure (VOUT_DISPSETTING_S)

Attribute name	Type	Description
enDispFmt	Enumerated value	Default output resolution setting. The definition of enumeration is shown in Table E.5 VOUT_FORMAT_E
enFrameRate	Enumerated value	Output frame rate setting. The definition of enumeration is shown in Table E.4 VOUT_VID_FRAME_RATE_E
enAutoDispFmt	Enumerated value	Adaptive resolution setting. The definition of enumeration is shown in Table E.5 VOUT_FORMAT_E
enAutoFrameRate	Enumerated value	Adaptive frame rate setting. The definition of enumeration is shown in Table E.4 VOUT_VID_FRAME_RATE_E
enAspectRatio	Enumerated value	Video aspect ratio setting. The definition of enumeration is shown in Table E.6 VOUT_ASPECT_RATIO_E
enAspectRatioConv	Enumerated value	Video aspect ratio adaptive setting
enOutputType	Enumerated value	Display the bound output device. The definition of enumeration is shown in Table E.7 VOUT_OUTPUT_TYPE_E
bOutputEnable	BOOL	Whether there is output, only valid for the current channel. The default is true
u8Hue	U8	Hue, range [0,100]
u8Brightness	U8	Brightness
u8Contrast	U8	Contrast
u8Saturation	U8	Saturation
u83dDof	U8	Depth of field value
en3dFmt	Enumerated value	3D format. The definition of enumerated value is shown in Table E.10 VOUT_3D_FORMAT_E
stOSDVirtualRect	Structure	OSD virtual resolution. The definition of structure is shown in Table E.19 VOUT_RECT_S
stDispOutRegion	Structure	OSD display area. The definition of structure is shown in Table E.20 VOUT_REGION_S
bClearLogo	BOOL	Clear start-up screen

E.3.5 Video module initialization parameter structure

Video module initialization parameter structure is shown in Table E.22.

Table E.22 – Video module initialization parameter structure (VOUT_INIT_PARAMS_S)

Attribute name	Type	Description
bDispEnable	BOOL	Enable screen output at the same time during initialization
astDispSettings [VOUT_DISPLAY_MAX]	Structure	Assign initial value, which can be dynamically modified later. The definition of structure is shown in Table E.21 VOUT_DISPSETTING_S
u32Dummy	U32	For future extension

E.3.6 Structure showing color

Structure showing color is shown in Table E.23.

Table E.23 – Structure showing color (VOUT_BG_COLOR_S)

Attribute name	Type	Description
u8Red	U8	Red component
u8Green	U8	Green component
u8Blue	U8	Blue component

E.3.7 Open the structure of the instantiation handle parameter

Open the structure of the instantiation handle parameter is shown in Table E.24.

Table E.24 – Open the structure of the instantiation handle parameter (VOUT_OPEN_PARAMS_S)

Attribute name	Type	Description
enDispChan	Enumerated value	The display channel that needs to be opened
enOutputType	Enumerated value	Video output type
u32Dummy	U32	Reserved parameters

E.3.8 Close the structure of the instantiation handle parameter

Close the structure of the instantiation handle parameter is shown in Table E.25.

Table E.25 – Close the structure of the instantiation handle parameter (VOUT_CLOSE_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved parameters

E.3.9 Structure of AV module termination parameters

Structure of AV module termination parameters is shown in Table E.26

Table E.26 – Structure of AV module termination parameters (VOUT_TERM_PARAM_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved parameters

E.3.10 Video output capability structure

Video output capability structure is shown in Table E.27.

Table E.27 – Video output capability structure (VOUT_DISP_CAPABILITY_S)

Attribute name	Type	Description
enVoutType	Enumerated value	Video output type. The definition of enumerated value is shown in Table E.7 VOUT_OUTPUT_TYPE_E
au32VidFormat	U32	Video resolution

Table E.27 – Video output capability structure (VOUT_DISP_CAPABILITY_S)

Attribute name	Type	Description
[VOUT_FORMAT_BUTT]		
u32WindowNum	U32	Number of windows supported

E.3.11 Video channel capability structure

Video channel capability structure is shown in Table E.28.

Table E.28 – Video channel capability structure (VOUT_CAPABILITY_S)

Attribute name	Type	Description
enDisplayChannel	Enumerated value	Video output channel. The definition of enumerated value is shown in Table E.2 VOUT_DISPLAY_CHANNEL_E
astDispCapabilityAttr [VOUT_DISPLAY_MAX]	Structure	Video output channel capability. The definition of enumerated value is shown in Table E.27 VOUT_DISP_CAPABILITY_S

E.3.12 The structure of the configuration parameters of the video output callback function

The structure of the configuration parameters of the video output callback function is shown in Table E.29.

Table E.29 – The structure of the configuration parameters of the video output callback function (VOUT_EVT_CONFIG_PARAMS_S)

Attribute name	Type	Description
enEvt	Enumerated value	AV event, which indicates which event this configuration is valid for. The definition of enumerated value is shown in Table E.3 VOUT_EVT_E
pfnCallback	Callback function pointer	The definition of callback function is shown in Table E.4.1 VOUT_CALLBACK_PFN
bEnable	BOOL	Indicates whether to enable the event callback
u32NotificationsToSkip	U32	Indicates that this event needs to be skipped several times before calling the registered callback function

E.3.13 Video channel delay structure

Video channel delay structure is shown in Table E.30.

Table E.30 – Video channel delay structure (VOUT_DELAY_S)

Attribute name	Type	Description
u32PanelMemcDelay	U32	Time delay

E.3.14 Create window parameter structure

Create window parameter structure is shown in Table E.31.

Table E.31 – Create window parameter structure (VOUT_WINDOW_CREATE_PARAMS_S)

Attribute name	Type	Description
enWindowChan	Enumerated value	Window creation delay parameter
bVirtual	BOOL	Reserved parameters

E.3.15 Delete window parameter structure

Delete window parameter structure is shown in Table E.32.

Table E.32 – Delete window parameter structure (VOUT_WINDOW_DESTROY_PARAM_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved parameters

E.3.16 RGB parameter structure

RGB parameter structure is shown in Table E.33.

Table E.33 – RGB parameter structure (VOUT_RGB_COLOR_S)

Attribute name	Type	Description
u32Red	U32	Red
u32Green	U32	Green
u32Blue	U32	Blue

E.3.17 Color temperature parameter structure

Color temperature parameter structure is shown in Table E.34.

Table E.34 – Color temperature parameter structure (VOUT_WINDOW_COLOR_TEMPERATURE_S)

Attribute name	Type	Description
u32RedGain	U32	Red gain
u32GreenGain	U32	Red gain
u32BlueGain	U32	Blue gain
u32RedOffset	U32	Red offset
u32GreenOffset	U32	Green offset
u32BlueOffset	U32	Blue offset

E.3.18 Window state parameter structure

Window state parameter structure is shown in Table E.35.

Table E.35 – Window state parameter structure (VOUT_WINDOW_STATUS_S)

Attribute name	Type	Description
u8WinEnableStatus	U8	Window enable
u8WindowConnected	U8	Link to this window
bWindowEnableMute	BOOL	Mute the window
u32ZOrderIndex	U32	Z order of window
stMuteRGBColor	Structure	Mute color
stColorTemperature	Structure	The colour temperature of the window

E.3.19 Window stop parameter structure

Window stop parameter structure is shown in Table E.36.

Table E.36 – Window stop parameter structure (VOUT_WINDOW_STOP_ATTR_S)

Attribute name	Type	Description
enStopMode	Enumerated value	Stop mode. The definition of enumeration is shown in Table E.13 VOUT_WINDOW_STOP_MODE_E

E.3.20 Window attribute setting parameter structure

Window attribute setting parameter structure is shown in Table E.37.

Table E.37 – Window attribute setting parameter structure (VOUT_WINDOW_USER_DEF_ASPECT_S)

Attribute name	Type	Description
bUserDefAspectRatio	BOOL	Whether to use user-set aspect ratio
u32UserAspectWidth	U32	The user expects to display the video width setting value range between 0~3840, 0 means using the video source resolution
u32UserAspectHeight	U32	The user expects to display the video height setting value range between 0~3840, 0 means using the video source resolution

E.3.21 Window setting parameter structure

Window setting parameter structure is shown in Table E.38.

Table E.38 – Window setting parameter structure (VOUT_WINDOW_SETTINGS_S)

Attribute name	Type	Description
stWindowUserDefAspect	Structure	Setting of window width and height attributes. The definition of structure is shown in Table E.37 VOUT_WINDOW_USER_DEF_ASPECT_S
enAspectRatio	Enumerated value	Setting of Video aspect ratio, definition of enumeration is shown in Table E.6 VOUT_ASPECT_RATIO_E
enAspectRatioConv	Enumerated value	Adaptive setting of Video aspect ratio
u8Alpha	U8	Transparency setting
u8Brightness	U8	Brightness
u8Contrast	U8	Contrast

Table E.38 – Window setting parameter structure (VOUT_WINDOW_SETTINGS_S)

Attribute name	Type	Description
u8Saturation	U8	Saturation
bVirtual	BOOL	Whether it is a virtual window

E.3.22 Frame address structure

Frame address structure is shown in Table E.39.

Table E.39 – Frame address structure (VOUT_FRAME_ADDR_S)

Attribute name	Type	Description
u32YAddr	U32	The address of the Y component data of the current frame
u32CAddr	U32	The address of the Cb component data of the current frame
u32CrAddr	U32	The address of the Cr component data of the current frame
u32YStride	U32	Y-component data span
u32CStride	U32	Cb-component data span
u32CrStride	U32	Cr-component data span

E.3.23 Video stream frame rate structure

Video stream frame rate structure is shown in Table E.40.

Table E.40 – Video stream frame rate structure (VOUT_VCODEC_FRMRATE_S)

Attribute name	Type	Description
u32fpsInteger	U32	The integer part of the frame rate of the stream
u32fpsDecimal	U32	The fractional part of the frame rate of the code stream (3 bits reserved)

E.3.24 Frame information structure

Frame information structure is shown in Table E.41.

