Audio Components

API Reference

Issue **00B03**

Date **2017-09-08**

HiSilicon Technologies Co., Ltd.

Address: New R&D Center, Wuhe Road,
Bantian, Longgang District,
Shenzhen 518129 P. R. China

Website: http://www.hisilicon.com

Email: support@hisilicon.com

About This Document

Purpose

This document provides reference information including the protocol description, application programming interfaces (APIs), and error codes for the programmers that develop intelligent analysis solutions using the audio module of HiSilicon media processors.

Related Versions

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

Product Name	Version
Hi3559A	V100ES
Hi3536D	V100
Hi3559A	V100
Hi3559C	V100

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
 DANGER	Alerts you to a high risk hazard that could, if not avoided, result in serious injury or death.

Symbol	Description
WARNING	Alerts you to a medium or low risk hazard that could, if not avoided, result in moderate or minor injury.
CAUTION	Alerts you to a potentially hazardous situation that could, if not avoided, result in equipment damage, data loss, performance deterioration, or unanticipated results.
TIP	Provides a tip that may help you solve a problem or save time.
NOTE	Provides additional information to emphasize or supplement important points in the main text.

Change History

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

Issue 00B03 (2017-09-08)

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

The description of the Hi3536D V100 is added.

In section 1.3, the **Requirement** and **Note** fields of HI_MPI_AENC_AacInit and HI_MPI_ADEC_AacInit are updated.

Issue 00B02 (2017-05-27)

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

In section 1.1, a caution part is added.

Issue 00B01 (2017-04-10)

This issue is the first draft release.

Contents

Tables

1 Audio Components

1.1 Introduction

The audio components integrate the advanced audio coding (AAC) protocol. The audio component interfaces are open, which facilitates integration of third-party encoding/decoding protocols by users. The sample code for AAC encoding and decoding is stored in the **sample/audio** directory.

CAUTION

If you need to use AAC patents, you must obtain authorization from the owner of copyright and pay licensing fees.

1.2 Important Concepts

- Audio encoding/decoding protocols

  The audio encoding/decoding functions of the SDK are implemented based on independent AAC libraries. The core codec performs encoding or decoding using the CPU software in user mode.

  Table 1-1 describes the AAC protocol.

Table 1-1 Audio encoding/encoding protocols

Protocol	Sampling Rate (kHz)	Frame Length (Sampling Points)	Bit Rate (kbit/s)	Compression Ratio	CPU Usage (MHz)	Description
AAC encoder	8, 16, 22.05, 24, 32, 44.1, or 48	The frame length supported by the AAC-LC is 1024	None	None	50	The AAC has two breakthroughs in technology evolution. • aacPlus1 (or eAAC): The spectral

Protocol	Sampling Rate (kHz)	Frame Length (Sampling Points)	Bit Rate (kbit/s)	Compression Ratio	CPU Usage (MHz)	Description
		sampling points, the frame length supported by the eAAC and eAACPlus is 2048 sampling points, and the frame length supported by the AACLD and AACELD is 512 sampling points.				bandwidth replication (SBR) technology is used so that the codec can implement the same voice quality as the bit rate is half of the original bit rate. • aacPlus2 (or eAACPlus): The parameter stereo (PS) technology is used so that the preferred voice quality can be achieved although the bit rate is low. By using the aacPlus2, the voice quality like CD is achieved at the bit rate of 48 kbit/s. • AAC-LD and AAC-ELD are low delay sound encoding/decoding processing solutions. AAC-LD is the standard requirement for the security industry and AAC-ELD is the future encoding format used in communication. For details about the stream range and the recommended bit rates, see Table 1-2 and Table 1-3.
AAC decoder	Compatible with all the sampling rates	512, 1024, 2048	None	None	25	Backward compatible. The traditional AAC decoder decodes only the low-frequency information of the aac Plus v1 stream, whereas the aacPlus decoder can restore the high-frequency information additionally. The AAC decoder not supporting PS generates only the

Protocol	Sampling Rate (kHz)	Frame Length (Sampling Points)	Bit Rate (kbit/s)	Compression Ratio	CPU Usage (MHz)	Description
						voice of a single channel when decoding the aac Plus v2 stream, the aacPlus decoder can generate the stereo voice. Note that the decoding method must be ADEC_MODE_STREAM.

