ANYKA
芯 随 你 动

版本: **1.0.7** **2014** 年 **03** 月

音频库

接口说明

声　　明

本手册不对包括但不限于下列事项担保：适销性、特殊用途的适用性；实施该用途不会侵害第三方的知识产权等权利。

安凯技术公司不对由使用本手册或执行本手册内容而带来的任何损害负责。

本手册是按当前的状态提供参考，随附产品或本书内容如有更改，恕不另行通知。

联　系　方　式

安凯（广州）微电子技术有限公司

地址：广州科学城科学大道 182 号创新大厦 C1 区 3 楼
电话: (86)-20-3221 9000

传真: (86)-20-3221 9258

邮编: 510663

销售热线:

(86)-20-3221 9499

电子邮箱:

sales@anyka.com

主页:

http: //www. anyka.com

版 本 变 更 说 明

以下表格对于本文档的版本变更做一个简要的说明。版本变更仅限于技术内容的变更，不包括版式、格式、句法等的变更。

版本	说明	完成日期
V1.0.0	正式发布	2011 年 12 月
V1.0.1	1. 按照文档规范进行整理 2. 增加编码及音效处理数据结构说明 3. 增加编码 及 DAbuffer 接口调用示例 4. 补充完善解码数据结构中一些不合理或遗漏的描述 5. 增加与媒体库的对应关系 6. 根据评审意见修改	2011 年 12 月
V1.0.2	1. 增加音量设置接口 （3.2.2 节和 4.7.8） 2. 修改 1.1.1 中 midi 支持的说明 3. 根据评审已经修改图 2-1 的箭头表达	2012 年 03 月
V1.0.3	1. 增加_SD_SetInbufMinLen 接口说明	2012 年 05 月
V1.0.4	1. 修改_SD_SetHandle 的参数说明的笔误 2. 增加“变速倍数对照表” 3. 增加 _SD_Decode_ParseFHead 和 _SD_Decode_Open_Fast 两个函数的接口描述，及其调用示例 4. 增加 speex 编、解码的接口说明，及调用示例 5. 增加_SD_Encode_Last 函数接口描述，及调用示例 6. 增加_SD_Decode_Seek()函数，及 T_AUDIO_SEEK_INFO 结构体的说明	2012 年 06 月
V1.0.5	1. 增加_SD_Get_Input_Buf_Info 函数接口描述 2. 修改_SD_SetInbufMinLen 的描述 3. T_AUDIO_SEEK_INFO 增加新成员	2012 年 09 月

V1.0.6	1. 对 5.2 的码流填充示例进行补充，以免误解。 2. 增加_SD_Encode_SetFramHeadFlag 函数接口描述	2013 年 05 月
V1.0.7	1. 增加_SD_GetAudioSpectrum_equNum 函数接口描述 2. 增加_SD_GetAudioSpectrumComplex 函数接口描述 3. 修改 m_eq 成员，增加设置每个频点对应的 Q 值的接口 4. 修改 m_agc 成员，设置降噪和 agc 的效果 5. 增加 _SD_Filter_Audio_Scale 接口说明	2014 年 03 月

与媒体播放库的版本对应关系

Corresponding audio codec lib	Corresponding filter lib	Corresponding spotlight media lib	Corresponding AVD media lib	音频库接口说明文档 Version	备注
V1.10.00	V1.3.00	V1.10.00	V1.16.06	V1.0.7	

目录

1 前言

本说明文档涉及的音频库，适用于安凯微电子公司发布的 AK10XX、AK11XX、AK37XX、AK780X、AK880X 和 AK98XX 等芯片。

1.1 支持的音频编解码格式

1.1.1 解码

格式	说明
WMA	支持 48K 以下采样率，比特率最高到 320kps ，不支持无损 WMA
AAC	AAC_LC 格式下最高比特率可到 320kps；支持 AAC+
AMR	支持 AMR-NB
WAVE	支持 PCM 及 MS/IMA ADPCM 两种模式 其它的不支持
MIDI	支持 GM 标准的 MIDI，不支持文件长度大于 128K 的 MIDI 且 spotlight 平台不支持 midi 播放。
FLAC	支持 Native FLAC,OGG FLAC 格式
APE	支持 fast,normal,high, extra high,不支持 insane 模式
DRA	只支持声道数小于等于 2 的模式
MP3	所有比特率都支持 支持 mp3pro
Ogg vorbis	支持到 48KHz，立体声
AC3	支持到 6 声道，48KHz 采样率，比特率不限
G.711	支持编解码, A-LAW 和 U-LAW
SBC	支持蓝牙立体声的 SBC 解码；采样率支持 16k，32k，44.1k，48k； 声道支持单双声道。
speex	支持 8k，单、双声道的解码

1.1.2 编码

格式	说明
MP3	支持 48K 以下采样率，其中 22050 以下采样率支持双声道，22050 以上只能支持单声道。
AAC	AAC_LC 格式
AMR	支持 AMR-NB，模式支持 475，515，590，670，740，795，102，122
WAV	支持 LPCM 及 IMA ADPCM 两种模式, 其它的不支持
speex	支持 8k，单、双声道的编码

1.2 支持的FILTER

- 音效处理一般都需要一些时间，由于系统性能限制对有些歌曲本身解码就慢的，不能支持音效处理了，例如 APE 一般就支持不了音效。

音效	说明
EQ	支持自然、流行、爵士、摇滚、经典等内置音效模式及用户自定义模式。EQ 有两套算法： 1.老的一套算法，音量不会强制变小，所以缺点就是当歌曲音量本身比较大的时候，EQ 以后可能溢出，产生噪音。 2. 新的一套算法，在 EQ 之前会根据每个滤波器的最大音量值，先将歌曲整体音量变小，优点是不会产生溢出，缺点是 EQ 之后可能是歌曲整体音量变小。 注：这两套算法的选择，要在库申请的时候说明，只编译一套发布，不可编程选择。
变速不变调	变速范围是 0.5～2 倍，变速有两套算法： 老的一套算法效果比较差，但速度比较快； 新的一套算法效果比较好，但速度比较慢。 就目前系统应用情况来看，一般都还是选择老的一套算法。 注：算法可通过编程选择，具体选择方法，见后面的说明。
重采样	可以重采样成想要的采样率，支持上采样和下采样。