Table E.41 – Frame information structure (VOUT_FRAME_INFO_S)

Attribute name	Type	Description
u32FrameIndex	U32	Frame index number in the video sequence
stVideoFrameAddr [2]	Structure	Frame address information. The definition of structure is shown in Table E.39 VOUT_FRAME_ADDR_S
u32Width	U32	Original image width
u32Height	U32	Original image height
s64SrcPts	S64	The original timestamp of the video frame
s64Pts	S64	Timestamp of video frame
u32AspectWidth	U32	Proportional width
u32AspectHeight	U32	Proportional height
stFrameRate	Structure	Frame rate. The definition of structure is shown in Table E.40 VOUT_VCODEC_FRMRATE_S

Table E.41 – Frame information structure (VOUT_FRAME_INFO_S)

Attribute name	Type	Description
enVideoFormat	Enumerated value	Video YUV format
bProgressive	BOOL	scanning method
enFieldMode	Enumerated value	Frame or field coding mode. The definition of enumeration is shown in Table E.15 VOUT_VIDEO_FIELD_MODE_E
bTopFieldFirst	BOOL	Top field priority sign
enFramePackingType	Enumerated value	3D packing type. The definition of enumeration is shown in Table E.16 VOUT_VIDEO_FRAME_PACKING_TYPE_E
u32Circumrotate	U32	Spin sign
bVerticalMirror	BOOL	Vertical mirror sign
bHorizontalMirror	BOOL	Horizontal mirror sign
u32DisplayWidth	U32	Display image width
u32DisplayHeight	U32	Display image height
u32DisplayCenterX	U32	Display the center x coordinate. The upper left corner of the original image is the coordinate origin
u32DisplayCenterY	U32	Display the center y coordinate. The upper left corner of the original image is the coordinate origin
u32ErrorLevel	U32	The error ratio in a decoded image. The value ranges from 0 to 100
bSecurityFrame	BOOL	Safety frame sign
u32Private [32]	U32	Private data

E.3.25 Video output module structure

Video output module structure is shown in Table E.42.

Table E.42 – Video output module structure (VOUT_MODULE_S)

Attribute name	Type	Description
stCommon	Structure	The definition of structure is shown in Table B.41 hw_module_t

E.4 The definition of Callback function

E.4.1 "Video output event callback function" interface

The prototype: typedef VOID (* VOUT_CALLBACK_PFN) (const HANDLE hVout, const VOUT_EVT_E enEvt,

const VOID* const pData);

Function: Video output event callback function

Input parameter: hVout system module handle

enEvt event type

pData The data returned by the event. The definition of data type is shown in Table E.3

Output parameters: None.

Return: 0: correct; non-zero: error

E.5 Call method

The hardware abstract interface call method of the Vout module is shown in Figure E.1.

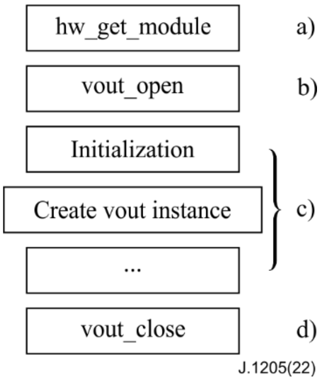

Figure E.1 – Vout module hardware abstract interface calling method

Figure E.1:

a) Call the hw_get_module() interface to get the HAL Stub of the Vout module: hw_get_module(VOUT_HARDWARE_MODULE_ID, &g_vout_module)

b) Call vout_open (g_vout_module,&pstDevice) to get the device handle of the Vout module: vout_open (g_vout_module,&pstDevice)

c) Can control the Vout hardware through a series of interface functions provided by the device handle

d) After completing the hardware manipulation, call the vout_close() interface to close the Vout device to avoid resource leakage.

E.6 Definition of Interface

E.6.1 "Open the video output module device" interface

The prototype: static inline int vout_open (const struct hw_module_t* pstModule, VOUT_DEVICE_S** pstDevice)；

Function: Open a video output module device

Input parameter: pstModule system module handle

Output parameter: pstDevice System device handle.

Return: 0: correct; non-zero: error

E.6.2 "Close the video output module device" interface

The prototype: static inline int vout_close (VOUT_DEVICE_S* pstDevice) ;

Function: Close a video output device

Input parameter: pstDevice Video output handle

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.3 "Video output module initialization" interface

The prototype: S32 (*vout_init) (struct _VOUT_DEVICE_S* pstVoutDev, const VOUT_INIT_PARAMS_S* const

pstInitParams);

Function: The video output module is initialized; the specific business functions need to be initialized before the module can be used normally

Input parameter: pstVoutDev Vout device handle.

pstInitParams Initialization parameters

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.4 "Terminate the video output module" interface

The prototype: S32 (*vout_term) (struct _VOUT_DEVICE_S* pstVoutDev, const VOUT_TERM_PARAM_S *

const pstTermParams);

Function: Terminate the video output module

Input parameter: pstVoutDev Vout device handle.

pstTermParams Termination module parameters

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.5 "Open a video output instance" interface

The prototype: S32 (*vout_open_channel) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE * const phVout,

const VOUT_OPEN_PARAMS_S * const pstOpenParams);

Function: Open a video output instance.

Input parameter: pstVoutDev Vout device handle.

pstOpenParams Example open parameters.

Output parameters: phVout Vout video output instance handle

Return: 0: correct; non-zero: error

E.6.7 "Close a video output instance" interface

The prototype: S32 (*vout_close_channel) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVout,

const VOUT_CLOSE_PARAMS_S * const pstCloseParams);

Function: close a video output instance

Input parameter: pstVoutDev Vout device handle.

hVout Vout Video output instance handle

pstCloseParams Instance shutdown parameters

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.8 "Get module device capability" interface

The prototype: S32 (*vout_get_capability) (struct _VOUT_DEVICE_S* pstVoutDev, VOUT_CAPABILITY_S *

const pstCapability);

Function: Obtain module equipment capabilities

Input parameter: pstVoutDev   Vout device handle.

Output parameters：pstCapability   Vout device capability parameters

Return: 0: correct; non-zero: error

E.6.9 "Configure the parameter of a Vout event" interface

The prototype: S32 (*vout_evt_config) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVout, const

VOUT_EVT_CONFIG_PARAMS_S * const pstCfg);

Function: Configure the parameters of a certain Vout event. The operations that can be performed through this function are regist/remove/disable/enable

1. Each event can independently register and configure its own callback function。
2. Callback function and handle binding.
3. For the same handle, only one callback function can be registered for an event, that is：the callback function registered later will overwrite the original callback function.

Input parameter: pstVoutDev   Vout device handle.

hVout   Vout Video output instance handle

pstCfg   Event configuration parameters

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.10 "Get the configuration parameter of a Vout event" interface

The prototype: S32 (*vout_get_evt_config) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVout,

const VOUT_EVT_E enEvt, VOUT_EVT_CONFIG_PARAMS_S * const pstCfg);

Function: Get the configuration parameters of a certain Vout event.

Input parameter: pstVoutDev   Vout device handle.

hVout   Vout Video output instance handle

enEvt   Event type

Output parameters：pstCfg   Event configuration parameters

Return: 0: correct; non-zero: error

E.6.11 "Close the display channel display function" interface

The prototype: S32 (*vout_outputchannel_mute) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVout, const VOUT_OUTPUT_TYPE_E enOutChannel);

Function: Close the display channel display function

Input parameter: pstVoutDev   Vout device handle

hVout   Vout Video output instance handle

enOutChannel Need to close the output channel

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.12 "Open the display channel display function" interface

The prototype: S32 (*vout_outputchannel_unmute) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE

hVout, const VOUT_OUTPUT_TYPE_E enOutChannel);

Function: Turn on the display channel display function

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

enOutChannel Need to open the output channel

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.13 "Set display parameter" interface

The prototype: S32 (*vout_display_set) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVout, const

VOUT_DISPSETTING_S * const pstSettings);

Function: Set display parameters

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

pstSettings Display parameters

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.14 "Get display parameter" interface

The prototype: S32 (*vout_display_get) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVout,

VOUT_DISPSETTING_S * const pstSettings);

Function: Get display parameters

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

Output parameters：pstSettings Display parameters

Return: 0: correct; non-zero: error

E.6.15 "CGMS start" interface

The prototype: S32 (*vout_vbi_cgms_start) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

VOUT_VBI_CGMS_TYPE_E enCgmsType, VOUT_VBI_CGMS_A_COPY_E enCopyRight);

Function: CGMS start

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

enCgmsType CGMS type

enCopyRight   Copy permission

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.16 "CGMS stop" interface

The prototype: S32 (*vout_vbi_cgms_stop) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout);

Function: CGMS stop

Input parameter: pstVoutDev   Vout device handle

hVout   Vout Video output instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.17 "microvision image setup" interface

The prototype: S32 (*vout_vbi_microvision_setup) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

U8 *pu8MvData);

Function: Set up microvision image.

Input parameter: pstVoutDev   Vout device handle

hVout   Vout Video output instance handle

pu8MvData   Configuration Data

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.18 "Open/close microvision image" interface

The prototype: S32 (*vout_vbi_microvision_enable) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

BOOL bEnable);

Function: Open/close microvision image

Input parameter: pstVoutDev   Vout device handle

hVout   Vout Video output instance handle

bEnable Switch control, True means open

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.19 "Set HDCP parameter" interface

The prototype: S32 (*vout_set_hdcp_params) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

void * const pvData, const U32 u32Length);

Function: Set HDCP parameter

Input parameter: pstVoutDev   Vout device handle hVout Vout Video output instance handle

pvData Parameter data

u32Length Parameter data length.

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.20 "Get HDCP verification status" interface

The prototype: S32 (*vout_get_hdcp_status) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

U32 *pu32Success);

Function: Get HDCP verification status

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

Output parameters: pu32Success HDCP verification status

Return: 0: correct; non-zero: error

E.6.21 "Get EDID raw data" interface

The prototype: S32 (*vout_get_edid) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVout,

U8 * const pu8EdidBuf, U32 * const pu32EdidLen);

Function: Get EDID raw data

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

Output parameters: pu8EdidBuf Edid Data cache

pu32EdidLen Return the length of the original EDID data

Return: 0: correct; non-zero: error

E.6.22 "Set the background color of the video window" interface

The prototype: S32 (*vout_set_bg_color) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

VOUT_BG_COLOR_S *pstVoutBgColor);

Function: Set the background color of the video window

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.23 "Get background color of the video window" interface

The prototype: S32 (*vout_get_bg_color) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

VOUT_BG_COLOR_S *pstVoutBgColor);

Function: Get the background color of the video window

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

Output parameters: pstVoutBgColor Background color information

Return: 0: correct; non-zero: error

E.6.24 "Set 3D mode" interface

The prototype: S32 (*vout_set_3dmode) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

VOUT_3D_MODE_E en3dMode);

Function: Set 3D mode

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

en3dMode 3D mode

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.25 "Get the 3D mode of DISP"interface

The prototype: S32 (*vout_get_3dmode) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

S32 (*vout_get_3dmode) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout,

Function: Get the 3D standard of DISP

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

Output parameters: pen3dMode 3D mode

Return: 0: correct; non-zero: error

E.6.26 "Set 3D output right interchange" interface

The prototype: S32 (*vout_set_3d_lr_switch) (struct _VOUT_DEVICE_S* pstVoutDev, HANDLE hVout, U32

u32Switch);

Function: Set 3D output right interchange

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

u32Switch Right eye priority setting

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.27 "Automatic detection of 3D mode" interface

The prototype: S32 (*vout_autodetect3dformat) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, VOUT_3D_FORMAT_E *pen3dFormat);

