



**音频组件**

**API 参考**

文档版本    01

发布日期    2019-07-25

版权所有 © 上海海思技术有限公司 2020。保留一切权利。

未经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 商标声明



**HISILICON**、海思和其他海思商标均为海思技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

## 注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

## 上海海思技术有限公司

地址：                    深圳市龙岗区坂田华为总部办公楼                    邮编：518129

网址：                    <http://www.hisilicon.com/cn/>

客户服务邮箱：          [support@hisilicon.com](mailto:support@hisilicon.com)



# 前 言

## 概述

本文档为使用上海海思媒体处理芯片的音频进行智能分析方案开发的程序员而写，目的是供您在开发过程中查阅音频支持的各种参考信息，包括各项协议说明、API、错误码等。

## 产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3559A	V100ES
Hi3536D	V100
Hi3536E	V100
Hi3559A	V100
Hi3559C	V100
Hi3519A	V100
Hi3556A	V100
Hi3516C	V500
Hi3516D	V300
Hi3516A	V300
Hi3559	V200
Hi3556	V200
Hi3562	V100



产品名称	产品版本
Hi3566	V100
Hi3568	V100
Hi3569	V100
Hi3516E	V200
Hi3516E	V300
Hi3518E	V300
Hi3516D	V200






## 读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

## 符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 <b>危险</b>	表示如不可避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 <b>警告</b>	表示如不可避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 <b>注意</b>	表示如不可避免则可能导致轻微或中度伤害的具有低等级风险的危害。
 <b>须知</b>	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 <b>说明</b>	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。



## 修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	版本	修改描述
2019-07-25	01	第 1 次正式版本发布 1.2 小节的【注意】涉及修改
2019-04-15	00B07	第 7 次临时版本发布 1.2 小节涉及修改
2019-03-30	00B06	第 6 次临时版本发布 增加 Hi3516DV200 内容。
2019-03-05	00B05	第 5 次临时版本发布 1.2 小节涉及修改
2018-10-15	00B04	第 4 次临时版本发布 1.2 小节涉及修改 1.3 小节, HI_MPI_AENC_AacInit 和 HI_MPI_ADEC_AacInit 涉及修改
2017-09-08	00B03	第 3 次临时版本发布 增加 Hi3536DV100 内容。 1.3 小节, HI_MPI_AENC_AacInit 和 HI_MPI_ADEC_AacInit 【需求】和【注意】涉及更新。
2017-05-27	00B02	第 2 次临时版本发布 1.1 小节中添加注意
2017-04-10	00B01	第 1 次临时版本发布



# 目 录

前 言 .....	i
1 音频组件 .....	1
1.1 概述 .....	1
1.2 重要概念 .....	1
1.3 API 参考 .....	5
1.4 数据类型 .....	15
1.5 错误码 .....	23



## 表格目录

表 1-1 音频编解码协议说明 .....	1
表 1-2 AAC Encoder 各协议码率设置 (码率单位为 kbps) .....	3
表 1-3 AAC Encoder Low Delay 协议码率设置 (码率单位为 kbps) .....	3
表 1-4 音频编码 API 错误码 .....	23
表 1-5 音频解码 API 错误码 .....	24



# 1 音频组件

## 1.1 概述

音频组件集成了 AAC 编解码协议，并开放接口，以便于用户集成第三方提供的编解码协议。AAC 编解码使用示例代码在 sample/audio 目录。

### 须知

客户如果需要使用 AAC 格式的专利，必须从版权权利人处获取授权，并缴纳 Licensing Fee。

## 1.2 重要概念

- 音频编解码协议

音频组件提供的编解码功能基于独立封装的 AAC 编解码库，核心编解码器工作在用户态，使用 CPU 软件编解码。

AAC 编解码协议说明如表 1-1 所示。

表1-1 音频编解码协议说明

协议	采样率	帧长 (采样点)	码率 (kbps)	压缩率	CPU 消耗	描述
AAC Encoder	8kHz , 16kHz,	• AACLC 支持 1024;	-	-	50 MHz	AAC 有两次突破性的技术升级：