3DSound	效果不好，目前基本没应用
降噪	降噪算法比较耗时，性能上具体支持到什么程度需进一步测试。 降噪一般是用在录音上。

附：**变速倍数对照表**

算法 0 的时候，级数为 0~15，5 为正常速度，0 为最慢级速（0.5 倍速），15 为最快级速（2 倍速）；

算法 1 的时候，级数为 0~10，5 为正常速度，0 为最慢级速(0.5 倍速)，10 为最快级速（2 倍速）

算法 0																
参数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
倍数	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2.0
算法 1																
参数	0	1	2	3	4	5	6	7	8	9	10	/	/	/	/	/
倍数	0.5	0.6	0.7	0.8	0.9	1.0	1.2	1.4	1.6	1.8	2.0					

2 流程图

2.1 播放流程

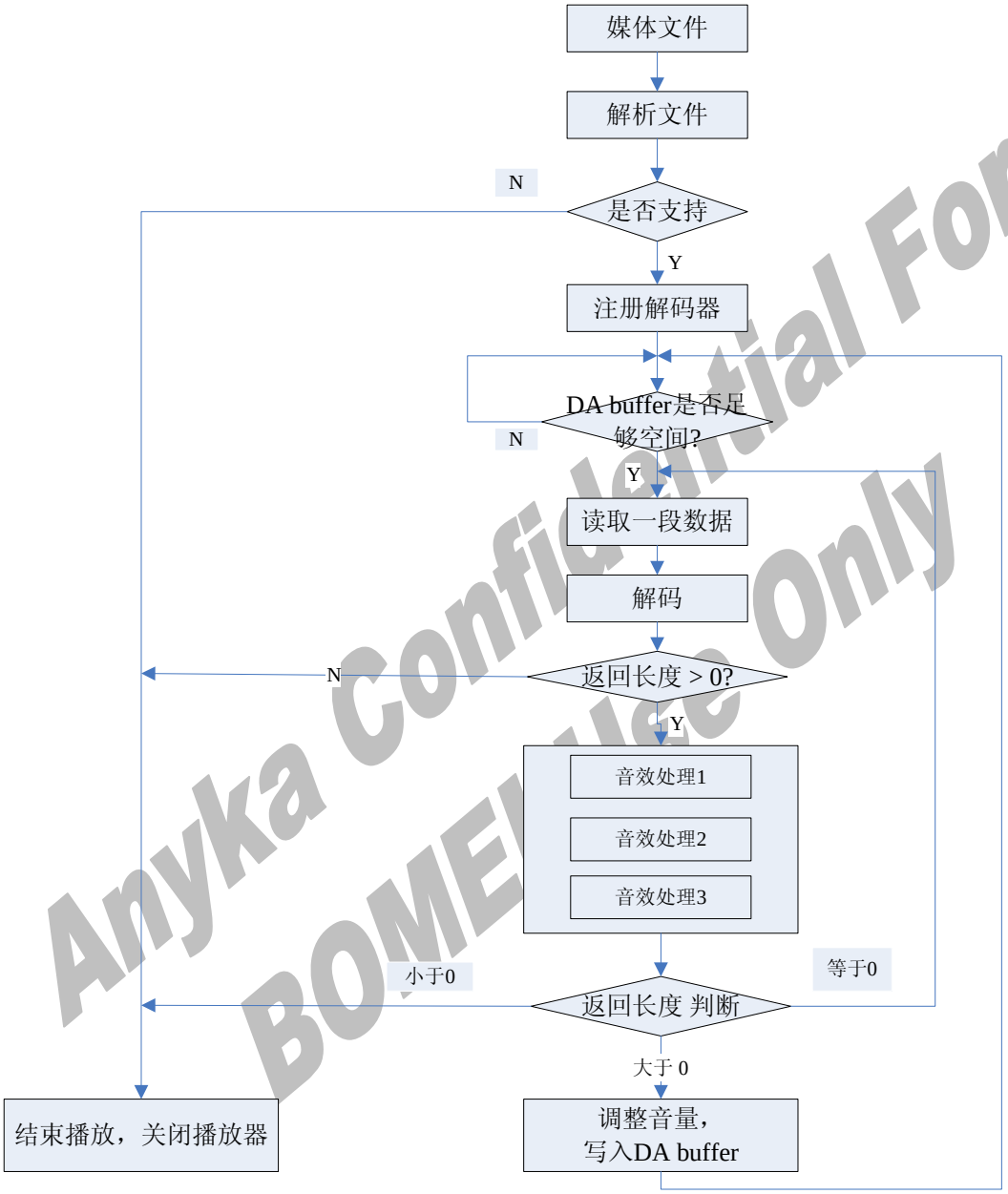

图 2-1 播放流程图

2.2 录音流程

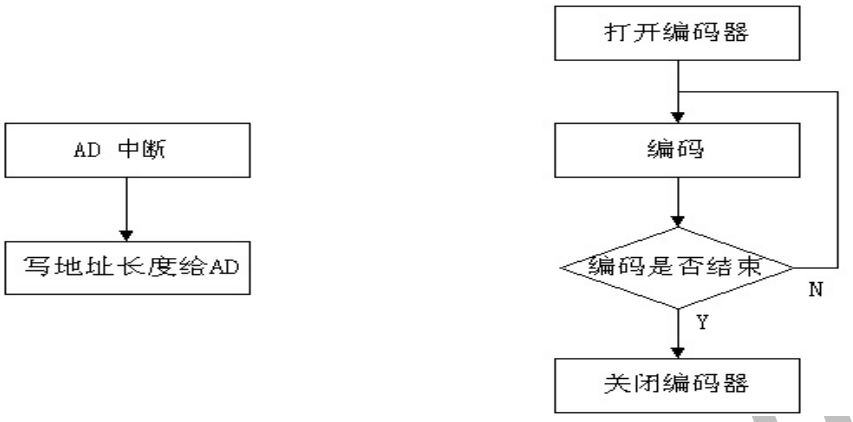

图 2-2 录音流程

3 数据结构

3.1 公共数据结构

3.1.1 T_AUDIO_CB_FUNS

```
typedef struct
{
        MEDIALIB_CALLBACK_FUN_MALLOC                    Malloc;
        MEDIALIB_CALLBACK_FUN_FREE                      Free;
        MEDIALIB_CALLBACK_FUN_PRINTF                            printf;
        MEDIALIB_CALLBACK_FUN_RTC_DELAY                 delay;
        MEDIALIB_CALLBACK_FUN_CMMBSYNCTIME              cmmbsynctime;
        MEDIALIB_CALLBACK_FUN_CMMBAUDIORECDATA  cmmbaudiorecdata;
#ifdef FOR_SPOTLIGHT
        MEDIALIB_CALLBACK_FUN_READ                      read;
        MEDIALIB_CALLBACK_FUN_SEEK                      seek;
        MEDIALIB_CALLBACK_FUN_TELL                      tell;
#endif
}T_AUDIO_CB_FUNS;
```

结构名	T_AUDIO_CB_FUNS	
定义概述	回调函数	
成员说明	MEDIALIB_CALLBACK_FUN_MALLOC	分配内存回调函数
	MEDIALIB_CALLBACK_FUN_FREE	释放内存回调函数
	MEDIALIB_CALLBACK_FUN_PRINTF	打印函数
	MEDIALIB_CALLBACK_FUN_RTC_DELAY	延时函数
	MEDIALIB_CALLBACK_FUN_CMMBSYNCTIME	Cmmb 播放时取得时间撮回调函数
	MEDIALIB_CALLBACK_FUN_CMMBAUDIORECDATA	CMMB 录像的音频录制回调函数
	MEDIALIB_CALLBACK_FUN_READ	只对 Spotlight 平台有用，用来读文件的回调函数
	MEDIALIB_CALLBACK_FUN_SEEK	只对 Spotlight 平台有用，用来 seek 文件的回调函数
	MEDIALIB_CALLBACK_FUN_TELL	只对 Spotlight 平台有用，用来取得当前读文件位置的回调函数

3.2 解码数据结构

3.2.1 T_AUDIO_DECODE_STRUCT

```
typedef struct {
        T_AUDIO_CB_FUNS     cb_fun;
        T_AUDIO_IN_INFO     info;