Function: Automatic detection of 3D mode

Input parameter: pstVoutDev Vout device handle

hVoutWindow Vout window handle

Output parameters: pen3dFormat Return to 3D mode

Return: 0: correct; non-zero: error

E.6.28 "Create display window" interface

The prototype: S32 (*vout_window_create) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVout, HANDLE *const phVoutWindow, const VOUT_WINDOW_CREATE_PARAMS_S * const pstCreateParams);

Function: Create a display window

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

pstCreateParams Create parameter

Output parameters: phVoutWindow Return to window handle

Return: 0: correct; non-zero: error

E.6.29 "Delete display window" interface

The prototype: S32 (*vout_window_destroy) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVout,

const HANDLE hVoutWindow, const VOUT_WINDOW_DESTROY_PARAM_S * const pstDestroyParams);

Function: Delete a display window

Input parameter: pstVoutDev Vout device handle

hVout Vout Video output instance handle

hVoutWindow Vout window handle

pstDestroyParams Delete the parameters in the window

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.30 "Set window parameter" interface

The prototype: S32 (*vout_window_set) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow,const VOUT_WINDOW_SETTINGS_S * const pstSettings);

Function: Set window parameter

Input parameter: pstVoutDev Vout device handle

hVoutWindow Vout window handle

pstSettings Setting parameter

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.31 "Get window parameter" interface

The prototype: S32 (*vout_window_get) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow,VOUT_WINDOW_SETTINGS_S * const pstSettings);

Function: Get window parameter

Input parameter: pstVoutDev Vout device handle

hVoutWindow Vout window handle

Output parameters:pstSettings Setting parameter

Return: 0: correct; non-zero: error

E.6.32 "Set size of the video input window" interface

The prototype: S32 (*vout_window_set_input_rect) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE

hVoutWindow, AV_COORD_T u32Left, AV_COORD_T u32Top, U32 u32Width, U32 u32Height);
Function:Set the size of the video input window

Input parameter: pstVoutDev Vout device handle

hVoutWindow Vout window handle

u32Left Enter the left coordinate of the window

u32Top Enter the top coordinate of the window

u32Width Input window width

u32Height Input window height

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.33 "Get size of the video input window" interface

The prototype: S32 (*vout_window_get_input_rect) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE

hVoutWindow, AV_COORD_T * const pu32Left, AV_COORD_T * const pu32Top, U32 * const pu32Width,

U32 * const pu32Height);

Function: Get the size of the video input window

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

Output parameters: pu32Left Enter the left coordinate of the window

pu32Top Enter the top coordinate of the window

pu32Width Input window width

u32Height Input window height

Return: 0: correct; non-zero: error

E.6.34 Set the size of the video output window" interface

The prototype: S32 (*vout_window_set_output_rect) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE

hVoutWindow, AV_COORD_T u32Left, AV_COORD_T u32Top, U32 u32Width, U32 u32Height);

Function: Set the size of the video output window

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

pu32Left Enter the left coordinate of the window

pu32Top Enter the top coordinate of the window

pu32Width Input window width

u32Height Input window height

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.35 "Get size of the video output window" interface

The prototype: S32 (*vout_window_get_output_rect) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE

hVoutWindow, AV_COORD_T *const pu32Left, AV_COORD_T *const pu32Top, U32 *const pu32Width, U32

*const pu32Height);

Function: Get the size of the video output window

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

Output parameters: pu32Left Enter the left coordinate of the window

pu32Top Enter the top coordinate of the window

pu32Width Input window width

u32Height Input window height

Return: 0: correct; non-zero: error

E.6.36 "Set size of video content window" interface

The prototype: S32 (*vout_window_set_video_rect) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE

hVoutWindow, AV_COORD_T s32Left, AV_COORD_T s32Top, U32 u32Width, U32 u32Height);

Function: Set the size of the video content window. The video content window refers to the area where the effective content of the video is displayed except the black border in the video output window.

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

pu32Left Enter the left coordinate of the window pu32Top Enter the top coordinate of the window

pu32Width Input window width

u32Height Input window height

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.37 "Set size of video content window"interface

The prototype: S32(*vout_window_get_video_rect) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE

hVoutWindow, AV_COORD_T* const ps32Left, AV_COORD_T* const ps32Top, U32* const pu32Width,

U32* const pu32Height);

Function: Set the size of the video content window. The video content window refers to the area where the effective content of the video is displayed except the black border in the video output window.

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

Output parameters: pu32Left Enter the left coordinate of the window

pu32Top Enter the top coordinate of the window

pu32Width Input window width

u32Height Input window height

Return: 0: correct; non-zero: error

E.6.38 "Get video window status" interface

The prototype: S32 (*vout_window_get_status) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow,VOUT_WINDOW_STATUS_S * const pstStatus);

Function: Get the status of the video window

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

Output parameter: pstStatus returns the status of the window.

Return: 0: correct; non-zero: error

E.6.39 "Pause video playback" interface

The prototype: S32 (*vout_window_freeze) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow);

Function: Pause video playback

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.40 "Resume video playback" interface

The prototype: S32 (*vout_window_unfreeze) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow);

Function: Resume video playback

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.41 "Close window output" interface

The prototype: S32 (*vout_window_mute) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow);

Function: Close window output

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.42 "Open window output" interface

The prototype: S32 (*vout_window_unmute) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow);

Function: Open window output

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.43 "Set background color after the window output is closed" interface

The prototype: S32 (*vout_window_set_mute_color) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE

hVoutWindow, VOUT_RGB_COLOR_S *pstRGBColor);

Function: Set the background color after the window output is closed.

Input parameter: pstVoutDev Vout device handle

hVoutWindow window handle

pstRGBColor The background color after the window output is closed

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.44 "Turn on or off the film mode" interface

The prototype: S32 (*vout_window_enable_filmmode) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE

hVoutWindow, BOOL bEnable);

Function: Turn on or off the film mode

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

bEnable   TRUE: turn on film mode; FALSE, turn off film mode

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.45 "Set the color temperature of the window" interface

The prototype: S32 (*vout_window_set_colortemperature) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, VOUT_WINDOW_COLOR_TEMPERATURE_S *pstColorTemperature);

Function: Set the color temperature of the window

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

pstColorTemperature   color temperature data

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.46 "Set the upper and lower order of the window" interface

The prototype: S32 (*vout_window_set_zorder) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, U32 u32ZOrderIndex);

Function: Set the upper and lower order of the window

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

u32ZOrderIndex   upper and lower order of the window

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.47 "Set panoramic mode" interface

The prototype: S32 (*vout_window_enable_panorama) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, BOOL bEnable);

Function: Set the panoramic mode

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

bEnable   true, enable; false, disable

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.48 "Display video queue frame" interface

The prototype: S32 (*vout_window_queue_frame) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, VOUT_FRAME_INFO_S *pstFrameInfo);

Function: Display video queue frame

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

pstFrameInfo   frame information

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.49 "Recover video display frame" interface

The prototype: S32 (*vout_window_dequeue_frame) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, VOUT_FRAME_INFO_S *pstFrameInfo, U32 u32TimeOut);

Function: Recover video display frame

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

pstFrameInfo   frame information

u32TimeOut   Timeout value, waiting for recoverable display frame within the timeout range

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.50 "Display window reset" interface

The prototype: S32 (*vout_window_reset) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, VOUT_WINDOW_SWITCH_MODE_E enWindowSwitchMode);

Function: Display window reset

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

enWindowSwitchMode   window switch mode

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.51 "Get virtual window width and height" interface

The prototype: S32 (*vout_window_get_virtual_size) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, U32 *const uVirtualScreenW, U32 *const uVirtualScreenH);

Function: Get virtual window width and height

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

Output parameters: uVirtualScreenW   virtual window width

uVirtualScreenH   virtual window height

Return: 0: correct; non-zero: error

E.6.52 "Get the playback display information" interface

The prototype: S32 (*vout_window_get_playinfo) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, U32 *const u32DelayTime, U32 *const u32DispRate, U32 *const u32FrameNumInBufQn);

Function: Get the playback display information

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

Output parameters: u32DelayTime   How long will the current latest frame be displayed;

u32DispRate   Display frame rate;

u32FrameNumInBufQn   How many frames are in the window queue

Return: 0: correct; non-zero: error

E.6.53 Window binding video source

The prototype: S32 (*vout_window_attach_input) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, const HANDLE hSource);

Function: Window binding video source

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

hSource   video source

Output parameters: None.

Return: 0: correct; non-zero: error

E.6.54 Window unbind video source

The prototype: S32 (*vout_window_detach_input) (struct _VOUT_DEVICE_S* pstVoutDev, const HANDLE hVoutWindow, const HANDLE hSource);

Function: Window unbind video source

Input parameter: pstVoutDev   Vout device handle

hVoutWindow   window handle

hSource   video source

Output parameters: None.

Return: 0: correct; non-zero: error

Annex F

AV module

(This annex forms an integral part of this Recommendation.)

This annex defines the hardware abstraction layer interface of the AV module. The definitions of the basic data types and operators are given in clause 6.

F.1 Constant definition

The definition of constants is shown in Table F.1.

Table F.1 – Constant definition

Constant	Description
const AV_HARDWARE_MODULE_ID = "audio_video"	Define AV module ID
const AV_HARDWARE_AV0 = "av0"	Define AV device ID

F.2 Enumeration definition

F.2.1 Enumeration definition of AV decoding status

Enumeration definition of AV decoding status is shown in Table F.2.

Table F.2 – Enumeration definition of AV decoding status (AV_DECODER_STATE_E)

Type name	Value	Description
AV_DECODER_STATE_RUNNING	0	Decoding
AV_DECODER_STATE_PAUSING	1	Pause decoding, real-time streaming cannot be paused
AV_DECODER_STATE_FREEZING	2	Decoding, no display, for real-time streaming you can use this instead of pause
AV_DECODER_STATE_STOPPED	3	Stop decoding
AV_DECODER_STATE_UNKNOWN	4	Illegal value
AV_DECODER_STATE_BUTT	5	Enumeration maximum value of AV decoding status

F.2.2 Enumeration definition of video decoding error recovery mode

Enumeration definition of video decoding error recovery mode is shown in Table F.3.

Table F.3 – Enumeration definition of video decoding error recovery mode (AV_ERROR_RECOVERY_MODE_E)

Type name	Value	Description
AV_ERROR_RECOVERY_MODE_NONE	0	No error correction, often used for debugging
AV_ERROR_RECOVERY_MODE_PARTIAL	1	Partial error correction
AV_ERROR_RECOVERY_MODE_HIGH	2	High error correction

Table F.3 – Enumeration definition of video decoding error recovery mode (AV_ERROR_RECOVERY_MODE_E)

Type name	Value	Description
AV_ERROR_RECOVERY_MODE_FULL	3	Full error correction, discard if there is an error, often used for debugging
AV_ERROR_RECOVERY_MODE_BUTT	4	Enumeration maximum value of video decoding error recovery mode

F.2.3 Enumeration definition of audio stream type

Enumeration definition of audio stream type is shown in Table F.4.