协议	采样率	帧长 (采样点)	码率 (kbps)	压缩率	CPU 消耗	描述
	22.05k Hz, 24k Hz, 32k Hz, 44.1kH z, 48kH z	<ul style="list-style-type: none"> <li>EAAC 和 EAACPLUS 支持 2048;</li> <li>AACLD 和 AACEL D 支持 512。</li> </ul>				<ul style="list-style-type: none"> <li>aacPlus1 (即 EAAC) , 增加 SBR(带宽扩展)技术, 使得编解码器可以在比原来少一半的码率的条件下达到相同的音质。</li> <li>aacPlus2 (即 EAACPLUS) , 增加 PS(参数立体声)技术, 在低码率情况下得到极佳的音质效果, aacPlus2 可以在 48kbit/s 的速率下得到 CD 音质。</li> <li>AAC-LD 和 AAC-ELD 都为低时延语音编解码处理方案, 其中 AAC-LD 为安防行业标准需求, AAC-ELD 为未来通讯使用编码格式。</li> </ul> <p>码流范围与推荐码率设置如表 1-2、表 1-3 所示。</p>
AAC Decoder	兼容全部速率	512、1024、2048	-	-	25 MHz	<p>后向兼容。传统 AAC 解码器, 仅解码 aac Plus v1 码流低频信息, 而 aacPlus 解码器则可以同时还原高频信息。不支持 PS 的 AAC 解码器, 解码 aac Plus v2 码流时, 仅能得到单声道信息, 而 aacPlus2 解码器则可以得到立体声声音。注意: 解码方式需要选用 ADEC_MODE_STREAM。</p>

注: “cpu 消耗” 的结果值基于 ARM9 288MHz 环境, 2/2 MHz 表示编码和解码分别占有 2M 和 2M CPU。



表1-2 AAC Encoder 各协议码率设置 (码率单位为 kbps)

采样率	声道	LC BitRate		Plus v1 BitRate		Plus v2 BitRate	
		Supported	Preferred	Supported	Preferred	Supported	Preferred
8kHz	Mono	16 ~ 48	24	—	—	—	—
	Stereo	16 ~ 96	32	—	—	—	—
16kHz	Mono	24 ~ 96	48	24 ~ 48	32	—	—
	Stereo	24 ~ 192	48	24 ~ 96	32	16 ~ 48	32
22.05kHz	Mono	32 ~ 132	64	32 ~ 64	48	—	—
	Stereo	32 ~ 265	48	32 ~ 128	64	16 ~ 64	32
24kHz	Mono	32 ~ 144	48	32 ~ 64	48	—	—
	Stereo	32 ~ 288	48	32 ~ 128	64	16 ~ 64	32
32kHz	Mono	32 ~ 192	48	32 ~ 64	48	—	—
	Stereo	32 ~ 320	128	32 ~ 128	64	16 ~ 64	32
44.1kHz	Mono	48 ~ 265	64	32 ~ 64	48	—	—
	Stereo	48 ~ 320	128	32 ~ 128	64	16 ~ 64	48
48kHz	Mono	48 ~ 288	64	32 ~ 64	48	—	—
	Stereo	48 ~ 320	128	32 ~ 128	64	16 ~ 64	48

注：“—”表示不支持这种情况。

表1-3 AAC Encoder Low Delay 协议码率设置 (码率单位为 kbps)

采样率	声道	LD BitRate		ELD BitRate	
		Supported	Preferred	Supported	Preferred
8kHz	Mono	16~96	24	32~96	32
	Stereo	16~192	48	64~192	64
16kHz	Mono	24~192	48	16~256	48



采样率	声道	LD BitRate		ELD BitRate	
		Supported	Preferred	Supported	Preferred
	Stereo	32~320	96	32~320	96
22.05kHz	Mono	32~256	48	24~256	48
	Stereo	48~320	96	32~320	96
24kHz	Mono	32~256	64	24~256	64
	Stereo	48~320	128	32~320	128
32kHz	Mono	48~320	64	32~320	64
	Stereo	64~320	128	64~320	128
44.1kHz	Mono	64~320	128	96~320	128
	Stereo	44~320	256	192~320	256
48kHz	Mono	64~320	128	96~320	128
	Stereo	64~320	256	192~320	256

- 音频编解码集成接口

SDK 发布包中开放接口用于注册和注销编码器和解码器，音频组件根据这些接口，提供了注册 AAC 编解码器的示例。用户既可以参考示例，注册自有的第三方编解码器，也可直接使用音频组件提供的示例来注册并使用组件中的 AAC 编解码器。