NOTE

The CPU usage value is obtained based on the 288 MHz ARM9. 2/2 MHz indicates that CPU usage for encoding and decoding is 2 MHz, respectively.

Table 1-2 Bit rates supported by the AAC encoder in various protocols (kbit/s)

Sampling Rate (kHz)	Audio Channel	LC BitRate		Plus v1 BitRate		Plus v2 BitRate	
		Supported	Preferred	Supported	Preferred	Supported	Preferred
8 kHz	Mono	16–48	24	–	–	–	–
	Stereo	16–96	32	–	–	–	–
16 kHz	Mono	24–96	48	2448	32	–	–
	Stereo	24–192	48	24–96	32	16–48	32
22.05 kHz	Mono	32–132	64	32–64	48	–	–
	Stereo	32–265	48	32–128	64	16–64	32
24 kHz	Mono	32–144	48	32–64	48	–	–
	Stereo	32–288	48	32–128	64	16–64	32
32 kHz	Mono	32–192	48	32–64	48	–	–
	Stereo	32–320	128	32–128	64	16–64	32
44.1 kHz	Mono	48–265	64	32–64	48	–	–
	Stereo	48–320	128	32–128	64	16–64	48
48 kHz	Mono	48–288	64	32–64	48	–	–
	Stereo	48–320	128	32–128	64	16–64	48

Note: "–" indicates that the bite rate is not supported.

Table 1-3 Bit rates supported by the AAC encoder in the low delay protocol (kbit/s)

Sampling Rate (kHz)	Audio Channel	LD BitRate		ELD BitRate	
		Supported	Preferred	Supported	Preferred
8 kHz	Mono	16–96	24	32–96	32
	Stereo	16–192	48	64–192	64
16 kHz	Mono	24–192	48	16–256	48
	Stereo	32–320	96	32–320	96
22.05 kHz	Mono	32–256	48	24–256	48
	Stereo	48–320	96	32–320	96
24 kHz	Mono	32–256	64	24–256	64
	Stereo	48–320	128	32–320	128
32 kHz	Mono	48–320	64	32–320	64
	Stereo	64–320	128	64–320	128
44.1 kHz	Mono	64–320	128	96–320	128
	Stereo	44–320	256	192–320	256
48 kHz	Mono	64–320	128	96–320	128
	Stereo	64–320	256	192–320	256

- Audio encoding/decoding integration interfaces

  Open interfaces in the SDK are used to register or deregister codecs. The audio components provide the samples for registering the AAC codec. You can register third-party codecs based on the samples, or register and use the AAC codec in the audio components based on the samples.

1.3 API Reference

The following APIs in the SDK release package are used to register or deregister codecs:

- HI_MPI_AENC_RegisterEncoder: Registers an encoder.
- HI_MPI_AENC_UnRegisterEncoder: Deregisters an encoder.
- HI_MPI_ADEC_RegisterDecoder: Registers a decoder.
- HI_MPI_ADEC_UnRegisterDecoder: Deregisters a decoder.

The registration samples provided in the audio component are as follows:

- HI_MPI_AENC_AacInit: Registers an AAC encoder.
- HI_MPI_ADEC_AacInit: Registers an AAC decoder.

HI_MPI_AENC_RegisterEncoder

[Description]

Registers an encoder.

[Syntax]

```
HI_S32 HI_MPI_AENC_RegisterEncoder(HI_S32 *ps32Handle, AENC_ENCODER_S
*pstEncoder);
```

[Parameter]

Parameter	Description	Input/Output
ps32Handle	Registration handle	Output
pstEncoder	Structure defining encoder attributes	Input

[Return Value]

Return Value	Description
0	Success.
Other values	Failure. The value is an error code. For details, see section 1.5 "Error Code."