Table F.4 – Enumeration definition of Audio stream type (AV_AUD_STREAM_TYPE_E)

Type name	Value	Description
AV_AUD_STREAM_TYPE_MP2	0	MP2 format
AV_AUD_STREAM_TYPE_MP3	1	MP3 format
AV_AUD_STREAM_TYPE_AAC	2	AAC format
AV_AUD_STREAM_TYPE_AC3	3	AC3 format
AV_AUD_STREAM_TYPE_DTS	4	DTS format
AV_AUD_STREAM_TYPE_DTS_EXPRESS	5	DTS EXPRESS format
AV_AUD_STREAM_TYPE_VORBIS	6	VORBIS format
AV_AUD_STREAM_TYPE_DVAUDIO	7	DVAUDIO format
AV_AUD_STREAM_TYPE_WMAV1	8	WMAV1 format
AV_AUD_STREAM_TYPE_WMAV2	9	WMAV2 format
AV_AUD_STREAM_TYPE_MACE3	10	MACE3 format
AV_AUD_STREAM_TYPE_MACE6	11	MACE6 format
AV_AUD_STREAM_TYPE_VMDAUDIO	12	VMDAUDIO format
AV_AUD_STREAM_TYPE_SONIC	13	SONIC format
AV_AUD_STREAM_TYPE_SONIC_LS	14	SONIC LS format
AV_AUD_STREAM_TYPE_FLAC	15	FLAC format
AV_AUD_STREAM_TYPE_MP3ADU	16	MP3ADU format
AV_AUD_STREAM_TYPE_MP3ON4	17	MP3ON4 format
AV_AUD_STREAM_TYPE_SHORTEN	18	SHORTEN format
AV_AUD_STREAM_TYPE_ALAC	19	ALAC format
AV_AUD_STREAM_TYPE_WESTWOOD_SND1	20	WESTWOOD_SND1 format
AV_AUD_STREAM_TYPE_GSM	21	GSM format
AV_AUD_STREAM_TYPE_QDM2	22	QDM2 format
AV_AUD_STREAM_TYPE_COOK	23	COOK format
AV_AUD_STREAM_TYPE_TRUESPEECH	24	TRUESPEECH format
AV_AUD_STREAM_TYPE_TTA	25	TTA format

Table F.4 – Enumeration definition of Audio stream type (AV_AUD_STREAM_TYPE_E)

Type name	Value	Description
AV_AUD_STREAM_TYPE_SMACKAUDIO	26	SMACKAUDIO format
AV_AUD_STREAM_TYPE_QCELP	27	QCELP format
AV_AUD_STREAM_TYPE_WAVPACK	28	WAVPACK format
AV_AUD_STREAM_TYPE_DSICINAUDIO	29	DSICINAUDIO format
AV_AUD_STREAM_TYPE_IMC	30	IMC format
AV_AUD_STREAM_TYPE_MUSEPACK7	31	MUSEPACK7 format
AV_AUD_STREAM_TYPE_MLP	32	MLP format
AV_AUD_STREAM_TYPE_GSM_MS	33	GSM_MS format
AV_AUD_STREAM_TYPE_ATRAC3	34	ATRAC3 format
AV_AUD_STREAM_TYPE_VOXWARE	35	VOXWARE format
AV_AUD_STREAM_TYPE_APE	36	APE format
AV_AUD_STREAM_TYPE_NELLYMOSER	37	NELLYMOSER format
AV_AUD_STREAM_TYPE_MUSEPACK8	38	MUSEPACK8 format
AV_AUD_STREAM_TYPE_SPEEX	39	SPEEX format
AV_AUD_STREAM_TYPE_WMAVOICE	40	WMAVOICE format
AV_AUD_STREAM_TYPE_WMAPRO	41	WMAPRO format
AV_AUD_STREAM_TYPE_WMALOSSLESS	42	WMALOSSLESS format
AV_AUD_STREAM_TYPE_ATRAC3P	43	ATRAC3P format
AV_AUD_STREAM_TYPE_EAC3	44	EAC3 format
AV_AUD_STREAM_TYPE_SIPR	45	SIPR format
AV_AUD_STREAM_TYPE_MP1	46	MP1 format
AV_AUD_STREAM_TYPE_TWINVQ	47	TWINVQ format
AV_AUD_STREAM_TYPE_TRUEHD	48	TRUEHD format
AV_AUD_STREAM_TYPE_MP4ALS	49	MP4ALS format
AV_AUD_STREAM_TYPE_ATRAC1	50	ATRAC1 format
AV_AUD_STREAM_TYPE_BINKAUDIO_RDFT	51	BINKAUDIO_RDFT format
AV_AUD_STREAM_TYPE_BINKAUDIO_DCT	52	BINKAUDIO_DCT format
AV_AUD_STREAM_TYPE_DRA	53	DRA format
AV_AUD_STREAM_TYPE_PCM	54	PCM format
AV_AUD_STREAM_TYPE_PCM_BLURAY	55	PCM_BLURAY format
AV_AUD_STREAM_TYPE_ADPCM	56	ADPCM format
AV_AUD_STREAM_TYPE_AMR_NB	57	AMR_NB format
AV_AUD_STREAM_TYPE_AMR_WB	58	AMR_WB format
AV_AUD_STREAM_TYPE_AMR_AWB	59	AMR_AWB format
AV_AUD_STREAM_TYPE_RA_144	60	RA_144 format
AV_AUD_STREAM_TYPE_RA_288	61	RA_288 format
AV_AUD_STREAM_TYPE_DPCM	62	DPCM format
AV_AUD_STREAM_TYPE_G711	63	G711 format

Table F.4 – Enumeration definition of Audio stream type (AV_AUD_STREAM_TYPE_E)

Type name	Value	Description
AV_AUD_STREAM_TYPE_G722	64	G722 format
AV_AUD_STREAM_TYPE_G7231	65	G7231 format
AV_AUD_STREAM_TYPE_G726	66	G726 format
AV_AUD_STREAM_TYPE_G728	67	G728 format
AV_AUD_STREAM_TYPE_G729AB	68	G729AB format
AV_AUD_STREAM_TYPE_MULTI	69	MULTI format
AV_AUD_STREAM_TYPE_MS12_DDP	70	MS12_DDP format
AV_AUD_STREAM_TYPE_MS12_AAC	71	MS12_AAC format
AV_AUD_STREAM_TYPE_MS12_AC4	72	MS12_AC4 format
AV_AUD_STREAM_TYPE_BUTT	73	Enumeration maximum value of audio stream type

F.2.4 Enumeration definition of video stream type

Enumeration definition of video stream type is shown in Table F.5.

Table F.5 – Enumeration definition of video stream type (AV_VID_STREAM_TYPE_E)

Type name	Value	Description
AV_VID_STREAM_TYPE_MPEG2	0	MPEG2 format
AV_VID_STREAM_TYPE_MPEG4	1	MPEG4 format
AV_VID_STREAM_TYPE_AVS	2	AVS format
AV_VID_STREAM_TYPE_AVSPLUS	3	AVS+ format
AV_VID_STREAM_TYPE_H263	4	H263 format
AV_VID_STREAM_TYPE_H264	5	H264 format
AV_VID_STREAM_TYPE_REAL8	6	REAL8 format
AV_VID_STREAM_TYPE_REAL9	7	REAL9 format
AV_VID_STREAM_TYPE_VC1	8	VC1 format
AV_VID_STREAM_TYPE_VP6	9	VP6 format
AV_VID_STREAM_TYPE_VP6F	10	VP6F format
AV_VID_STREAM_TYPE_VP6A	11	VP6A format
AV_VID_STREAM_TYPE_MJPEG	12	MJPEG format
AV_VID_STREAM_TYPE_SORENSON	13	SORENSON format
AV_VID_STREAM_TYPE_DIVX3	14	DIVX3 format
AV_VID_STREAM_TYPE_RAW	15	RAW format
AV_VID_STREAM_TYPE_JPEG	16	JPEG format
AV_VID_STREAM_TYPE_VP8	17	VP8 format
AV_VID_STREAM_TYPE_VP9	18	VP9 format
AV_VID_STREAM_TYPE_MSMPEG4V1	19	MSMPEG4V1 format
AV_VID_STREAM_TYPE_MSMPEG4V2	20	MSMPEG4V2 format
AV_VID_STREAM_TYPE_MSVIDEO1	21	MSVIDEO1 format
AV_VID_STREAM_TYPE_WMV1	22	WMV1 format
AV_VID_STREAM_TYPE_WMV2	23	WMV2 format
AV_VID_STREAM_TYPE_RV10	24	RV10 format
AV_VID_STREAM_TYPE_RV20	25	RV20 format

Table F.5 – Enumeration definition of video stream type (AV_VID_STREAM_TYPE_E)

Type name	Value	Description
AV_VID_STREAM_TYPE_SVQ1	26	SVQ1 format
AV_VID_STREAM_TYPE_SVQ3	27	SVQ3 format
AV_VID_STREAM_TYPE_H261	28	H261 format
AV_VID_STREAM_TYPE_VP3	29	VP3 format
AV_VID_STREAM_TYPE_VP5	30	VP5 format
AV_VID_STREAM_TYPE_CINEPAK	31	CINEPAK format
AV_VID_STREAM_TYPE_INDEO2	32	INDEO2 format
AV_VID_STREAM_TYPE_INDEO3	33	INDEO3 format
AV_VID_STREAM_TYPE_INDEO4	34	INDEO4 format
AV_VID_STREAM_TYPE_INDEO5	35	INDEO5 format
AV_VID_STREAM_TYPE_MJPEGB	36	MJPEGB format
AV_VID_STREAM_TYPE_MVC	37	MVC format
AV_VID_STREAM_TYPE_HEVC	38	HEV format
AV_VID_STREAM_TYPE_DV	39	DV format
AV_VID_STREAM_TYPE_WMV3	40	WMV3 format
AV_VID_STREAM_TYPE_HUFFYUV	41	HUFFYUV format
AV_VID_STREAM_TYPE_REALMAGICMPEG4	42	REALMAGIC MPEG4 format
AV_VID_STREAM_TYPE_DIVX	43	DIVX format
AV_VID_STREAM_TYPE_BUTT	44	Enumeration maximum value of video stream type

F.2.5 Enumeration definition of AV event

Enumeration definition of AV event is shown in Table F.6.