【注意】

- AAC 编解码器的使用支持静态库注册的形式，相关库包括 libaaccomm.a、libaacenc.a、libaacsbrrenc.a、libaacdec.a、libaacsbrdec.a。其中，libaacsbrrenc.a 在 AAC 编码器不使用 EAAC 或者 EAACPLUS 类型编码时可裁剪；libaacsbrdec.a 在 AAC 解码器不使用 EAAC 或者 EAACPLUS 类型解码时可裁剪。当使用 EAAC 或者 EAACPLUS 编解码类型时，须在注册编解码器之前进行 SBRENC、SBRDEC 功能模块的静态注册。
- AAC 编解码器的使用还支持动态库调用的形式，相关库包括 libaaccomm.so、libaacenc.so、libaacsbrrenc.so、libaacdec.so、libaacsbrdec.so。其中，libaacsbrrenc.so 在 AAC 编码器不使用 EAAC 或者 EAACPLUS 类型编码时可裁剪；libaacsbrdec.so 在 AAC 解码器不使用 EAAC 或者 EAACPLUS 类型解码时可裁剪。



- AAC 编解码器不支持同时使用静态库注册和动态库调用。
- 仅  
Hi3516EV200/Hi3516EV300/Hi3518EV300/Hi3516DV200/Hi3568V100/Hi3559V200/Hi3556V200/Hi3562V100/Hi3566V100/Hi3559AV100/Hi3519AV100 支持 AAC 编解码器的静态库注册。
- 仅  
Hi3559AV100/Hi3519AV100/Hi3516CV500/Hi3516DV300/Hi3516AV300/Hi3559V200/Hi3556V200/Hi3562V100/Hi3566V100/Hi3516EV200/Hi3516EV300/Hi3518EV300/Hi3516DV200 支持 AAC 编解码器的动态库裁剪。
- AAC 编解码器的静态库注册只允许成功调用一次，不支持重复注册。

### 【举例】

下面的代码实现静态注册 AAC 的 SBRENC 和 SBRDEC 模块。

```
HI_S32 s32Ret;
HI_VOID* pSbrEncHandle = HI_AAC_SBRENC_GetHandle();

/* register SBRENC module. */
s32Ret = AACEncoderRegisterModule(pSbrEncHandle);
if(s32Ret)
{
    printf("register SBRENC module fail, s32Ret = %d\n", s32Ret);
}

HI_VOID* pSbrDecHandle = HI_AAC_SBRDEC_GetHandle();

/* register SBRDEC module. */
s32Ret = AACDecoderRegisterModule(pSbrDecHandle);
if(s32Ret)
{
    printf("register SBRDEC module fail, s32Ret = %d\n", s32Ret);
}
```

## 1.3 API 参考

SDK 发布包中的以下 API 用于注册和注销编码器和解码器。



- [HI\\_MPI\\_AENC\\_RegisterEncoder](#): 注册编码器。
- [HI\\_MPI\\_AENC\\_UnRegisterEncoder](#): 注销编码器。
- [HI\\_MPI\\_ADEC\\_RegisterDecoder](#): 注册解码器。
- [HI\\_MPI\\_ADEC\\_UnRegisterDecoder](#): 注销解码器。

音频组件中提供的注册示例:

- [HI\\_MPI\\_AENC\\_AacInit](#): 注册 AAC 编码器。
- [HI\\_MPI\\_ADEC\\_AacInit](#): 注册 AAC 解码器。

## HI\_MPI\_AENC\_RegisterEncoder

### 【描述】

注册编码器。

### 【语法】

```
HI_S32 HI_MPI_AENC_RegisterEncoder(HI_S32 *ps32Handle, AENC\_ENCODER\_S *pstEncoder);
```

### 【参数】

参数名称	描述	输入/输出
ps32Handle	注册句柄。	输出
pstEncoder	编码器属性结构体。	输入

### 【返回值】

返回值	描述
0	成功。
非 0	失败, 其值为 <a href="#">错误码</a> 。

### 【需求】

- 头文件: hi\_comm\_aenc.h、mpi\_aenc.h
- 库文件: libmpi.a

### 【注意】



- 用户通过传入编码器属性结构体，向 AENC 模块注册一个编码器，并返回注册句柄，用户可以最后通过注册句柄来注销该编码器。
- AENC 模块最大可注册 20 个编码器，且自身已注册 LPCM、G711a、G711u、G726、ADPCM 五个编码器。
- 同一种编码协议不允许重复注册编码器，例如假如已注册 AAC 编码器，不允许另外再注册一个 AAC 编码器。
- 编码器属性包括编码器类型、最大码流长度、编码器名称、打开编码器的函数指针、进行编码的函数指针、关闭编码器的函数指针。