[Requirement]

- Header file: hi_comm_aenc.h and mpi_aenc.h
- Library file: libmpi.a

[Note]

- You can register an encoder with the AENC module by transferring a desired encoder attribute structure to the module, and the registration handle is returned. You can deregister this encoder by using the registration handle.
- You can register a maximum of 20 encoders (including the five encoders LPCM, G711a, G711u, G726, and ADPCM registered with the AENC module) with the AENC module.
- An encoding protocol can be used to register only one encoder of the same type. For example, you are not allowed to register another AAC encoder if you have registered one.
- Encoder attributes include encoder type, maximum stream length, encoder name, function pointer for starting an encoder, encoding function pointer, and function pointer for closing an encoder.
  - Encoder type

    Encoding protocols are marked in enumeration form in the SDK. You can select an encoder type to register a specified encoder based on protocols.
  - Maximum stream length
  - The maximum stream length after each frame is encoded. The AENC module allocates the memory space based on the registered maximum stream length.

- Encoder name

  The encoder name is expressed in character strings and is displayed in the proc information.

- Function pointer for starting an encoder

  This is a function pointer in the SDK. The following is its prototype:

  HI_S32 (*pfnOpenEncoder)(HI_VOID *pEncoderAttr, HI_VOID **ppEncoder);

  The first parameter specifies encoder attributes. The second parameter is the pointer to an encoder handle, and this pointer is used to return a handle for operating the encoder. The preceding parameters are packaged by users. Take the memory allocation into account when packaging the second parameter because the encoder handle is also used for encoding and closing an encoder.

- Encoding function pointer

  This is a function pointer in the SDK. The following is its prototype:

  HI_S32 (*pfnEncodeFrm)(HI_VOID *pEncoder, const AUDIO_FRAME_S *pstData,

  HI_U8 *pu8Outbuf,HI_U32 *pu32OutLen);

  The parameter is the encoder handle that is returned when the previous function starts an encoder. The second parameter is the pointer to the audio frame data structure in the SDK. The third parameter is the pointer to the output buffer. The fourth parameter specifies the output buffer length.

- Function pointer for closing an encoder

  This is a function pointer in the SDK. The following is its prototype:

  HI_S32 (*pfnCloseEncoder)(HI_VOID *pEncoder);

  This parameter is an encoder handle that is returned when an encoder is started.

- Users must encapsulate a third-party encoder based on the preceding function prototypes and register this encoder with the AENC module by using the encoder attribute structure, implementing integration of a third-party encoder.

- Register an encoder of a specified type before creating encoding channels. Encoders do not need to be repeatedly registered.

[Example]

The following code describes how to register an AAC encoder:

```
HI_S32 s32Handle, s32Ret;
    AENC_ENCODER_S stAac;

    stAac.enType = PT_AAC;
    snprintf(stAac.aszName, sizeof(stAac.aszName), "Aac");
    stAac.u32MaxFrmLen = MAX_AAC_MAINBUF_SIZE;
    stAac.pfnOpenEncoder = OpenAACEncoder;
    stAac.pfnEncodeFrm = EncodeAACFrm;
    stAac.pfnCloseEncoder = CloseAACEncoder;
    s32Ret = HI_MPI_AENC_RegisterEncoder(&s32Handle, &stAac);
    if (s32Ret)
    {
        return s32Ret;
    }
```

```
    return HI_SUCCESS;
```

[See Also]

None

HI_MPI_AENC_UnRegisterEncoder

[Description]

Deregisters an encoder.

[Syntax]

```
HI_S32 HI_MPI_AENC_UnRegisterEncoder(HI_S32 s32Handle);
```

[Parameter]

Parameter	Description	Input/Output
s32Handle	Registration handle returned when an encoder is registered	Input

[Return Value]

Return Value	Description
0	Success.
Other values	Failure. The value is an error code. For details, see section 1.5 "Error Code."