Table F.6 – Enumeration definition of AV event (AV_EVT_E)

Type name	Value	Description
AV_EVT_BASE	0	Baseline value of audio and video events
AV_VID_EVT_BASE	0	Baseline value of video events
AV_VID_EVT_DECODE_START	0	Video decoding starts， the definition of returned data is shown in Table F.25 AV_VID_STATUS_S
AV_VID_EVT_DECODE_STOPPED	1	Video decoding stopped， the definition of returned data is shown in Table F.25 AV_VID_STATUS_S
AV_VID_EVT_NEW_PICTURE_DECODED	2	The new picture is decoded. The definition of returned data is shown in AV_VID_FRAMEINFO_S
AV_VID_EVT_DISCARD_FRAME	3	Drop a video frame
AV_VID_EVT_PTS_ERROR	4	Video PTS error

Table F.6 – Enumeration definition of AV event (AV_EVT_E)

Type name	Value	Description
AV_VID_EVT_UNDERFLOW	5	Video data underflow
AV_VID_EVT_ASPECT_RATIO_CHANGE	6	The video aspect ratio has changed， the definition of returned data is shown in Table F.25 AV_VID_STATUS_S
AV_VID_EVT_STREAM_FORMAT_CHANGE	7	The video streaming format has changed， the definition of returned data is shown in Table F.25 AV_VID_STATUS_S
AV_VID_EVT_OUT_OF_SYNC	8	Sync lost， the definition of returned data is shown in Table F.25 AV_VID_STATUS_S
AV_VID_EVT_BACK_TO_SYNC	9	Sync recovery， the definition of returned data is shown in Table F.25 AV_VID_STATUS_S
AV_VID_EVT_DATA_OVERFLOW	10	Video data overflow， the definition of returned data is shown in Table F.25 AV_VID_STATUS_S
AV_VID_EVT_DATA_UNDERFLOW	11	Video data underflow， the definition of returned data is shown in Table F.25 AV_VID_STATUS_S
AV_VID_EVT_PICTURE_DECODING_ERROR	12	Image decoding failed， the definition of returned data is shown in Table F.25 AV_VID_STATUS_S
AV_VID_EVT_CODEC_UNSUPPORT	13	Do not support the format decoding
AV_VID_EVT_BUTT	14	Enumeration maximum value of AV event
AV_AUD_EVT_BASE	14	Audio event basic value
AV_AUD_EVT_DECODE_START	14	Audio decoding starts， the definition of returned data is shown in Table F.24 AV_VID_STATUS_S
AV_AUD_EVT_DECODE_STOPPED	15	Audio decoding stopped, the definition of returned data is shown in Table F.24 AV_VID_STATUS_S
AV_AUD_EVT_NEW_FRAME	16	New audio frame decoding completed， the definition

Table F.6 – Enumeration definition of AV event (AV_EVT_E)

Type name	Value	Description
		of returned data is shown in Table F.24 AV_VID_STATUS_S
AV_AUD_EVT_DISCARD_FRAME	17	Drop an audio frame
AV_AUD_EVT_PTS_ERROR	18	Audio PTS error
AV_AUD_EVT_UNDERFLOW	19	Audio data underflow
AV_AUD_EVT_DECODING_ERROR	20	Audio decoding failed
AV_AUD_EVT_PCM_UNDERFLOW	21	PCM data underflow， the definition of returned data is shown in Table F.24 AV_VID_STATUS_S
AV_AUD_EVT_FIFO_OVERFLOW	22	Audio data overflow， the definition of returned data is shown in Table F.24 AV_VID_STATUS_S
AV_AUD_EVT_LOW_DATA_LEVEL	23	The definition of returned data is shown in Table F.24 AV_VID_STATUS_S
AV_AUD_EVT_OUT_OF_SYNC	24	Sync lost， the definition of returned data is shown in Table F.24 AV_VID_STATUS_S
AV_AUD_EVT_BACK_TO_SYNC	25	Sync recovery， the definition of returned data is shown in Table F.24 AV_VID_STATUS_S
AV_AUD_EVT_CODEC_UNSUPPORT	26	Unsupported decoding format
AV_AUD_EVT_BUTT	27	Enumeration maximum value of audio event
AV_INJECT_EVT_BASE	27	Injection event baseline value
AV_INJECT_EVT_DATA_UNDERFLOW	27	Injection data underflow, the definition of returned data is shown in Table F.20 AV_VID_STATUS_S
AV_INJECT_EVT_DATA_OVERFLOW	28	Injection data overflow, the definition of returned data is shown in Table F.204 AV_VID_STATUS_S
AV_INJECT_EVT_IMPOSSIBLE_WITH_MEM_PROFILE	29	The current memory configuration does not support， the definition of returned data is shown in Table F.20

Table F.6 – Enumeration definition of AV event (AV_EVT_E)

Type name	Value	Description
		AV_VID_STATUS_S
AV_STREAM_PLAY_EOS	30	Play end event
AV_EVT_BUTT	31	Enumeration maximum value of audio and video event

F.2.6 Enumeration definition of video stop mode

Enumeration definition of video stop mode is shown in Table F.7.

Table F.7 – Enumeration definition of video stop mode (AV_VID_STOP_MODE_E)

Type name	Value	Description
AV_VID_STOP_MODE_FREEZE	0	Freeze frame
AV_VID_STOP_MODE_BLACK	1	Black screen
AV_VID_STOP_MODE_BUTT	2	Enumeration maximum value of video stop mode

F.2.7 Enumeration definition of program data source type

Enumeration definition of program data source type is shown in Table F.8.

Table F.8 – Enumeration definition of program data source type (AV_SOURCE_TYPE_E)

Type name	Value	Description
AV_SOURCE_TUNER	0	Tuner Data comes from Tuner
AV_SOURCE_MEM	1	Data comes from memory

F.2.8 Enumeration definition of memory injection data type

Enumeration definition of memory injection data type is shown in Table F.9.

Table F.9 – Enumeration definition of memory injection data type (AV_DATA_TYPE_E)

Type name	Value	Description
AV_DATA_TYPE_NONE	0	Invalid value
AV_DATA_TYPE_TS	1	TS format data, injected through Demux
AV_DATA_TYPE_PES	2	PES format data
AV_DATA_TYPE_ES	4	ES format data
AV_DATA_TYPE_PCM	8	PCM format data
AV_DATA_TYPE_IFRAME	16	I format data

F.2.9 Enumeration definition of audio and video sync mode

Enumeration definition of audio and video sync mode is shown in Table F.10.

Table F.10 – Enumeration definition of audio and video sync mode (AV_SYNC_MODE_E)

Type name	Value	Description
AV_SYNC_MODE_DISABLE	0	Turn off sync
AV_SYNC_MODE_AUTO	1	Automatic processing of sync mode and sync parameters
AV_SYNC_MODE_PCR	2	Based on PCR
AV_SYNC_MODE_VID	3	Video PTS is the sync mode based on the clock
AV_SYNC_MODE_AUD	4	Audio PTS as a clock reference sync mode
AV_SYNC_MODE_SCR	5	System clock reference sync mode
AV_SYNC_MODE_BUTT	6	Enumeration maximum value of audio and video sync mode

F.2.10 Enumeration definition of Types of memory injected data content

Enumeration definition of Types of memory injected data content is shown in Table F.11.

Table F.11 – Enumeration definition of Types of memory injected data content (AV_CONTENT_TYPE_E)

Type name	Value	Description
AV_CONTENT_DEFAULT	0	Default type, such as TS
AV_CONTENT_AUDIO	1	Audio data
AV_CONTENT_VIDEO	2	Video data
AV_CONTENT_BUTT	3	Enumeration maximum value of Types of memory injected data content

F.2.11 Enumeration definition of 3D stream source format

Enumeration definition of 3D stream source format is shown in Table F.12.

Table F.12 – Enumeration definition of 3D stream source format (AV_3D_FORMAT_E)

Type name	Value	Description
AV_3D_FORMAT_OFF	0	
AV_3D_FORMAT_SIDE_BY_SIDE	1	Side by side, left and right half
AV_3D_FORMAT_TOP_AND_BOTTOM	2	Up and down mode
AV_3D_FORMAT_SIDE_BY_SIDE_FULL	3	Side by side, left and right audience
AV_3D_FORMAT_FRAME_PACKING	4	Frame packing
AV_3D_FORMAT_FIELD_ALTERNATIVE	5	Field interleaving
AV_3D_FORMAT_LINE_ALTERNATIVE	6	Line interleaving
AV_3D_FORMAT_AUTO	7	Automatic mode, this mode requires both driver and stream support
AV_3D_FORMAT_BUTT	8	Enumeration value of 3D stream source format

F.2.12 Enumeration definition of video rotation angle

Enumeration definition of video rotation angle is shown in Table F.13.

Table F.13 – Enumeration definition of video rotation angle (AV_VID_ROTATION_E)

Type name	Value	Description
AV_VID_ROTATION_0	0	Not rotating
AV_VID_ROTATION_90	1	90 degree rotation
AV_VID_ROTATION_180	2	180 degree rotation
AV_VID_ROTATION_270	3	270 degree rotation
AV_VID_ROTATION_BUTT	4	Enumeration maximum value of video rotation angle

F.2.13 Enumeration definition of data input stream interface type

Enumeration definition of data input stream interface type is shown in Table F.14.

Table F.14 – Enumeration definition of data input stream interface type (AV_STREAM_TYPE_E)

Type name	Value	Description
AV_STREAM_TYPE_TS	0	TS stream
AV_STREAM_TYPE_ES	1	ES stream
AV_STREAM_TYPE_BUTT	2	Enumeration maximum value of data input stream interface type

F.2.14 Enumeration definition decoding ability of decoder (resolution)

Enumeration definition decoding ability of decoder (resolution) is shown in Table F.15.

Table F.15 – Enumeration definition decoding ability of decoder (resolution) (VDEC_RESO_LEVEL_E)

Type name	Value	Description
VDEC_RESO_LEVEL_QCIF	0	QCIF decoding
VDEC_RESO_LEVEL_CIF	1	CIF decoding
VDEC_RESO_LEVEL_D1	2	D1 decoding
VDEC_RESO_LEVEL_720P	3	720p resolution
VDEC_RESO_LEVEL_FULLHD	4	FULL HD resolution
VDEC_RESO_LEVEL_1280x800	5	1280x800 resolution
VDEC_RESO_LEVEL_800x1280	6	800x1280 resolution
VDEC_RESO_LEVEL_1488x1280	7	1488x1280 resolution
VDEC_RESO_LEVEL_1280x1488	8	1280x1488 resolution
VDEC_RESO_LEVEL_2160x1280	9	2160x1280 resolution
VDEC_RESO_LEVEL_1280x2160	10	1280x2160 resolution
VDEC_RESO_LEVEL_2160x2160	11	2160x2160 resolution
VDEC_RESO_LEVEL_3840x2160	12	3840x2160 resolution
VDEC_RESO_LEVEL_4096x2160	13	4096x2160 resolution

Table F.15 – Enumeration definition decoding ability of decoder (resolution) (VDEC_RESO_LEVEL_E)

Type name	Value	Description
VDEC_RESO_LEVEL_2160x4096	14	2160x4096 resolution
VDEC_RESO_LEVEL_4096x4096	15	4096x4096 resolution
VDEC_RESO_LEVEL_8192x4096	16	8192x4096 resolution
VDEC_RESO_LEVEL_4096x8192	17	4096x8192 resolution
VDEC_RESO_LEVEL_8192x8192	18	8192x8192 resolution
VDEC_RESO_LEVEL_BUTT	19	Enumeration maximum value of decoding ability of decoder (resolution)

F.3 Definition of data structure

F.3.1 Sync zone parameter setting

Sync zone parameter setting is shown in Table F.16.