- 编码器类型

SDK 以枚举标识编码协议，注册时应选择相关协议的编码器类型。

- 最大码流长度

每帧编码后码流的最大长度，AENC 模块将根据注册的最大码流长度分配内存大小。

- 编码器名称

编码器名称用字符串表示，用于在 proc 信息中显示。

- 打开编码器的函数指针

SDK 封装的一个函数指针，其函数原型为：

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

其中第一个参数是编码器属性，用于传入不同类型的编码器的特定属性；第二个参数是编码器句柄，用于返回可用于操作编码器的句柄。这两个参数均由用户封装，用户封装第二个参数时需要注意分配内存，因为编码器句柄还将用于编码和关闭编码器。

- 进行编码的函数指针

SDK 封装的一个函数指针，其函数原型为：

```
HI_S32 (*pfnEncodeFrm)(HI_VOID *pEncoder, const AUDIO_FRAME_S  
*pstData,  
HI_U8 *pu8Outbuf, HI_U32 *pu32OutLen);
```

第一个参数是上一个函数打开编码器时返回的编码器句柄；第二个参数是 SDK 的音频帧数据结构的指针，用于传入音频帧数据；第三个参数是输出缓存指针；第四个参数是输出缓存长度。

- 关闭编码器的函数指针

SDK 封装的一个函数指针，其函数原型为：

```
HI_S32 (*pfnCloseEncoder)(HI_VOID *pEncoder);
```



参数是打开编码器时返回的编码器句柄。

- 用户需根据这几个函数原型封装第三方编码器，并通过编码器属性结构体注册给 AENC 模块，从而实现第三方编码器的集成。
- 必须在创建编码通道前注册相关类型的编码器，编码器不需要重复注册。

#### 【举例】

下面的代码举例 AAC 编码器的注册：

```
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;
```

#### 【相关主题】

无。

## HI\_MPI\_AENC\_UnRegisterEncoder

#### 【描述】

注销解码器。

#### 【语法】

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

#### 【参数】



参数名称	描述	输入/输出
s32Handle	注册句柄（注册编码器时获得的句柄）。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为 <a href="#">错误码</a> 。

【需求】

- 头文件：hi\_comm\_aenc.h、mpi\_aenc.h
- 库文件：libmpi.a

【注意】

通常不需要注销编码器。

【举例】

无

【相关主题】

无。

## HI\_MPI\_ADEC\_RegisterDecoder

【描述】

注册解码器。

【语法】

```
HI_S32 HI_MPI_ADEC_RegisterDecoder(HI_S32 *ps32Handle, ADEC\_DECODER\_S *pstDecoder);
```

【参数】

参数名称	描述	输入/输出
ps32Handle	注册句柄。	输出





参数名称	描述	输入/输出
pstDecoder	解码器属性结构体。	输入

#### 【返回值】

返回值	描述
0	成功。
非 0	失败，其值为 <a href="#">错误码</a> 。

#### 【需求】

- 头文件：hi\_comm\_adec.h、mpi\_adec.h
- 库文件：libmpi.a

#### 【注意】

- 用户通过传入解码器属性结构体，向 ADEC 模块注册一个解码器，并返回注册句柄，用户可以最后通过注册句柄来注销该解码器。
- ADEC 模块最大可注册 20 个解码器，且自身已注册 LPCM、G711a、G711u、G726、ADPCM 五个解码器。
- 同一种解码协议不允许重复注册解码器，例如假如已注册 AAC 解码器，不允许另外再注册一个 AAC 解码器。
- 解码器属性包括解码器类型、解码器名称、打开解码器的函数指针、进行解码的函数指针、获取音频帧信息的函数指针、关闭解码器的函数指针。
  - 解码器类型  
SDK 以枚举标识解码协议，注册时应选择相关协议的解码器类型。
  - 解码器名称  
解码器名称用字符串表示，用于在 proc 信息中显示。
  - 打开解码器的函数指针  
SDK 封装的一个函数指针，其函数原型为：  
`HI_S32 (*pfnOpenDecoder)(HI_VOID *pDecoderAttr, HI_VOID **ppDecoder);`  
其中第一个参数是解码器属性，用于传入不同类型的解码器的特定属性；第二个参数是解码器句柄，用于返回可用于操作解码器的句柄。这两个参数均由用