[Requirement]

- Header file: hi_comm_aenc.h and mpi_aenc.h
- Library file: libmpi.a

[Note]

Typically, encoders do not need to be deregistered.

[Example]

None

[See Also]

None

HI_MPI_ADEC_RegisterDecoder

[Description]

Registers a decoder.

[Syntax]

```
HI_S32 HI_MPI_ADEC_RegisterDecoder(HI_S32 *ps32Handle, ADEC_DECODER_S
*pstDecoder);
```

[Parameter]

Parameter	Description	Input/Output
ps32Handle	Registration handle	Output
pstDecoder	Structure defining decoder attributes	Input

[Return Value]

Return Value	Description
0	Success.
Other values	Failure. The value is an error code. For details, see section 1.5 "Error Code."

[Requirement]

- Header file: hi_comm_adec.h and mpi_adec.h
- Library file: libmpi.a

[Note]

- You can register a decoder with the ADEC module by transferring a desired decoder attribute structure to the module, and the registration handle is returned. You can deregister this decoder by using the returned registration handle.
- You can register a maximum of 20 decoders (including the five decoders LPCM, G711a, G711u, G726, and ADPCM registered with the ADEC module) with the ADEC module.
- A decoding protocol can be used to register only one decoder of the same type. For example, you are not allowed to register another AAC decoder if you have registered one.
- Decoder attributes include decoder type, decoder name, function pointer for starting a decoder, decoding function pointer, function pointer for obtaining audio frame information, and function pointer for closing a decoder.
  - Decoder type

    Decoding protocols are marked in enumeration form in the SDK. You can select a decoder type to register a specified decoder based on protocols.
  - Decoder name

    The decoder name is expressed in character strings and is displayed in the proc information.
  - Function pointer for starting a decoder

    This is a function pointer in the SDK. The following is its prototype:

    HI_S32 (*pfnOpenDecoder)(HI_VOID *pDecoderAttr, HI_VOID **ppDecoder);

    The first parameter specifies decoder attributes. The second parameter is the pointer to a decoder handle, and this pointer is used to return a handle for operating the

decoder. The preceding parameters are packaged by users. You should take the memory allocation into account when encapsulating the second parameter because the decoder handle is also used for decoding and closing a decoder.

- Decoding function pointer

  This is a function pointer in the SDK. The following is its prototype:

  HI_S32 (*pfnDecodeFrm)(HI_VOID *pDecoder, HI_U8 **pu8Inbuf,

  HI_S32 *ps32LeftByte, HI_U16 *pu16Outbuf,

  HI_U32 *pu32OutLen,HI_U32 *pu32Chns);

  The first parameter is the decoder handle that is returned when the previous function starts a decoder. The second parameter is the input buffer that is used to send audio frame data. The third parameter is used to return the number of remaining bytes for streaming decoding (the sent audio frame data is not a complete frame). The fourth parameter is the output buffer. The fifth parameter is the mono channel length of output data. The sixth parameter is the number of output channels. After being decoded, stream data may be output in mono or stereo sound mode.

- Function pointer for obtaining audio frame information

  This is a function pointer in the SDK. The following is its prototype:

  HI_S32 (*pfnGetFrmInfo)(HI_VOID *pDecoder, HI_VOID *pInfo);

  The first parameter is the decoder handle that is returned when a decoder is started. The second parameter is audio frame information packaged by users. Some decoders can obtain the sampling point and sampling rate of audio data after the data is decoded. This function prototype can be packaged as an empty function if this function is not needed.

- Function pointer for closing a decoder

  This is a function pointer in the SDK. The following is its prototype:

  HI_S32 (*pfnCloseDecoder)(HI_VOID *pDecoder);

  This parameter is a decoder handle that is returned when a decoder is started.