Table F.16 – Sync zone parameter setting (AV_SYNC_REGION_PARAMS_S)

Attribute name	Type	Description
s32VidPlusTime	S32	Video advance sync benchmark time
s32VidNegativeTime	S32	Video is behind the sync benchmark
s32AudPlusTime	S32	Audio advance sync reference time
s32AudNegativeTime	S32	Audio is behind the sync reference time
bSmoothPlay	BOOL	TRUE: Sync slowly, repeat every few frames or discard a frame, you may see slow motion in the effect, FALSE: Sync quickly, you may see a stutter in the effect

F.3.2 AV sync parameter setting

AV sync parameter setting is shown in Table F.17.

Table F.17 – AV sync parameter setting (AV_SYNC_PARAM_S)

Attribute name	Type	Description
stSyncStartRegion	Structure	The definition of AV_SYNC_REGION_PARAMS_S is shoun in Table F.16, indicating the start of synchronization adjustment area
stSyncNovelRegion	Structure	The definition of AV_SYNC_REGION_PARAMS_S is shoun in Table F.16, indicates the start of synchronization abnormal adjustment area
u32PreSyncTimeoutMs	U32	When starting to play, the pre-sync timeout time, if it is 0, it means no pre-sync
bPreSyncSmoothPlay	BOOL	When starting playback, the synchronization method before synchronization or within the pre-sync timeout period is reached
bQuickOutput	BOOL	Whether to output the first frame quickly when starting playback

F.3.3 Data injection setting parameter structure

Data injection setting parameter structure is shown in Table F.18.

Table F.18 – Data injection setting parameter structure (AV_INJECT_SETTINGS_S)

Attribute name	Type	Description
enDataType	Enumerated type	Type of data injected. The definition of AV_DATA_TYPE_E is shown in Table F.9
enInjectContent	Enumerated type	The injected data format. The definition of AV_CONTENT_TYPE_E is shown in Table F.11
u32BufSize	U32	Cache size, when the value is zero, it means that the HAL layer decides on its own
u32InjectMinLen	U32	The minimum length of the injected data, when the value is 0, it means that the HAL layer decides on its own

F.3.4 Memory status information structure

Memory status information structure is shown in Table F.19.

Table F.19 – Memory status information structure (AV_BUF_STATUS_S)

Attribute name	Type	Description
enDataType	Enumerated type	Type of data injected. The definition of AV_DATA_TYPE_E is shown in Table F.9
u32Size	U32	Cache size
u32Free	U32	Free cache size
u32Used	U32	Used cache size
u32InjectMinLen	U32	The minimum length of each injection data

F.3.5 Inject state structure

Inject state structure is shown in Table F.20.

Table F.20 – Inject state structure (AV_INJECT_STATUS_S)

Attribute name	Type	Description
hInjecter	HANDLE	Injector handle
hAv	HANDLE	Player handle
stBufStatus	Structure	Memory status. the definition of structure for values is shown in Table F.19 AV_BUF_STATUS_S
enSourceId	U32	When TS is injected, the corresponding TS stream channel

F.3.6 Injector opening parameters

Injector opening parameters is shown in Table F.21.

Table F.21 – Injector opening parameters (AV_INJECTER_OPEN_PARAMS_S)

Attribute name	Type	Description
stSettings	Structure	Injector setting parameters. The definition for values is shown in Table F.18 AV_INJECT_SETTINGS_S

F.3.7 Injector closing parameters

Injector closing parameters is shown in Table F.22.

Table F.22 – Injector closing parameters (AV_INJECTER_CLOSE_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserve for future extension

F.3.8 Program source parameters

Program source parameters is shown in Table F.23.

Table F.23 – Program source parameters (AV_SOURCE_PARAMS_S)

Attribute name	Type	Description
enDemuxId	Enumerated value	DemuxID， the definition for values is shown in Table B.4 DMX_ID_E

F.3.9 State structure of audio, current stream, decoder, DAC

State structure of audio, current stream, decoder, DAC is shown in Table F.24.

Table F.24 – State structure of audio, current stream, decoder, DAC (AV_AUD_STATUS_S)

Attribute name	Type	Description
u32PacketCount	U32	Confirm whether there is audio by checking whether there is an audio package
u32FrameCount	U32	Determine if there is audio by checking the number of audio frames
enDecodeState	Enumerated value	Audio decoding status， the definition for values is shown in Table F.2 AV_DECODER_STATE_E
enStreamType	Enumerated value	The stream type obtained from the stream, the definition for values is shown in Table F.4 AV_AUD_STREAM_TYPE_E
u32SampleRate	U32	Audio sampling rate (32000,44100,48000)
u32BitWidth	U32	The number of bits occupied by each sampling point of the audio, such as 8 bits, 16 bits
enSourceType	Enumerated value	Data Sources， the definition for values is shown in Table F.8 AV_SOURCE_TYPE_E
u32PesBufferSize	U32	PES buffer size
u32PesBufferFreeSize	U32	Free PES buffer size
u32EsBufferSize	U32	ES buffer size
u32EsBufferFreeSize	U32	Free ES buffer size
s64Pts	S64	PTC used by the current tone decoder
s64FirstPts	S64	The first PTS obtained by the decoder
u32Stc	U32	STC used by the current audio decoder
u16Pid	U32	Audio PID
u16ADPid	U32	Audio PID
u32FrameBufTime	U32	Frame butter time

F.3.10 Video, current stream, decoder, DAC state structure

Video, current stream, decoder, DAC state structure is shown in Table F.25.

Table F.25 – Video, current stream, decoder, DAC state structure (AV_VID_STATUS_S)

Attribute name	Type	Description
u32DispPicCount	U32	Determine if there is any video by checking the number of displayed video frames
enDecodeState	Enumerated value	Video decoder status， the definition for values is shown in Table F.2 AV_DECODER_STATE_E
enStreamType	Enumerated value	Video type. The definition for values is shown in Table F.5 AV_VID_STREAM_TYPE_E
u16FpsInteger	U16	Integer part of frame rate
u16FpsDecimal	U16	Fractional part of frame rate
enSourceType	Enumerated value	Data Sources， the definition for values is shown in Table F.8 AV_SOURCE_TYPE_E
en3dFormat	Enumerated value	3D TV program source information. The definition for values is shown in Table F.8 AV_SOURCE_TYPE_E
u32PesBufferSize	U32	PES buffer size
u32PesBufferFreeSize	U32	Free PES buffer size
u32EsBufferSize	U32	ES buffer size
u32EsBufferFreeSize	U32	Free ES buffer size
s64Pts	S64	PTC used by the current tone decoder
s64FirstPts	S64	The first PTS obtained by the decoder
u32Stc	U32	STC used by the current audio decoder
bInterlaced	BOOL	Progressive or interlaced
u32SourceWidth	U32	Video source width
u32SouceHeight	U32	Video source height
u32DisplayWidth	U32	Video display width
u32DisplayHeight	U32	Video display height
bByPass	BOOL	True means transparent transmission
u16Pid	U16	Video PID

F.3.11 Play state structure

Play state structure is shown in Table F.26.

Table F.26 – Play state structure (AV_STATUS_S)

Attribute name	Type	Description
stAudStatus	Structure	Audio playback status. The definition for values is shown in Table.24 AV_AUD_STATUS_S
stVidStatus	Structure	Video playback status. The definition for values is shown in Table.25 AV_VID_STATUS_S
u32TsBufferSize	U32	TS buffer size
u32TsBufferFreeSize	U32	Free TS buffer size
u32TsPacketSize	U32	TS packet size
ahInjecter [AV_CONTENT_BUTT]	HANDLE	The injector bound to the player, with: AV_CONTENT_E as the subscript index, and 0 as the illegal handle

Table F.26 – Play state structure (AV_STATUS_S)

Attribute name	Type	Description
s64LocalTime	S64	Local synchronization reference time, -1 is an invalid value
u16PcrPid	U16	PCR PID
s32AVDiffTime	S32	Audio and video synchronization time difference, in milliseconds

F.3.12 Audio decoder parameter structure

Audio decoder parameter structure is shown in Table F.27.

Table F.27 – Audio decoder parameter structure (AV_ADEC_PARAM_S)

Attribute name	Type	Description
u32Version	U32	Audio encoding version, only wma encoding, 0x160 (WMAV1), 0x161 (WMAV2)
u32SampleRate	U32	Audio sampling rate (32000, 44100, 48000), 0 means not concerned
u32BitWidth	U32	The number of bits occupied by each sampling point of the audio, such as: 8 bits, 16 bits, 0 means not concerned
u32Channels	U32	Number of channels: 0,1,2,4,6,8,10... ,0 means not concerned
u32BlockAlign	U32	Packet size, 0 means not concerned
u32Bps	U32	Audio bit rate (bit/s), 0 means not concerned
bBigEndian	BOOL	Large and small, only PCM format is valid
u32ExtradataSize	U32	Extended data length
pu8Extradata	pointer	The address of the extended data, pointer type U8
pCodecContext	pointer	Audio decoding context, pointer type VOID

F.3.13 Video decoder parameter structure

Video decoder parameter structure is shown in Table F.28.

Table F.28 – Video decoder parameter structure (AV_VDEC_PARAM_S)

Attribute name	Type	Description
u32Fps	U32	Frame rate (x1000). The actual frame rate is multiplied by 1000,0 means not concerned
u32Bps	U32	Video bit rate (bit/s),0 means not concerned
u16Width	U16	Width, in pixels,0 means not concerned
u16Height	U16	Height, in pixels,0 means not concerned
u32Profile	U32	Profile level
u32Version	U32	Decoder version
bFlip	BOOL	Set to 1 when the image needs to be inverted, otherwise set to 0
pCodecContext	pointer	Video decoding context, pointer type is void

F.3.14 AV setting parameter structure

AV setting parameter structure is shown in Table F.29.