        T_AUDIO_BUF         in_buf;  //U8
        T_AUDIO_BUF         out_buf;//U16
        T_AUDIO_MEDIA_FUN media_fun;
        T_AUDIO_SEEK_INFO seek_info;
        T_VOID *pdec_strc;
        T_S64 sample_count;
```

T_VOID *pHandle;

T_U8 bufmode;

T_U32 inbufminlen; //minimum data length for decoding once

}T_AUDIO_DECODE_STRUCT;

结构名	T_AUDIO_DECODE_STRUCT	
定义概述	每个解码器的全局结构变量	
成员说明	cb_fun	上层传入的 callback 函数，包括 malloc 等
	info	输入音频格式的信息
	in_buf	编解码内部缓冲
	out_buf	编解码输出 buffer 缓冲。
	media_fun	每个解码器的对外函数结构
	pdec_strc	每个解码器全局变量结构
	sample_count	解码出来的 PCM 计数,用于音视频同步
	pHandle	解码句柄，用于一些特殊使用场合，如 CMMB
	bufmode	缓冲模式，用于设置是即时解码还是缓冲解码： bufmode = _SD_BM_NORMAL，缓冲解码，解码器会要求输入 buffer 中缓冲有一定长度的码流后才进行解码 bufmode = _SD_BM_LIVE，即时解码，不要求输入 buffer 中有多少码流，被调用即无条件解码，如果解码过程中发现码流不足，解码失败。 注：如果设置为 Normal 模式，在解码结束时（即上层确定码流已经完成，但音频输入缓冲中仍有未解完的码流）,需要将 bufmode 设置为 _SD_BM_ENDING，才能将音频输入 buffer 中最后剩余的码流解码。
	Inbufminlen	Bufmode 为 Normal 模式时，用于设置进行解码所需要的缓冲码流的长度

3.2.2 T_AUDIO_IN_INFO

```
typedef struct
{
#ifdef FOR_SPOTLIGHT
        T_S32   haudio;
#endif
        T_U32  m_Type;                  //media type
        T_U32  m_SampleRate;            //sample rate, sample per second
        T_U16  m_Channels;              //channel number
        T_U16  m_BitsPerSample;         //bits per sample

        T_U32   m_InbufLen;              //input buffer length
        T_U8    *m_szData;
        T_U32   m_szDataLen;

        union {
                struct
                {
                        T_U32  cmmb_adts_flag;
                }m_aac;
                struct
                {
                        T_U32  nFileSize;
                } m_midi;
        }m_Private;
#ifdef BLUETOOTH_PLAY
        T_U32  decVolEna;   // decode volume enable::  1: 音频库里解码的时候做音量控制, 0:音频库里解码的时候不做音量控制
        T_U32  decVolume;   // decode volume value:: this volume is effective, when decVolCtl==1
#endif
}T_AUDIO_IN_INFO;
```