- Users must encapsulate a third-party decoder based on the preceding function prototypes and register this decoder with the ADEC module by using the decoder attribute structure, implementing integration of a third-party decoder.

- You should register a decoder of a specified type before creating decoding channels. Decoders do not need to be repeatedly registered.

[Example]

The following code describes how to register an AAC decoder:

```
HI_S32 s32Handle, s32Ret;
   ADEC_DECODER_S stAac;

   stAac.enType = PT_AAC;
   snprintf(stAac.aszName, sizeof(stAac.aszName), "Aac");
   stAac.pfnOpenDecoder = OpenAACDecoder;
   stAac.pfnDecodeFrm = DecodeAACFrm;
   stAac.pfnGetFrmInfo = GetAACFrmInfo;
   stAac.pfnCloseDecoder = CloseAACDecoder;
   stAac.pfnResetDecoder = ResetAACDecoder;
   s32Ret = HI_MPI_ADEC_RegisterDecoder(&s32Handle, &stAac);
```

```
if (s32Ret)
{
    return s32Ret;
}

return HI_SUCCESS;
```

[See Also]

None

HI_MPI_ADEC_UnRegisterDecoder

[Description]

Deregisters a decoder.

[Syntax]

```
HI_S32 HI_MPI_ADEC_UnRegisterDecoder(HI_S32 s32Handle);
```

[Parameter]

Parameter	Description	Input/Output
s32Handle	Registration handle returned when a decoder is registered	Input

[Return Value]

Return Value	Description
0	Success.
Other values	Failure. The value is an error code. For details, see section 1.5 "Error Code."

[Requirement]

- Header files: hi_comm_adec.h and mpi_adec.h
- Library files: libmpi.a

[Note]

Typically, decoders do not need to be deregistered.

[Example]

None

[See Also]

None

HI_MPI_AENC_AacInit

[Description]

Registers an AAC encoder.

[Syntax]

```
HI_S32 HI_MPI_AENC_AacInit(HI_VOID);
```

[Parameter]

None

[Return Value]

Return Value	Description
0	Success.
Other values	Failure. The value is an error code. For details, see section 1.5 "Error Code."

[Requirement]

- Source file: audio_aac_adp.c
- Header file: audio_aac_adp.h
- Library file: libaacenc.a

[Note]

This interface is implemented in **audio_aac_adp.c**. But **audio_aac_adp.c** is not encapsulated into a library. When this interface is used, the compilation succeeds only when **audio_aac_adp.c** and **audio_aac_adp.h** are included. These two files are placed in the **sample/audio/adp** folder by default.

[Example]

None

[See Also]

None

HI_MPI_ADEC_AacInit

[Description]

Registers an AAC decoder.

[Syntax]

```
HI_S32 HI_MPI_ADEC_AacInit(HI_VOID);
```

[Parameter]

None

[Return Value]

Return Value	Description
0	Success.
Other values	Failure. The value is an error code. For details, see section 1.5 "Error Code."

[Requirement]

- Source file: aduio_aac_adp.h
- Header file: audio_aac_adp.h
- Library file: libaacdec.a

[Note]

For details, see the **Note** field of HI_MPI_AENC_AacInit.

[Example]

None

[See Also]

None

1.4 Data Structures

The following are the data structures of audio components:

- AENC_ENCODER_S: Defines encoder attributes.
- ADEC_DECODER_S: Defines decoder attributes.
- AAC_TYPE_E: Defines the type of the AAC encoding/decoding protocol.
- AAC_BPS_E: Defines the AAC encoding rate.
- AAC_TRANS_TYPE_E: Defines the transmission package type of the AAC audio encoding/decoding protocol.
- AENC_ATTR_AAC_S: Defines the attributes of the AAC encoding protocol.
- ADEC_ATTR_AAC_S: Defines the attributes of the AAC decoding protocol.

AENC_ENCODER_S

[Description]

Defines encoder attributes.