Table F.29 – AV setting parameter structure (AV_SETTINGS_S)

Attribute name	Type	Description
stAdecParams	Structure	Audio decoding parameters. The definition for values is shown in Table F.27 AV_ADEC_PARAM_S
stVdecParams	Structure	Video decoding parameters. The definition for values is shown in Table F.27 AV_ADEC_PARAM_S
stSourceParams	Structure	Data Sources. The definition for values is shown in Table F.23 AV_SOURCE_PARAMS_S
enAvSyncMode	Enumerated value	Sync mode. The definition for values is shown in Table
stSyncParams	Structure	Sync parameters. The definition for values is shown in Table F.17 AV_SYNC_PARAM_SDd
enErrRecoveryMode	Enumerated value	Error recovery mode. The definition for values is shown in Table F.3 AV_ERROR_RECOVERY_MODE_E
enVidStopMode	Enumerated value	Video stop mode. The definition for values is shown in Table F.7 AV_VID_STOP_MODE_E
enVidStreamType	Enumerated value	Video stream type. The definition for values is shown in Table F.5 AV_VID_STREAM_TYPE_E
enAudStreamType	Enumerated value	Audio stream type. The definition for values is shown in Table F.4 AV_AUD_STREAM_TYPE_E
en3dFormat	Enumerated value	3D TV settings. The definition for values is shown in Table F.12 AV_3D_FORMAT_E
bVidDecodeOnce	BOOL	FALSE: normal setting, decode all input data; TRUE: decode only the first picture, usually used for preview
s32Speed	S32	100: normal play, 200: double speed, -200: fast backward 2 times speed, 50: 1/2 times speed slow forward, 0 means free play, etc. Mainly configured to identify fast forward and fast rewind
u16PcrPid	U16	PID corresponding to PCR
u16AudPid	U16	Audio corresponding PID
u16AudADPid	U16	PID corresponding to audio AD
u16VidPid	U16	PID corresponding to the video
s32AudSyncOffseMs	S32	Set the time offset of the audio relative to the synchronization reference
s32VidSyncOffseMs	S32	Set the time offset of the video relative to the synchronization reference
bEos	BOOL	End sign
enVideoRotation	Enumerated value	Video rotation angle

F.3.15 I frame parameter structure

I frame parameter structure is shown in Table F.30.

Table F.30 – I frame parameter structure (AV_IFRAME_PARAMS_S)

Attribute name	Type	Description
u32DataLength	U32	Data length
pvIframeData	pointer	void Data address, pointer type void

F.3.16 I frame decoding parameter structure

I frame decoding parameter structure is shown in Table F.31.

Table F.31 – I frame decoding parameter structure (AV_IFRAME_DECODE_PARAMS_S)

Attribute name	Type	Description
stIframeParams	Structure	I frame data information. The definition of structure is shown in Table F.30 AV_IFRAME_PARAMS_S

F.3.17 Play event configuration parameter structure

Play event configuration parameter structure is shown in Table F.32.

Table F.32 – Play event configuration parameter structure (AV_EVT_CONFIG_PARAMS_S)

Attribute name	Type	Description
enEvt	Enumerated value	AV event. The definition for values is shown in Table F.6 AV_EVT_E
pfnCallback	Callback function pointer	When this parameter is empty, it means cancel the callback function registered before
bEnableCallback	BOOL	Indicates whether to enable the callback
u32NotificationToSkip	U32	Indicates that the event needs to be skipped several times before calling the registered callback function

F.3.18 AV module initialization parameter structure

AV module initialization parameter structure is shown in Table F.33

Table F.33 – AV module initialization parameter structure (AV_INIT_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved parameters

F.3.19 AV player instance creation parameter structure

AV player instance creation parameter structure is shown in Table F.34.

Table F.34 – AV player instance creation parameter structure (AV_INIT_PARAMS_S)

Attribute name	Type	Description
stSourceParams	Structure	Input source parameters. The definition of structure is shown in Table F.23 AV_SOURCE_PARAMS_S
enStreamType	Enumerated value	Stream type. The definition of enumerated value is shown in Table F.14 AV_STREAM_TYPE_E

F.3.20 Close the player instance parameter structure

Close the player instance parameter structure is shown in Table F.35.

Table F.35 – Close the player instance parameter structure (AV_DESTROY_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved parameters

F.3.21 AV module termination parameter structure

AV module termination parameter structure is shown in Table F.36.

Table F.36 – AV module termination parameter structure (AV_TERM_PARAMS_S)

Attribute name	Type	Description
u32Dummy	U32	Reserved parameters

F.3.22 AV module termination parameter structure

AV module termination parameter structure is shown in Table F.37.

Table F.37 – AV module termination parameter structure (AV_PCM_PARAMS_S)

Attribute name	Type	Description
u32SampleRate	U32	Audio sampling rate (32000, 44100, 48000), 0 means not concerned
u32BitWidth	U32	The number of bits occupied by each sampling point of the audio, such as 8 bit, 16 bit, 0 means not concerned
u32AudChannel	U32	Number of channels: 0,1,2,4,6,8,10... ,0 means not concerned
bBigEndian	BOOL	TRUE: big endian; FALSE: little endian

F.3.23 Structure of decoder capability

Structure of decoder capability is shown in Table F.38.

Table F.38 – Structure of decoder capability (AV_VDEC_CAPABILITY_S)

Attribute name	Type	Description
bSupportedDecType	BOOL	Whether the type of enDecCapLevel is supported
enDecCapLevel	Enumerated value	Decoding ability, the definition of enumerated value is shown in Table F.15 VDEC_RESO_LEVEL_E
u32Number	U32	Maximum number of video decoding
u32Fps	U32	Frame rate (x1000), actual frame rate multiplied by 1000
u32Dummy	U32	Reserved parameters

F.3.24 The structure of the player's decoder capability

The structure of the player's decoder capability is shown in Table F.39.

Table F.39 – The structure of the player's decoder capability (AV_DECODER_CAPABILITY_S)

Attribute name	Type	Description
enInjectDataType	Enumerated value	Type of data injected. The definition of enumerated value is shown in Table F.9 AV_DATA_TYPE_E
au32AudDecode [AV_AUD_STREAM_TYPE_BUTT]	U32	Non-zero means supported, 0 means not supported
au32AudBypass [AV_AUD_STREAM_TYPE_BUTT]	U32	Non-zero means supported, 0 means not supported
stVidDecoder [AV_VID_STREAM_TYPE_BUTT]	Structure	Video decoding capability

F.3.25 Structure of player capabilities

Structure of player capabilities is shown in Table F.40.

Table F.40 – Structure of player capabilities (AV_CAPABILITY_S)

Attribute name	Type	Description
stDecoderCapability	Structure	Player decoding capability

F.3.26 Structure of ES data parameters

Structure of ES data parameters is shown in Table F.41.

Table F.41 – Structure of ES data parameters (AV_ES_PARAMS_S)

Attribute name	Type	Description
u32SampleRate	U32	Audio sampling rate (32000, 44100, 48000), 0 means not concerned

F.3.27 AV ES data parameter structure

AV ES data parameter structure is shown in Table F.42.

Table F.42 – AV ES data parameter structure (AV_ES_DATA_S)

Attribute name	Type	Description
s64TimeStamp	S64	Time stamp
pvHeader	pointer	ES data header, if there is no data header, it can be set to NULL, pointer type is VOID
u32HeaderLen	U32	ES data header length, if not set to 0
pvEsBuf	pointer	Data buffer address, pointer type is VOID
u32EsLen	U32	Buffer length
pvPrivate	pointer	Private data buffer address, pointer type is VOID
u32PrivateLen	U32	Private data buffer length

F.3.28 AV module structure

AV module structure is shown in Table F.43.

Table F.43 – AV module structure (AV_MODULE_S)

Attribute name	Type	Description
stCommon	Structure	The definition of structure is shown in Table B.41 hw_module_t

F.4 The definition of Callback function

F.4.1 AV event callback function

The prototype: typedef S32 (* AV_CALLBACK_PFN_T) (const HANDLE hAvHandle, const AV_EVT_E enEvt, const VOID* pvData);

Function: av event callback function type declaration.

Input parameter: hAvHandle AV module handle

enEvt AV event type

pvData AV data the description is shown in Table F.6 AV_EVT_E

Output parameters： None.

Return: 0: correct; non-zero: error

F.5 Call method

The call method of the hardware abstract interface of the AV module is shown in Figure F.1.

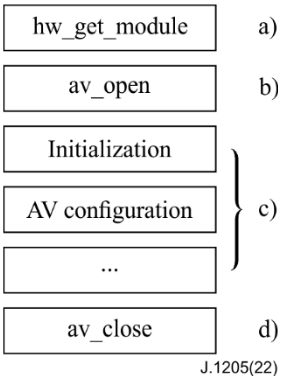

Figure F.1 – AV module hardware abstract interface calling method

Figure F.1:

a) Call the hw_get_module() interface to get the HAL Stub of the AV module: hw_get_module(AV_HARDWARE_MODULE_ID, &g_av_module).

b) Call av_open (g_av_module,&pstDevice) to get the device handle of the AV module: av_open (g_av_module,&pstDevice).

c) Control the AV hardware through a series of interface functions provided by the device handle.

d) After completing the hardware manipulation, call the av_close () interface to close the AV device to avoid resource leakage.

F.6 Definition of interface

F.6.1 "Open the AV device" interface

The prototype: static inline S32 av_open (const struct hw_module_t* pstModule, AV_DEVICE_S** ppstDevice)；

Function: Open the AV module device.

Input parameter: pstModule AV module handle

Return: 0: correct; non-zero: error

F.6.2 "Close the AV module" device.

The prototype: static inline S32 av_close (AV_DEVICE_S* pstDevice)

Function: Close an AV module device

Input parameter: pstDevice AV device handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.3 "AV initialization" interface

The prototype: S32 (*av_init) (struct _AV_DEVICE_S *pstDev, const AV_INIT_PARAMS_S * const pstInitParams);

Function: AV module initialization

Input parameter: pstDev AV device handle

pstInitParams initialization module parameters

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.4 "Create AV player instance" interface

The prototype: S32 (*av_create) (struct _AV_DEVICE_S *pstDev, HANDLE * const phAv, const AV_CREATE_PARAMS_S * const pstCreateParams);

Function: Create a player instance

Input parameter: pstDev AV device handle

pstCreateParams Create the parameters of the player

Output parameters: phAv AV player handle

Return: 0: correct; non-zero: error

F.6.5 "Delete the AV playback instance" interface

The prototype: S32 (*av_destroy) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle, const AV_DESTROY_PARAMS_S * const pstDestroyParams);

Function: delete a player instance

Input parameter: pstDev AV device handle

hAvHandle AV player handle

pstDestroyParams Deletes the parameters of the playback instance, currently reserved.

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.6 "Terminal AV playback instance" interface

The prototype: S32 (*av_term) (struct _AV_DEVICE_S *pstDev, const AV_TERM_PARAMS_S * const pstTermParams);

Function: Terminate a player instance.

nput parameter: pstDev AV device handle

pstTermParams Termination parameter, currently reserved.

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.7 "Get AV playback capability" interface

The prototype: S32 (*av_get_capability) (struct _AV_DEVICE_S *pstDev, AV_CAPABILITY_S *pstCapability);

Function: Obtain the ability of AV player.