结构名	T_AUDIO_IN_INFO	
定义概述	需要解码音频的信息	
	haudio	只对 Spotlight 平台有用，播放的媒体资源句柄
	m_Type	音频流类型： _SD_MEDIA_TYPE_UNKNOWN , _SD_MEDIA_TYPE_MIDI , _SD_MEDIA_TYPE_MP3 , _SD_MEDIA_TYPE_AMR , _SD_MEDIA_TYPE_AAC , _SD_MEDIA_TYPE_WMA , _SD_MEDIA_TYPE_PCM , _SD_MEDIA_TYPE_ADPCM_IMA _SD_MEDIA_TYPE_ADPCM_MS , _SD_MEDIA_TYPE_ADPCM_FLASH _SD_MEDIA_TYPE_APE , _SD_MEDIA_TYPE_FLAC , _SD_MEDIA_TYPE_OGG_FLAC , _SD_MEDIA_TYPE_RA8LBR , _SD_MEDIA_TYPE_DRA, _SD_MEDIA_TYPE_OGG_VORBIS, _SD_MEDIA_TYPE_AC3, _SD_MEDIA_TYPE_PCM_ALAW, _SD_MEDIA_TYPE_PCM_ULAW, _SD_MEDIA_TYPE_SBC, _SD_MEDIA_TYPE_SPEEX
	m_SampleRate	采样率
	m_Channels	声道数
	m_BitsPerSample	每个 sample 位数

结构名	T_AUDIO_IN_INFO	
	m_InbufLen	设置音频解码输入 buffer 的大小，如果设置为 0，会使用音频库内部定义的缺省值；如果非 0，但设置的值小于各个解码器的最低要求值，则设置无效，使用解码器给定的最低要求值。各解码器最低要求的输入 buffer 大小这里不罗列(因为可能经常调整这个值)。 音频库里面会根据这个 m_InbufLen 信息，分配相应大小的解码输入 buffer。
	m_szData	一个数据指针，长度为 m_szDataLen，存储了各个解码器在文件解析后的一些信息
	m_szDataLen	m_szData 指针指向数据的长度
	m_Private	部分类型私有信息 1、 cmmb_adts_flag 用于 AAC 解码器： bit[1]:是否支持 CMMB 节目 aac 音频录制 bit[2]:是否支持 CMMB AAC 音频 SBR 解码。cmmb_adts_flag 的值非 0 就表示是 CMMB 的 AAC 解码 例如： 1) CMMB 无 SBR 解码，无录制：设为 1； 2) CMMB 无 SBR 解码，带录制： 设为 2； 3) CMMB 带 SBR 解码，无录制：设为 4； 4) CMMB 带 SBR 解码， 带录制：设为 6； 5) 非 CMMB 的 AAC 解码：设为 0； 2、 nFileSize 用于 MIDI 合成 由于 MIDI 的文件一般比较小，在音频库中，会将整个 MIDI 文件一次性 load 到输入 buffer 中。文件长度信息也需要传给解码器
	decVolEna	解码时是否做音量缩放的标志位： 1: 音频库里解码的时候做音量控制，0:音频库里解码的时候不做音量控制 说明：目前这个变量只对 10 芯片的蓝牙平台开放

结构名	T_AUDIO_IN_INFO	
	decVolume	解码时的音量值： 只有 decVolEna==1 的时候，这个音量值才能生效。 说明：目前这个变量只对 10 芯片的蓝牙平台开放

3.2.3 T_AUDIO_BUF

```
typedef struct {
        T_U32  buf_rp;          // if ReadPos == WritePos, means the Buffer is empty
        T_U32  buf_wp;
        T_U32  buf_len;// buffer total length
        T_VOID      *pbuf;          // buffer address

        T_VOID      *bkbuf;         //tempbuf
        T_U32       lockbuf_len;    //be used len
        T_BOOL      using_bkbuf;    //flag
}T_AUDIO_BUF;
```

结构名	T_AUDIO_BUF	
定义概述	解码器输入输出 buffer 结构	
成员说明	buf_rp	读指针，如果 buf_rp== buf_wp，表示 buffer 为空
	buf_wp	写指针
	buf_len	Buffer 长度
	pbuf	Buffer 指针
	bkbuf;	库内部使用，外面不需要用。
	lockbuf_len	库内部使用，外面不需要用。
	using_bkbuf	库内部使用，外面不需要用。

3.2.4 T_AUDIO_DECODE_OUT

typedef struct

{

T_U32 m_SampleRate; //sample rate, sample per second

T_U16 m_Channels; //channel number

T_U16 m_BitsPerSample; //bits per sample

T_U32 m_ulSize; //m_pBuffer lenth

T_U32 m_ulDecDataSize; //decode return data lenth

T_U8 *m_pBuffer; //out sample fill buffer

}T_AUDIO_DECODE_OUT;

结构名	T_AUDIO_DECODE_OUT	
定义概述	解码器输出信息结构体	
成员说明	m_SampleRate	解码返回的采样率
	m_Channels	解码返回的声道数
	m_BitsPerSample	解码返回的样点 bit 数
	m_ulSize	解码输出 buffer 的总大小 注意： 这个变量，在调用_SD_Decode_Open()的时候是输出参数，意思是建议给输出 buffer 分配的空间大小，实际上因为内存紧张等原因可以不分配这么大； 在调用_SD_Decode（）的时候是输入参数，每次调用 sd_decode 的时候都要给这个变量设值。
	m_ulDecDataSize	解码输出 buffer 的有效输出长度
	m_pBuffer	解码输出 buffer 的指针

3.2.5 T_AUDIO_SEEK_INFO

```
typedef struct
{
    T_S32   real_time;
    union {
        struct{
            T_U32 nCurBitIndex;
            T_U32 nFrameIndex;
        } m_ape;
        struct{
            T_U32 Indx;
            T_U32 offset;
            T_BOOL flag;
            T_U32  last_granu;
            T_U32  now_granu;
            T_BOOL is_eos;
            T_U32  re_data;
            T_U32  pack_no;
            T_U32  list[255];
        }m_speex;
        struct{
            T_U8   secUse;
            T_U8   secLen;
            T_U8   tmpSec;
            T_BOOL is_eos;
            T_BOOL is_bos;
            T_U8   endpack;
            T_U32  gos;
            T_U32  high_gos;
            T_U8  list[255];
        }m_vorbis;
    }m_Private;
```