[Syntax]

```
typedef struct hiAENC_ENCODER_S
{
    PAYLOAD_TYPE_E  enType;
```

```
    HI_U32          u32MaxFrmLen;
HI_CHAR         aszName[16];
    HI_S32          (*pfnOpenEncoder)(HI_VOID *pEncoderAttr, HI_VOID
**ppEncoder);
    HI_S32          (*pfnEncodeFrm)(HI_VOID *pEncoder, const AUDIO_FRAME_S
*pstData, HI_U8 *pu8Outbuf,HI_U32 *pu32OutLen);
    HI_S32          (*pfnCloseEncoder)(HI_VOID *pEncoder);
} AENC_ENCODER_S;
```

[Member]

Member	Description
enType	Type of an encoding protocol. For details, see chapter 2 "System Control" in the HiMPP Media Processing Software Development Reference.
u32MaxFrmLen	Maximum stream length
aszName	Encoder name
pfnOpenEncoder	Function pointer for starting an encoder
pfnEncodeFrm	Encoding function pointer
pfnCloseEncoder	Function pointer for closing an encoder

[Note]

None

[See Also]

HI_MPI_AENC_RegisterEncoder

ADEC_DECODER_S

[Description]

Defines decoder attributes.

[Syntax]

```
typedef struct hiADEC_DECODER_S
{
    PAYLOAD_TYPE_E  enType;
    HI_CHAR         aszName[16];
    HI_S32          (*pfnOpenDecoder)(HI_VOID *pDecoderAttr, HI_VOID
**ppDecoder);
    HI_S32          (*pfnDecodeFrm)(HI_VOID *pDecoder, HI_U8
**pu8Inbuf,HI_S32 *ps32LeftByte, HI_U16 *pu16Outbuf,HI_U32
*pu32OutLen,HI_U32 *pu32Chns);
    HI_S32          (*pfnGetFrmInfo)(HI_VOID *pDecoder, HI_VOID *pInfo);
```

```
    HI_S32        (*pfnCloseDecoder)(HI_VOID *pDecoder);
} ADEC_DECODER_S;
```

[Member]

Member	Description
enType	Type of a decoding protocol. For details, see chapter 2 "System Control" in the HiMPP Media Processing Software Development Reference.
aszName	Decoder name
pfnOpenDecoder	Function pointer for starting a decoder
pfnDecodeFrm	Decoding function pointer
pfnGetFrmInfo	Function pointer for obtaining audio frame information
pfnCloseDecoder	Function pointer for closing a decoder

[Note]

None

[See Also]

HI_MPI_ADEC_RegisterDecoder

AAC_TYPE_E

[Description]

Defines the type of the AAC encoding/decoding protocol.

[Syntax]

```
typedef enum hiAAC_TYPE_E
{
    AAC_TYPE_AACLC     = 0,
    AAC_TYPE_EAAC      = 1,
    AAC_TYPE_EAACPLUS   = 2,
    AAC_TYPE_AACLD     = 3,
    AAC_TYPE_AACELD    = 4,
    AAC_TYPE_BUTT,
}AAC_TYPE_E;
```

[Member]

Member	Description
AAC_TYPE_AACLC	AAC-LC
AAC_TYPE_EAAC	eAAC format (also known as HEAAC, AAC+, or aacPlusV1)

Member	Description
AAC_TYPE_EAACPLUS	eAACPlus format (also known as AAC++ or aacPlusV2)
AAC_TYPE_AACLD	AACLD format
AAC_TYPE_AAECLD	AACELD format

[Note]

None

[See Also]

None

AAC_BPS_E

[Description]

Defines the AAC encoding rate.