户封装，用户封装第二个参数时需要注意分配内存，因为解码器句柄还将用于解码和关闭解码器。

– 进行解码的函数指针

SDK 封装的一个函数指针，其函数原型为：

```
HI_S32 (*pfnDecodeFrm)(HI_VOID *pDecoder, HI_U8 **pu8Inbuf, HI_S32  
*ps32LeftByte, HI_U16 *pu16Outbuf, HI_U32 *pu32OutLen, HI_U32  
*pu32Chns);
```

第一个参数是上一个函数打开解码器时返回的解码器句柄；第二个参数是输入缓存，用于传入音频帧数据；第三个参数用于返回剩余字节数，用于流式解码，即每次送入的音频帧数据不是完整的一帧的情形；第四个参数是输出缓存；第五个参数是输出数据的单声道长度；第六个参数是输出的通道数，码流数据经解码后，可能输出单声道，也可能输出立体声。

– 获取音频帧信息的函数指针

SDK 封装的一个函数指针，其函数原型为：

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

第一个参数是打开解码器时返回的解码器句柄；第二个参数是用户封装的音频帧信息，有的解码器解析码流时会获取解码后音频数据的采样点、采样率等信息；如果用户解码器不需要此函数，可以为该函数原型封装一个空函数。

– 关闭解码器的函数指针

SDK 封装的一个函数指针，其函数原型为：

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

参数是打开解码器时返回的解码器句柄。

– 用户需根据这几个函数原型封装第三方解码器，并通过解码器属性结构体注册给 ADEC 模块，从而实现第三方解码器的集成。

- 必须在创建解码通道前注册相关类型的解码器，解码器不需要重复注册。

【举例】

下面的代码举例 AAC 解码器的注册：

```
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;
```

【相关主题】

无。

## HI\_MPI\_ADEC\_UnRegisterDecoder

【描述】

注销解码器。

【语法】

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

【参数】

参数名称	描述	输入/输出
s32Handle	注册句柄（注册解码器时获得的句柄）。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为 <a href="#">错误码</a> 。

【需求】



- 头文件: hi\_comm\_adec.h、 mpi\_adec.h
- 库文件: libmpi.a

【注意】

通常不需要注销解码器。

【举例】

无

【相关主题】

无。

## HI\_MPI\_AENC\_AacInit

【描述】

注册 AAC 编码器。

【语法】

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

【参数】

无

【返回值】

返回值	描述
0	成功。
非 0	失败, 其值为 <a href="#">错误码</a> 。

【需求】

- 源文件: audio\_aac\_adp.c
- 头文件: audio\_aac\_adp.h
- 库文件: libaaccomm.so、 libaacenc.so

【注意】



此接口在 audio\_aac\_adp.c 里实现，而 audio\_aac\_adp.c 并没有封装成库，所以在使用此接口时，需要包含 audio\_aac\_adp.c 和 audio\_aac\_adp.h 才能编译通过。这两个文件默认放置在 sample/audio/adp 文件夹中。此外，在需要使用到 SBRENC 功能时，需要添加 libaacsbrenc.so 库。

**【举例】**

无。

**【相关主题】**

无。

## HI\_MPI\_ADEC\_AacInit

**【描述】**

注册 AAC 解码器。

**【语法】**

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

**【参数】**

无

**【返回值】**

返回值	描述
0	成功。
非 0	失败，其值为 <a href="#">错误码</a> 。

**【需求】**

- 源文件：audio\_aac\_adp.h
- 头文件：audio\_aac\_adp.h
- 库文件：libaaccomm.so、libaacdec.so

**【注意】**

请参考 [HI\\_MPI\\_AENC\\_AacInit](#) 接口注意项的说明。此外，在需要使用到 SBRDEC 功能时，需要添加 libaacsbrdec.so 库。



【举例】

无。

【相关主题】

无。

## 1.4 数据类型

音频组件相关数据类型、数据结构定义如下：

- [AENC\\_ENCODER\\_S](#)：定义编码器属性结构体。
- [ADEC\\_DECODER\\_S](#)：定义解码器属性结构体。
- [AAC\\_TYPE\\_E](#)：定义 AAC 音频编解码协议类型。
- [AAC\\_BPS\\_E](#)：定义 AAC 音频编码码率。
- [AAC\\_TRANS\\_TYPE\\_E](#)：定义 AAC 音频编解码协议传输封装类型。
- [AENC\\_ATTR\\_AAC\\_S](#)：定义 AAC 编码协议属性结构体。
- [ADEC\\_ATTR\\_AAC\\_S](#)：定义 AAC 解码协议属性结构体。