}T_AUDIO_SEEK_INFO;

结构名	T_AUDIO_SEEK_INFO	
定义概述	Seek 后需要 demuxer 库传给音频库的一些信息	
成员说明	real_time	实际 Seek 到的位置的时间

	m_Private	不同格式音频，所需要的不同的私有信息。这个接口是 seek 时 demuxer 库传数据给音频库用的，平台一般不需要使用。
		m_ape：ape 的私有信息
		m_speex：speex 的私有信息
		struct{
		T_U32 nCurBitIndex; //不用，预留
		T_U32 nFrameIndex; // 第几帧
		} m_ape;
		struct{
		T_U32 Indx; // seek 位置开始，当前 page 中的 packet 数
		T_U32 offset; //seek 后,当前 page 剩余需要解码的数据大小
		T_BOOL flag; //seek 标志位，seek 时置 1
		T_U32 last_granu; //解码完上一个 page 时解码出的总 sample 数
		T_U32 now_granu; //解码完当前 page 后，解出的总样点数
		T_BOOL is_eos; //码流结束标志
		T_U32 re_data; //seek 后，当前 page 含有不完整 packet 大小
		T_U32 pack_no; //seek 的位置在当前 page 中的第几个 packet
		T_U32 list[255]; //记录 seek 后的 ogg page 中每个 packet 的大小
		}m_speex;
		struct{
		T_U8 secUse; //已经读取的 section 数目
		T_U8 secLen; //一个 page 中包含的 section 数
		T_U8 tmpSec; //已经解码的 section 数目
		T_BOOL is_eos; //是不是最后一个 page
		T_BOOL is_bos; //是不是第一个 page

		T_U8 endpack; //当前 page 中最后一个 packet 的位置
		//解码出的 sample 数是一个 64 位的数，目前只取低 32 位
		T_U32 gos; //解码完当前 page 后解码出总的 sample 数，低 32 位
		T_U32 high_gos; //解码完当前 page 后解码出总的 sample 数，高 32 位，(暂时不用，留给以后需要)
		T_U8 list[255]; //记录一个 page 中每个 section 的大小，一个 page 中最多含有 255 个 section
		}m_vorbis;

3.3 编码数据解构

3.3.1 T_AUDIO_REC_INPUT

typedef struct

{

T_AUDIO_CB_FUNS cb_fun;

T_AUDIO_ENC_IN_INFO enc_in_info;

}T_AUDIO_REC_INPUT;

结构名	T_AUDIO_FILTER_INPUT	
定义概述	音效处理 open 时的输入参数结构体	
成员说明	cb_fun	上层传入的 callback 函数，包括 malloc 等
	enc_in_info	输入音频格式的信息

3.3.2 T_AUDIO_ENC_IN_INFO

typedef struct

{

T_U32 m_Type;

T_U16 m_nChannel;

T_U16 m_BitsPerSample;

T_U32 m_nSampleRate;

union{

```
        struct{
                T_AUDIO_AMR_ENCODE_MODE mode;
        }m_amr_enc;
        struct{
                T_U32 enc_bits;
        }m_adpcm;
        struct{
                T_U32 bitrate;
                T_BOOL mono_from_stereo;
        }m_mp3;
        struct{
                T_U32 bitrate;
                T_BOOL cbr;
                T_BOOL dtx_disable;
                char *comments[64];
        }m_speex;
    }m_private;
    T_U32 encEndFlag;
    }m_private;
}T_AUDIO_ENC_IN_INFO;
```

结构名	T_AUDIO_ENC_IN_INFO	
定义概述	编码输入的音频信息	
	m_Type	编码类型，应该是下面之一： _SD_MEDIA_TYPE_UNKNOWN , _SD_MEDIA_TYPE_MP3 , _SD_MEDIA_TYPE_AMR , _SD_MEDIA_TYPE_AAC , _SD_MEDIA_TYPE_ADPCM_IMA , _SD_MEDIA_TYPE_SPEEX
	m_nChannel	要编码的声道数，2 为双声道，1 为单声道
	m_BitsPerSample	每个样点的 bit 数，**固定为 16**

结构名	T_AUDIO_ENC_IN_INFO	
	m_nSampleRate	采样率
	m_Private	部分类型私有信息 1、m_amr_enc 用于 AMR 编码 mode：amr 的编码模式： AMR_ENC_MR475 = 0, AMR_ENC_MR515, AMR_ENC_MR59, AMR_ENC_MR67, AMR_ENC_MR74, AMR_ENC_MR795, AMR_ENC_MR102, AMR_ENC_MR122 2、m_adpcm 用于 ADPCM 编码 enc_bits：编码后每个样点的 bit 数，**设置为 4** 3、 m_mp3 用于 MP3 编码 bitrate：编码后的比特率，可设置也可不设置，不设置的时候编码器内部会使用默认的比特率。 mono_from_stereo：预留，暂时没用到 4、 m_speex 用于 speex 编码 bitrate：设置 speex 的编码比特率 cbr：设置编码方式是 CBR 还是 VBR. 1:CBR ； 0:VBR；系统默认为 VBR dtx_disable：设置是否关闭 DTX 功能。 1：关闭 dtx 功能； 0：不关闭 dtx 功能； 系统默认是不关闭。 Comments：预留，可以填充歌曲的 commen 信息，不填也可以。

结构名	T_AUDIO_ENC_IN_INFO	
	encEndFlag	编码结束表示，即用户按结束录音键的时候，要设置这位为 1。 目前这个主要是给 speex 用，因为 speex 编码结束的时候，要给码流写个特殊标志位。

3.3.3 T_AUDIO_ENC_OUT_INFO

```
typedef struct
{
        T_U16  wFormatTag;
        T_U16  nChannels;
        T_U32  nSamplesPerSec;
        union {
                struct {
                        T_U32  nAvgBytesPerSec;
                        T_U16  nBlockAlign;
                        T_U16  wBitsPerSample;
                        T_U16  nSamplesPerPacket;
                } m_adpcm;
        }m_Private;
}T_AUDIO_ENC_OUT_INFO;
```