[Syntax]

```
typedef enum hiAAC_BPS_E
{
    AAC_BPS_8K     = 8000,
    AAC_BPS_16K    = 16000,
    AAC_BPS_22K    = 22000,
    AAC_BPS_24K    = 24000,
    AAC_BPS_32K    = 32000,
    AAC_BPS_48K    = 48000,
    AAC_BPS_64K    = 64000,
    AAC_BPS_96K    = 96000,
    AAC_BPS_128K   = 128000,
    AAC_BPS_256K   = 256000,
    AAC_BPS_320K   = 320000,
    AAC_BPS_BUTT
}AAC_BPS_E;
```

[Member]

Member	Description
AAC_BPS_8K	8 kbit/s
AAC_BPS_16K	16 kbit/s
AAC_BPS_22K	22 kbit/s
AAC_BPS_24K	24 kbit/s
AAC_BPS_32K	32 kbit/s

Member	Description
AAC_BPS_48K	48 kbit/s
AAC_BPS_64K	64 kbit/s
AAC_BPS_96K	96 kbit/s
AAC_BPS_128K	128 kbit/s
AAC_BPS_256K	256 kbit/s
AAC_BPS_320K	320 kbit/s

[Note]

None

[See Also]

None

AAC_TRANS_TYPE_E

[Description]

Defines the transmission package type of the AAC audio encoding/decoding protocol.

[Syntax]

```
typedef enum hiAAC_TRANS_TYPE_E
{
    AAC_TRANS_TYPE_ADTS = 0,
    AAC_TRANS_TYPE_LOAS= 1,
    AAC_TRANS_TYPE_LATM_MCP1 = 2,
    AAC_TRANS_TYPE_BUTT
}AAC_TRANS_TYPE_E;
```

[Member]

Member	Description
AAC_TRANS_TYPE_ADTS	ADTS package type, which is supported by AACLC, EAAC, and EAACPLUS
AAC_TRANS_TYPE_LOAS	LOAS package type, which is supported by AACLC, EAAC, EAACPLUS, AACLD, and AACELD
AAC_TRANS_TYPE_LATM_MCP1	LATM1 package type, which is supported by AACLC, EAAC, EAACPLUS, AACLD, and AACELD

[Note]

Due to the lack of frame header synchronization mechanism, the LATM1 format cannot recover quickly when exceptions occur in the stream. Therefore, the LATM1 format is not recommended.

[See Also]

None

AENC_ATTR_AAC_S

[Description]

Defines the attributes of the AAC encoding protocol.

[Syntax]

```
typedef struct hiAENC_ATTR_AAC_S
{
    AAC_TYPE_E          enAACType;
    AAC_BPS_E           enBitRate;
    AUDIO_SAMPLE_RATE_E enSmpRate;
    AUDIO_BIT_WIDTH_E   enBitWidth;
    AUDIO_SOUND_MODE_E  enSoundMode;
    AAC_TRANS_TYPE_E    enTransType;
    HI_S16            s16BandWidth;
}AENC_ATTR_AAC_S;
```

[Member]

Member	Description
enAACType	AAC encoding type
enBitRate	Encoding bit rate. Value range: LC: 16–320; eAAC: 24–128; eAAC+: 16–64; AACLD: 16–320; AACELD: 32–320; It is measured in kbit/s.
enSmpRate	Sampling rate of the audio data. Value range: LC: 8–48; eAAC: 16–48; eAAC+: 16–48; AACLD: 8–48; AACELD: 8–48; It is measured in kHz.

Member	Description
enBitWidth	Bit width of the audio data. Only 16-bit data width is supported.
enSoundMode	Sound mode of the input data. Both the mono and stereo sound modes are supported.
enTransType	AAC transmission package type Value range: • AAC_TRANS_TYPE_ADTS: 0 • AAC_TRANS_TYPE_LOAS: 1 • AAC_TRANS_TYPE_LATM_MCP1: 2
s16BandWidth	Target frequency band range Value range: 0 or 1000–enSmpRate/2 Unit: Hz

[Note]

None

[See Also]

None

ADEC_ATTR_AAC_S

[Description]

Defines the attributes of the AAC decoding protocol.