Input parameter: pstDev AV device handle

Output parameters：pstCapability Player capability handle

Return: 0: correct; non-zero: error

F.6.8 "Configure AV playback event" interface

The prototype: S32 (*av_evt_config) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle, const AV_EVT_CONFIG_PARAMS_S * const pstCfg);

Function: Configure the parameters of an AV event

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

pstCfg event configuration parameter.

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.9 "Get AV playback event" interface

The prototype: S32 (*av_get_evt_config) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle, const AV_EVT_E enEvt, AV_EVT_CONFIG_PARAMS_S * const pstCfg);

Function: Get the parameters of an AV event

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

pstCfg event configuration parameter.

Output parameters：enEvt AV event parameter.

Return: 0: correct; non-zero: error

F.6.10 "Get AV playback status" interface

The prototype: S32 (*av_get_status) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle, AV_STATUS_S * pstStatus);

Function: Get AV playback status

Input parameter: pstDev   AV device handle

hAvHandle   AV player instance handle

Output parameters：pstStatus   AV playback status

Return: 0: correct; non-zero: error

F.6.11 "Get AV settings" interface

The prototype: S32 (*av_get_config) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle, AV_SETTINGS_S * pstSettings);

Function: Get AV settings

Input parameter: pstDev   AV device handle

hAvHandle   AV player instance handle

Output parameters：pstSettings   AV settings parameters

Return: 0: correct; non-zero: error

F.6.12 "Set AV parameter" interface

The prototype: S32 (*av_set_config) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle, AV_SETTINGS_S * const pstSettings);

Function: Set AV parameters

Input parameter: pstDev   AV device handle

hAvHandle   AV player instance handle

pstSettings   AV settings parameters

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.13 "Start AV playback" interface

The prototype: S32 (*av_start) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle);

Function: Start AV decoding.

Input parameter: pstDev   AV device handle

hAvHandle   AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.14 "Stop AV playback" interface

The prototype: S32 (*av_stop) (struct _AV_DEVICE_S* pstDev, const HANDLE hAvHandle);

Function: Stop AV decoding

Input parameter: pstDev   AV device handle

hAvHandle   AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.15 "Pause AV playback" interface

The prototype: S32 (*av_pause) (struct _AV_DEVICE_S* pstDev, const HANDLE hAvHandle);

Function: Pause audio and video decoding, real-time streaming does not support this operation.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.16 "Restore AV playback" interface

The prototype: S32 (*av_resume) (struct _AV_DEVICE_S* pstDev, const HANDLE hAvHandle);

Function: Restore audio and video decoding, real-time streaming does not support this operation.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.17 "Reset AV playback" interface

The prototype: S32 (*av_reset) (struct _AV_DEVICE_S* pstDev, const HANDLE hAvHandle, S64 s64TimeMs);

Function: reset audio and video buffer to a certain point in time

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

s64TimeMs reset time point, in milliseconds

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.18 "Start video decoding" interface

The prototype: S32 (*av_start_video) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle);

Function: Start video decoding.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.19 "Pause video decoding" interface

The prototype: S32 (*av_pause_video) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle);

Function: Pause video decoding.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.20 "Freeze video decoding" interface

The prototype: S32 (*av_freeze_video) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle, BOOL bFreeze);

Function: Video decoding continues, but the display is not updated.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

bFreeze TRUE: video freeze; FALSE: video unfreeze

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.21 "Restore video decoding" interface

The prototype: S32 (*av_resume_video) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle);

Function: restore video decoding and display.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.22 "Stop video decoding" interface

The prototype: S32 (*av_stop_video) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle);

Function: Stop video decoding.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.23 "Clear cached video" interface

The prototype: S32 (*av_clear_video) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle)

Function: Clear the data in the video display cache, and also clear the display on the screen.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.24 "Start audio decoding" interface

The prototype: S32 (*av_start_audio) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle);

Function: start audio decoding

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.25 "Pause audio decoding" interface

The prototype: S32 (*av_pause_audio) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle);

Function: Pause audio decoding

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.26 "Restore audio decoding" interface

The prototype: S32 (*av_resume_audio) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle);

Function: restore audio decoding.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.27 "Stop audio decoding" interface

The prototype: S32 (*av_stop_audio) (struct _AV_DEVICE_S *pstDev, const HANDLE hAvHandle);

Function: Stop audio decoding.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.28 "Decoding I frame" interface

The prototype: S32 (*av_decode_iframe) (struct _AV_DEVICE_S* pstDev, const HANDLE hAvHandle, const AV_IFRAME_DECODE_PARAMS_S* const pstIframeDacodeParams);

Function: decode an I frame, whether to update the video display depends on the current video display settings, the I frame of the video layer can be displayed by calling the av_enable_video function; av_disable_video is hidden.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

pstIframeDacodeParams I frame data waiting to be decoded

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.29 "Release I frame resource" interface

The prototype: S32 (*av_release_iframe) (struct _AV_DEVICE_S* pstDev, const HANDLE hAvHandle)

Function: Release the resources for decoding an I frame.

Input parameter: pstDev AV device handle

hAvHandle AV player instance handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.30 "Open the AV injector" interface

The prototype: S32 (*av_injecter_open) (struct _AV_DEVICE_S *pstDev, HANDLE * const phInjecter, AV_INJECTER_OPEN_PARAMS_S * const pstOpenParams);

Function: open an injector instance

Input parameter: pstDev AV device handle

pstOpenParams Parameters when opening the injector

Output parameters: phInjecter AV injector handle

Return: 0: correct; non-zero: error

F.6.31 "Close the AV injector" interface

The prototype: S32 (*av_injecter_close) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter, AV_INJECTER_CLOSE_PARAMS_S * const pstCloseParams);

Function: Close an injector instance.

Input parameter: pstDev AV device handle

phInjecter AV injector handle

pstCloseParams Parameters when opening the injector, currently reserved

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.32 "Associate injector and decoder" interface

The prototype: S32 (*av_injecter_attach) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter, const HANDLE hAvHandle);

Function: Associate an injector to the decoder.

Input parameter: pstDev AV device handle

hInjecter Injector handle

hAvHandle AV player handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.33 "Cancel the association between the injector and the decoder"interface

The prototype: S32 (*av_injecter_detach) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter);

Function: cancel the injector association.

Input parameter: pstDev AV device handle

hInjecter Injector handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.34 "Inject data" interface

The prototype: S32 (*av_inject_data) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter, VOID * const pvData, const U32 u32Length, U32 u32TimeoutMs);

Function: Inject the data to be decoded.

Input parameter: pstDev AV device handle

hInjecter Injector handle

pvData Data to be decoded

u32Length Data length

u32TimeoutMs Overtime

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.35 "Get the number of free memory of the injector" interface

The prototype: S32 (*av_inject_get_freebuf) (struct _AV_DEVICE_S* pstDev, const HANDLE hInjecter, U32 u32ReqLen, VOID** ppvBufFree, U32* pu32FreeSize);

Function: Get the free buffer address pointer in the next injection buffer and the size of the continuous free buffer, fill in the data and push the data to the decoder through the av_inject_write_complete function.

Input parameter: pstDev AV device handle

hInjecter Injector handle

u32ReqLen expected data length

Output parameters：ppvBufFree Return the address pointer of the free space;

pu32FreeSize Return space length

Return: 0: correct; non-zero: error

F.6.36 "Complete data injection" interface

The prototype: S32 (*av_inject_write_complete) (struct _AV_DEVICE_S* pstDev, const HANDLE hInjecter, U32 u32WriteSize, S64 s64Pts);

Function: Call av_inject_get_freebuf to get the free buffer and copy the data, then call this function to send to the decoder.

Input parameter: pstDev AV device handle

hInjecter Injector handle

u32WriteSize The length of the data to be written;

s64Pts The PTS of the data.

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.37 "Get the cache status of the memory injection method" interface

The prototype: S32 (*av_inject_get_buf_status) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter, AV_BUF_STATUS_S * const pstBufStatus);

Function: Get the cache status of the memory injection method.

Input parameter: pstDev AV device handle

hInjecter Injector handle

pstBufStatus cache status

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.38 "Get the injector setting parameter" interface

The prototype: S32 (*av_inject_get_setting) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter, AV_INJECT_SETTINGS_S * const pstSettings);

Function: Get the injector setting parameters.

Input parameter: pstDev AV device handle

hInjecter Injector handle

Output parameters : pstSettings Injector parameters

Return: 0: correct; non-zero: error

F.6.39 "Reset injector buffer" interface

The prototype: S32 (*av_inject_reset_buf) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter);

Function: reset injection cache

Input parameter: pstDev AV device handle

hInjecter Injector handle

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.40 "Set PCM parameter" interface

The prototype: S32 (*av_inject_set_pcm_params) (struct _AV_DEVICE_S* pstDev, const HANDLE hInjecter, AV_PCM_PARAMS_S* const pstParams);

Function: Set PCM parameters.

Input parameter: pstDev AV device handle

hInjecter Injector handle

pstParams PCM parameters

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.41 "Set ES parameter" interface

The prototype: S32 (*av_inject_set_es_params) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter, AV_ES_PARAMS_S * const pstParams);

Function: Set ES parameters.

Input parameter: pstDev AV device handle

hInjecter Injector handle

pstParams Parameters

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.42 "Injecting ES data in memory mode" interface

The prototype: S32 (*av_inject_es_data) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter, AV_ES_DATA_S * const pstPesData, U32 u32TimeoutMs);

Function: inject ES data in memory mode

Input parameter: pstDev AV device handle

hInjecter Injector handle

pstPesData ES data pointer

u32TimeoutMs Timeout, in milliseconds

Output parameters: None.

Return: 0: correct; non-zero: error

F.6.43 "Terminate data injection" interface

The prototype: S32 (*av_inject_abort) (struct _AV_DEVICE_S *pstDev, const HANDLE hInjecter)

Function: Terminate data injection.

Input parameter: pstDev AV device handle

hInjecter Injector handle

Output parameters: None.

Return: 0: correct; non-zero: error

Bibliography

[b-ITU-T J.205] Recommendation ITU-T J.205 (2012), *Requirements for an application control framework using integrated broadcast and broadband digital television*.

[b-ITU-T J.1203] Recommendation ITU-T J.1203 (2022), *Smart television operating system – Specification*.

[b-ITU-T J.1204] Recommendation ITU-T J.1204 (2022), *Smart television operating system – Security framework*.

[b-OpenGL ES 2.0.25] Khronos Group (2010), OpenGL ES Common Profile Specification, Version 2.0.25.
https://www.khronos.org/registry/OpenGL/specs/es/2.0/es_full_spec_2.0.pdf

[b-OpenMAX IL 1.1.2] Khronos Group (2008), *OpenMAX Integration Layer Application Programming Interface Specification*, version 1.1.2.
https://registry.khronos.org/OpenMAX-IL/specs/OpenMAX_IL_1_1_2_Specification.pdf

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	**Cable networks and transmission of television, sound programme and other multimedia signals**
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2023