结构名	T_AUDIO_ENC_OUT_INFO	
定义概述	编码输出的音频信息	
	wFormatTag	针对 wave 录音，标识是哪种格式 wave，例如 0x1 表示是 LPCM 0x11 表示是 IMA 的 ADPCM 这个信息是要写到 wave 文件头中的，具体如何写可以参考后面的编码调用示例（只说明了 IMA 的 ADPCM 怎么写）
	nChannels	要编码的声道数，2 为双声道，1 为单声道

结构名	T_AUDIO_ENC_OUT_INFO	
	nSamplesPerSec	采样率
	m_Private	部分类型私有信息 m_adpcm 用于 ADPCM 编码，表示 ADPCM 编码后的码流特性，这些信息都是要写到文件头中的。（具体如何写可以参考后面的编码调用示例）

3.3.4 T_AUDIO_ENC_BUF_STRC

typedef struct

{

T_VOID *buf_in;

T_VOID *buf_out;

T_U32 len_in;

T_U32 len_out;

}T_AUDIO_ENC_BUF_STRC;

结构名	T_AUDIO_ENC_BUF_STRC	
定义概述	编码的输入输出 buffer	
	buf_in	编码的输入 buffer 指针
	buf_out	编码的输出 buffer 指针，该 buffer 大小不能小于 buf_in
	len_in	输入有效数据长度
	len_out	编码前表示输出 buffer 的可用空间大小; 编码后表示编码输出数据长度。

3.4 音效处理数据结构

3.4.1 T_AUDIO_FILTER_INPUT

typedef struct

{

T_AUDIO_FILTER_CB_FUNS cb_fun;

```
    T_AUDIO_FILTER_IN_INFO    m_info;
}T_AUDIO_FILTER_INPUT;
```

结构名	T_AUDIO_FILTER_INPUT	
定义概述	音效处理 open 时的输入参数结构体	
成员说明	cb_fun	上层传入的 callback 函数，包括 malloc 等
	m_info	输入音频格式的信息

3.4.2 T_AUDIO_FILTER_IN_INFO

```
typedef struct
{
    T_U32  m_Type;              //media type
    T_U32  m_SampleRate;        //sample rate, sample per second
    T_U16  m_Channels;          //channel number
    T_U16  m_BitsPerSample;     //bits per sample
    union {
        struct {
            T_EQ_MODE eqmode;
            // Reserved for Old Interface
            T_U8    m_FilterNum ;   //Filter Number
            double  m_g;            //gain by DB
            // New Interface
            T_U32 rectification;    // 0 ~ 1024, pre-amplified volume
            // For User Presets
            T_U32 bands;    //1~10
            T_U32 bandfreqs[_SD_EQ_MAX_BANDS];
            T_S16 bandgains[_SD_EQ_MAX_BANDS];
            T_U16 bandQ[_SD_EQ_MAX_BANDS];
        } m_eq;
        struct {
            T_WSOLA_TEMPO tempo;
    T_WSOLA_ARITHMATIC arithmeticChoice;
```

```
    } m_wsola;
    struct{
        T_U8 is3DSurround;
    }m_3dsound;
    struct {
        //目标采样率 1:48k 2:44k 3:32k 4:24K 5:22K 6:16K 7:12K 8:11K 9:8K
        T_RES_OUTSR  outSrindex;

        //设置最大输入长度(bytes)，open 时需要用作动态分配的依据
        //后面具体调用重采样时，输入长度不能超过这个值
        T_U32 maxinputlen;
    }m_resample;
    struct{
        T_U32 AGClevel;  // make sure AGClevel < 32767
        /* used in AGC_1 */
        T_U32  max_noise;
        T_U32  min_noise;
        /* used in AGC_2 */
        T_U8  noiseReduceDis;  // 是否屏蔽自带的降噪功能
        T_U8  agcDis;  // 是否屏蔽自带的 AGC 功能
        T_U8  agcPostEna;
        T_U16 maxGain;  // 最大放大倍数
        T_U16 minGain;  // 最小放大倍数
        T_U32 notch_radius;  //(T_U16)(0.x*(1<<15))
        T_U32 nr_range; // 1~300,越低降噪效果越明显
    }m_agc;
    struct{
        T_U32 ASLC_ena;  // 0:disable aslc;  1:enable aslc
        T_U32 NR_Level;  //  0 ~ 4，越大，降噪越狠
    }m_NR;
}m_Private;
```

}T_AUDIO_FILTER_IN_INFO;

结构名	T_AUDIO_FILTER_IN_INFO	
定义概述	音效参数结构体	
成员说明	m_Type	滤波处理类型,可设置为以下之一： _SD_FILTER_UNKNOWN , _SD_FILTER_EQ , _SD_FILTER_WSOLA , _SD_FILTER_RESAMPLE, _SD_FILTER_3DSOUND, _SD_FILTER_DENOICE, _SD_FILTER_AGC
	m_SampleRate	采样率
	m_Channels	声道数
	m_BitsPerSample	每个 sample 位数
	eqmode	Eq 模式： _SD_EQ_MODE_NORMAL _SD_EQ_MODE_CLASSIC _SD_EQ_MODE_JAZZ , _SD_EQ_MODE_POP , _SD_EQ_MODE_ROCK , _SD_EQ_MODE_EXBASS , _SD_EQ_MODE_SOFT , _SD_EQ_USER_DEFINE
	m_FilterNum m_g	当 EQ 模式为用户自定义模式，且使用老的一套 EQ 算法时有效： m_FilterNum：表示第几个滤波器，0－4 个滤波器 m_g ：EQ 增益，db 范围：-12~12