### AENC\_ENCODER\_S

【说明】

定义编码器属性结构体。

【定义】

```
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;
```

【成员】



成员名称	描述
enType	编码协议类型，见《HiMPP 媒体处理软件 Vxx 开发参考》“系统控制”章节。
u32MaxFrmLen	最大码流长度。
aszName	编码器名称。
pfnOpenEncoder	打开编码器的函数指针。
pfnEncodeFrm	进行编码的函数指针。
pfnCloseEncoder	关闭编码器的函数指针。

#### 【注意事项】

无。

#### 【相关数据类型及接口】

[HI\\_MPI\\_AENC\\_RegisterEncoder](#)

## ADEC\_DECODER\_S

#### 【说明】

定义解码器属性结构体。

#### 【定义】

```
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;
```

#### 【成员】



成员名称	描述
enType	解码协议类型，见《HiMPP 媒体处理软件 Vxx 开发参考》“系统控制”章节。
aszName	解码器名称。
pfnOpenDecoder	打开解码器的函数指针。
pfnDecodeFrm	进行解码的函数指针。
pfnGetFrmInfo	获取音频帧信息的函数指针
pfnCloseDecoder	关闭解码器的函数指针。

#### 【注意事项】

无。

#### 【相关数据类型及接口】

[HI\\_MPI\\_ADEC\\_RegisterDecoder](#)

## AAC\_TYPE\_E

#### 【说明】

定义 AAC 音频编解码协议类型。

#### 【定义】

```
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;
```

#### 【成员】





成员名称	描述
AAC_TYPE_AACLC	AACLC 格式。
AAC_TYPE_EAAC	eAAC 格式（也称为 HEAAC、AAC+或 aacPlusV1）。
AAC_TYPE_EAACPLUS	eAACPLUS 格式（也称为 AAC++或 aacPlusV2）。
AAC_TYPE_AACLD	AACLD 格式。
AAC_TYPE_AACELD	AACELD 格式。

**【注意事项】**

无。

**【相关数据类型及接口】**

无。

## AAC\_BPS\_E

**【说明】**

定义 AAC 音频编码码率。

**【定义】**

```
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;
```



### 【成员】

成员名称	描述
AAC_BPS_8K	8kbit/s
AAC_BPS_16K	16kbit/s
AAC_BPS_22K	22kbit/s
AAC_BPS_24K	24kbit/s
AAC_BPS_32K	32kbit/s
AAC_BPS_48K	48kbit/s
AAC_BPS_64K	64kbit/s
AAC_BPS_96K	96kbit/s
AAC_BPS_128K	128kbit/s
AAC_BPS_256K	256kbit/s
AAC_BPS_320K	320kbit/s

### 【注意事项】

无。

### 【相关数据类型及接口】

无。

## AAC\_TRANS\_TYPE\_E

### 【说明】

定义 AAC 音频编解码协议传输封装类型。

### 【定义】

```
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;
```



### 【成员】

成员名称	描述
AAC_TRANS_TYPE_ADTS	ADTS 封装格式。AACLC/EAAC/EAACPLUS 支持。
AAC_TRANS_TYPE_LOAS	LOAS 封装格式。 AACLC/EAAC/EAACPLUS/AACLD/AACELD 支持。
AAC_TRANS_TYPE_LATM_MCP1	LATM1 封装格式。 AACLC/EAAC/EAACPLUS/AACLD/AACELD 支持。