[Syntax]

```
typedef struct hiADEC_ATTR_AAC_S
{
    AAC_TRANS_TYPE_E enTransType;
}ADEC_ATTR_AAC_S;
```

[Member]

Member	Description
enTransType	AAC transmission package type Value range: • AAC_TRANS_TYPE_ADTS: 0 • AAC_TRANS_TYPE_LOAS: 1 • AAC_TRANS_TYPE_LATM_MCP1: 2

[Note]

None

[See Also]

None

1.5 Error Code

Error Code for AENC APIs

Table 1-4 describes the error code for AENC APIs.

Table 1-4 Error code for AENC APIs

Error Code	Macro Definition	Description
0xA0178001	HI_ERR_AENC_INVALID_DEVID	The ID of the AENC device is invalid.
0xA0178002	HI_ERR_AENC_INVALID_CHNID	The ID of the AENC channel is invalid.
0xA0178003	HI_ERR_AENC_ILLEGAL_PARAM	The AENC parameter settings are invalid.
0xA0178004	HI_ERR_AENC_EXIST	The AENC channel exists.
0xA0178005	HI_ERR_AENC_UNEXIST	The AENC channel is not created.
0xA0178006	HI_ERR_AENC_NULL_PTR	The pointer of the input parameter is null.
0xA0178007	HI_ERR_AENC_NOT_CONFIG	The AENC channel attributes are not configured.
0xA0178008	HI_ERR_AENC_NOT_SUPPORT	This operation is not supported.
0xA0178009	HI_ERR_AENC_NOT_PERM	The operation is forbidden.
0xA017800C	HI_ERR_AENC_NOMEM	The memory is insufficient.
0xA017800D	HI_ERR_AENC_NOBUF	Failed to allocate the buffer of the AENC channel.
0xA017800E	HI_ERR_AENC_BUF_EMPTY	The buffer of the AENC channel is empty.
0xA017800F	HI_ERR_AENC_BUF_FULL	The buffer of the AENC channel is full.
0xA0178010	HI_ERR_AENC_SYS_NOTREADY	The system is not initialized.
0xA0178040	HI_ERR_AENC_ENCODER_ERR	Data errors occur during AENC encoding.

Error Code for ADEC APIs

Table 1-5 describes the error code for ADEC APIs.

Table 1-5 Error code for ADEC APIs

Error Code	Macro Definition	Description
0xA0188001	HI_ERR_ADEC_INVALID_DEVID	The ID of the ADEC device is invalid.
0xA0188002	HI_ERR_ADEC_INVALID_CHNID	The ID of the ADEC channel is invalid.
0xA0188003	HI_ERR_ADEC_ILLEGAL_PARAM	The ADEC parameter settings are invalid.
0xA0188004	HI_ERR_ADEC_EXIST	The ADEC channel exists.
0xA0188005	HI_ERR_ADEC_UNEXIST	The ADEC channel is not created.
0xA0188006	HI_ERR_ADEC_NULL_PTR	The pointer of the input parameter is null.
0xA0188007	HI_ERR_ADEC_NOT_CONFIG	The ADEC channel attributes are not configured.
0xA0188008	HI_ERR_ADEC_NOT_SUPPORT	This operation is not supported.
0xA0188009	HI_ERR_ADEC_NOT_PERM	The operation is forbidden.
0xA018800C	HI_ERR_ADEC_NOMEM	The memory is insufficient.
0xA018800D	HI_ERR_ADEC_NOBUF	Failed to allocate the buffer of the ADEC channel.
0xA018800E	HI_ERR_ADEC_BUF_EMPTY	The buffer of the ADEC channel is empty.
0xA018800F	HI_ERR_ADEC_BUF_FULL	The buffer of the ADEC channel is full.
0xA0188010	HI_ERR_ADEC_SYS_NOTREADY	The system is not initialized.
0xA0188040	HI_ERR_ADEC_DECODER_ERR	Data errors occur during ADEC decoding.