结构名	T_AUDIO_FILTER_IN_INFO	
	rectification	对新的 EQ 算法有效，用来记录 EQ 各个频段最大放大音量，其值是 0 ~ 1024
	Bands Bandfreqs Bandgains bandQ	当 EQ 模式为用户自定义模式时，且使用新的一套 EQ 算法时有效： Bands: 表示总共有多少个滤波器，可以不设置 Bandfreqs：记录每个滤波器的中心频率 Bandgains：记录每个滤波器的增益 bandQ: 记录每个滤波器的 Q 值 注意： 1. bandQ 赋值形式为 (T_U16)(x.xxx*(1<<10)) 2. bandQ 如果设置为 0，则采用库内部的默认值为 (T_U16)(1.22*(1<<10)) 3. x.xxx < 采样率/(2*该频带的中心频点), 并且 x.xxx 值必须小于 64.000
	Tempo	算法 0 的时候，级数为 0~15，5 为正常速度，0 为最慢级速（0.5 倍速），15 为最快级速（2 倍速）； 算法 1 的时候，级数为 0~10，5 为正常速度，0 为最慢级速(0.5 倍速)，10 为最快级速（2 倍速）
	arithmeticChoice	变速算法选择 0：算法 0，wsola 变速效果差，运算量小 1：算法 1，pjwsola 变速效果好，运算量大
	is3DSurround	是否使能 3D 环绕立体声功能
	outSrindex	重采样输出目标采样率 1:48k 2:44k 3:32k 4:24K 5:22K 6:16K 7:12K 8:11K 9:8K
	maxinputlen;	用于设置重采样时最大输入长度(bytes)，open 时需要用作动态分配的依据。后面具体调用重采样时，输入长度不能超过这个值
	AGClevel	用于设置 AGC 自动控制控制的幅度，该值不能大于 32767

结构名	T_AUDIO_FILTER_IN_INFO	
	max_noise min_noise	预留
	以下是降噪+AGC 的 AGC2 库使用，这个库目前只在回音消除中调用，平台不直接调用	
	noiseReduceDis	表示是否屏蔽 AGC2 自带的降噪功能： 1-屏蔽降噪；0-打开降噪
	agcDis	表示是否屏蔽 AGC2 自带的自动增益控制(AGC)功能 1-屏蔽 AGC；0-打开 AGC
	agcPostEna	在 agcDis==0 的情况下，设置是否真正的 AGC2 库里面做 AGC： 0：表示真正在库里面做 agc，即 filter_control 出来的数据是已经做好 agc 的； 1: 表示库里面只要计算 agc 的 gain 值，不需要真正做 agc 处理；真正的 agc 由外面的调用者后续处理
	maxGain	设置 AGC2 的最大增益值，例如设置 3
	minGain	设置 AGC2 的最小增益值，例如设置 1
	notch_radius	设置 AGC2 的低频滤除系数，如： (T_U16)(0.9*(1<<15))
	nr_range	1~300,越低 AGC2 降噪效果越明显，例如设置为 15
	以下是纯降噪库使用的接口	
	ASLC_ena	降噪时选择要不要做 ASLC。 0:不做 ASLC; 1:做 ASLC
	NR_Level	降噪级别，可设置为 0,1,2,3,4；缺省值为 2

3.4.3 T_AUDIO_FILTER_CB_FUNS

typedef struct

{

MEDIALIB_CALLBACK_FUN_MALLOC Malloc;

MEDIALIB_CALLBACK_FUN_FREE Free;

MEDIALIB_CALLBACK_FUN_PRINTF printf;

MEDIALIB_CALLBACK_FUN_RTC_DELAY delay;

}T_AUDIO_FILTER_CB_FUNS;

结构名	T_AUDIO_FILTER_CB_FUNS	
定义概述	回调函数	
成员说明	MEDIALIB_CALLBACK_FUN_MALLOC	分配内存回调函数
	MEDIALIB_CALLBACK_FUN_FREE	释放内存回调函数
	MEDIALIB_CALLBACK_FUN_PRINTF	打印函数
	MEDIALIB_CALLBACK_FUN_RTC_DELAY	延时函数

3.4.4 T_AUDIO_FILTER_BUF_STRC

typedef struct

{

T_VOID *buf_in;

T_U32 len_in;

T_VOID *buf_out;

T_U32 len_out;

T_VOID *buf_in2; //预留给混音用的，该功能目前还不可用

T_U32 len_in2;

}T_AUDIO_FILTER_BUF_STRC;

结构名	T_AUDIO_FILTER_BUF_STRC	
定义概述	音效处理输入输出 buffer 信息	
成员说明	buf_in	输入 buffer 指针
	len_in	输入 buffer 有效数据大小
	buf_out	输出 buffer 指针

结构名	T_AUDIO_FILTER_BUF_STRC	
	len_out	输出 buffer 可用空间大小
	buf_in2	预留给混音用的，第二路输入数据 buffer 指针。 由于混音功能目前还没投入使用，所以该成员是预留用的。
	len_in2	预留给混音用的，第二路输入数据长度

4 对外接口

4.1 解码缓冲区控制接口

注意：

_SD_Buffer_GetAddr 和_SD_Buffer_UpdateAddr 是配套使用的；

_SD_Buffer_Check 和 _SD_Buffer_Update 是配套使用的

4.1.1 _SD_Buffer_Check

原　　型	T_AUDIO_BUF_STATE _SD_Buffer_Check(T_VOID *audio_decode, T_AUDIO_BUFFER_CONTROL *buffer_control)	
功能概述	判断缓冲区状态	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
	buffer_control	缓冲区控制的结构
返回值	缓冲区状态	
返回值说明	详见 T_AUDIO_BUF_STATE 结构说明	
注意事项		
调用示例	见典型调用示例	

4.1.2 _SD_Buffer_Update

原　　型	T_S32 _SD_Buffer_Update(T_VOID *audio_decode, T_U32 len)	
功能概述	更新缓冲区状态	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
	len	上层向缓冲区填写数据长度
更新	更新是否成功	
返回值说明	AK_TRUE　　更新成功 AK_FALSE　　更新失败	
注意事项		
调用示例	见典型调用示例	