### 【注意事项】

LATM1 格式由于不具备同步帧头机制，在码流出现问题时无法快速恢复，**不推荐使用**。

### 【相关数据类型及接口】

无。

## AENC\_ATTR\_AAC\_S

### 【说明】

定义 AAC 编码协议属性结构体。

### 【定义】

```
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;
```



【成员】

成员名称	描述
enAACType	AAC 编码类型 (Profile) 。
enBitRate	编码码率。 取值范围： LC: 16 ~ 320; EAAC: 24 ~ 128; EAAC+: 16 ~ 64; AACLD: 16 ~ 320; AACELD: 32 ~ 320; 以 kbit/s 为单位。
enSmpRate	音频数据的采样率。 取值范围： LC: 8 ~ 48; EAAC: 16 ~ 48; EAAC+: 16 ~ 48。 AACLD: 8 ~ 48; AACELD: 8 ~ 48; 以 kHz 为单位。
enBitWidth	音频数据采样精度，只支持 16bit。
enSoundMode	输入数据的声道模式。支持输入为单声道或双声道。
enTransType	AAC 传输封装类型。 取值范围： <ul style="list-style-type: none"><li>• AAC_TRANS_TYPE_ADTS: 0</li><li>• AAC_TRANS_TYPE_LOAS: 1</li><li>• AAC_TRANS_TYPE_LATM_MCP1: 2</li></ul>
s16BandWidth	目标频段范围。取值范围是，0 或 1000~enSmpRate/2，单位 Hz



【注意事项】

无。

【相关数据类型及接口】

无。

## ADEC\_ATTR\_AAC\_S

【说明】

定义 AAC 解码协议属性结构体。

【定义】

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

【成员】

成员名称	描述
enTransType	AAC 传输封装类型。 取值范围： <ul style="list-style-type: none"><li>• AAC_TRANS_TYPE_ADTS: 0</li><li>• AAC_TRANS_TYPE_LOAS: 1</li><li>• AAC_TRANS_TYPE_LATM_MCP1: 2</li></ul>

【注意事项】

无。

【相关数据类型及接口】

无。



## 1.5 错误码

### 音频编码错误码

音频编码 API 错误码如表 1-4 所示。

表1-4 音频编码 API 错误码

错误代码	宏定义	描述
0xA0178001	HI_ERR_AENC_INVALID_DEVID	音频设备号无效
0xA0178002	HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效
0xA0178003	HI_ERR_AENC_ILLEGAL_PARAM	音频编码参数设置无效
0xA0178004	HI_ERR_AENC_EXIST	音频编码通道已经创建
0xA0178005	HI_ERR_AENC_UNEXIST	音频编码通道未创建
0xA0178006	HI_ERR_AENC_NULL_PTR	输入参数空指针错误
0xA0178007	HI_ERR_AENC_NOT_CONFIG	编码通道未配置
0xA0178008	HI_ERR_AENC_NOT_SUPPORT	操作不被支持
0xA0178009	HI_ERR_AENC_NOT_PERM	操作不允许
0xA017800C	HI_ERR_AENC_NOMEM	系统内存不足
0xA017800D	HI_ERR_AENC_NOBUF	编码通道缓存分配失败
0xA017800E	HI_ERR_AENC_BUF_EMPTY	编码通道缓存空
0xA017800F	HI_ERR_AENC_BUF_FULL	编码通道缓存满
0xA0178010	HI_ERR_AENC_SYS_NOTREADY	系统没有初始化
0xA0178040	HI_ERR_AENC_ENCODER_ERR	音频编码数据错误

### 音频解码错误码

音频解码 API 错误码如表 1-5 所示。



表1-5 音频解码 API 错误码

错误代码	宏定义	描述
0xA0188001	HI_ERR_ADEC_INVALID_DEVID	音频解码设备号无效
0xA0188002	HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效
0xA0188003	HI_ERR_ADEC_ILLEGAL_PARAM	音频解码参数设置无效
0xA0188004	HI_ERR_ADEC_EXIST	音频解码通道已经创建
0xA0188005	HI_ERR_ADEC_UNEXIST	音频解码通道未创建
0xA0188006	HI_ERR_ADEC_NULL_PTR	输入参数空指针错误
0xA0188007	HI_ERR_ADEC_NOT_CONFIG	解码通道属性未配置
0xA0188008	HI_ERR_ADEC_NOT_SUPPORT	操作不被支持
0xA0188009	HI_ERR_ADEC_NOT_PERM	操作不允许
0xA018800C	HI_ERR_ADEC_NOMEM	系统内存不足
0xA018800D	HI_ERR_ADEC_NOBUF	解码通道缓存分配失败
0xA018800E	HI_ERR_ADEC_BUF_EMPTY	解码通道缓存空
0xA018800F	HI_ERR_ADEC_BUF_FULL	解码通道缓存满
0xA0188010	HI_ERR_ADEC_SYS_NOTREADY	系统没有初始化
0xA0188040	HI_ERR_ADEC_DECODER_ERR	音频解码数据错误