4.1.3 _SD_Buffer_Clear

原　　型	T_S32 _SD_Buffer_Clear(T_VOID *audio_decode)	
功能概述	清除缓冲区数据	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
返回值	清除是否成功	
返回值说明	AK_TRUE　　清除成功 AK_FALSE　　清除失败	
注意事项		
调用示例	例如做 seek 操作后，要对音频解码用的结构体 T_AUDIO_DECODE_STRUCT 内的一些成员进行复位操作。	

4.1.4 _SD_Buffer_GetAddr

原　　型	T_VOID* _SD_Buffer_GetAddr(T_VOID *audio_decode, T_U32 len)	
功能概述	获取音频播放内部缓冲区的地址指针	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
	len	一次写入 buffer 的数据长度
返回值	返回 buffer 的地址指针	

原　　型	T_VOID* _SD_Buffer_GetAddr(T_VOID *audio_decode, T_U32 len)
返回值说明	返回值非空，则或得了缓冲区的地址指针
注意事项	
调用示例	见典型调用示例

4.1.5 _SD_Buffer_UpdateAddr

原　　型	T_S32 _SD_Buffer_UpdateAddr(T_VOID *audio_decode, T_U32 len)	
功能概述	更新音频播放内部缓冲区写指针	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
	len	需要更新 buffer 的大小
返回值	是否更新成功	
返回值说明	AK_TRUE : 更新成功 AK_FLASE : 更新失败	
注意事项		
调用示例	见典型调用示例	

4.2 解码接口

4.2.1 _SD_Decode_Open

原　　型	T_VOID *_SD_Decode_Open(T_AUDIO_DECODE_INPUT *audio_input, T_AUDIO_DECODE_OUT *audio_output)	
功能概述	打开音频解码库，申请解码所需内存等	
参数说明	audio_input	需要解码输入的信息
	audio_output	解码输出的信息
返回值	音频解码内部保留结构指针	
返回值说明	AK_NULL　　打开失败	
注意事项	输入和输出信息详见结构说明	

原　　型	T_VOID *_SD_Decode_Open(T_AUDIO_DECODE_INPUT *audio_input, T_AUDIO_DECODE_OUT *audio_output)
调用示例	见典型调用示例

4.2.2 _SD_Decode

原　　型	T_S32 _SD_Decode(T_VOID *audio_decode, T_AUDIO_DECODE_OUT *audio_output)	
功能概述	音频解码	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
	audio_output	解码输出的信息
返回值	解码数据长度	
返回值说明	解码长度，以 byte 为单位；0 或负数时，表示解码失败	

注意事项	注意： 每次调用改接口，都要设置要 audio_output 结构体成员 m_ulSize 赋值，例如：
	audio_output.m_pBuffer = obuf;
	audio_output.m_ulSize = OUTBUF_LEN;	

调用示例	见典型调用示例	

4.2.3 _SD_Decode_Close

原　　型	T_S32 _SD_Decode_Close(T_VOID *audio_decode)	
功能概述	关闭音频解码库，释放解码所需内存等	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
返回值	关闭是否成功	
返回值说明	AK_TRUE　　关闭成功 AK_FALSE　　关闭失败	
注意事项		
调用示例	见典型调用示例	

4.3 录音及编码接口

4.3.1 _SD_Encode_Open

原　　型	T_VOID *_SD_Encode_Open(T_AUDIO_REC_INPUT *enc_input, T_AUDIO_ENC_OUT_INFO *enc_output)	
功能概述	打开音频编码库，分配编码所需的内存等	
参数说明	enc_input	需要录音输入的信息
	enc_output	编码输出信息。
返回值	音频编码内部保留结构指针	
返回值说明	AK_NULL　　打开失败	
注意事项	输入信息详见结构说明	
调用示例	见典型调用示例	

4.3.2 _SD_Encode

原　　型	T_S32 _SD_Encode(T_VOID *audio_encode, T_AUDIO_ENC_BUF_STRC *enc_buf_strc)	
功能概述	音频编码	
参数说明	audio_encode	编码参数的保留结构，在 open 时获得的返回指针
	enc_buf_strc	编码输入输出 buffer 信息
返回值	编码数据长度	
返回值说明	编码长度，以 byte 为单位	
注意事项		
调用示例	见典型调用示例	

4.3.3 _SD_Encode_Close

原　　型	T_S32 _SD_Encode_Close(T_VOID *audio_record)	
功能概述	关闭音频编码库，释放内存等	
参数说明	audio_decode	录音参数的保留结构，在 open 时获得的返回指针
返回值	关闭是否成功	

原　　型	T_S32 _SD_Encode_Close(T_VOID *audio_record)
返回值说明	AK_TRUE　　关闭成功 AK_FALSE　　关闭失败
注意事项	
调用示例	见典型调用示例

4.3.4 _SD_Encode_Last

原　　型	T_S32 _SD_Encode_Last(T_VOID *audio_encode,T_AUDIO_ENC_BUF_STRC *enc_buf_strc)	
功能概述	音频编码	
参数说明	audio_encode	编码参数的保留结构，在 open 时获得的返回指针
	enc_buf_strc	编码输入输出 buffer 信息
返回值	编码数据长度	
返回值说明	编码长度，以 byte 为单位	
注意事项	这个接口是在编码结束、CLOSE 之前调用，给那些需要写结束标志到编码码流中，的录音格式用的。 目前这个接口，只有 speex 编码才需要用。	
调用示例	见典型调用示例	

4.3.5 _SD_Encode_SetFramHeadFlag

原　　型	T_S32 _SD_Encode_SetFramHeadFlag(T_VOID *audio_encode, int flag)	
功能概述	音频（AAC）编码时设置是否需要音频库返回编码的帧头 7 字节数据	
参数说明	audio_encode	编码参数的保留结构，在 open 时获得的返回指针
	flag	表示是否需要音频库返回帧头数据 _SD_ENC_SAVE_FRAME_HEAD：需要返回帧头数据 _SD_ENC_CUT_FRAME_HEAD：不需要返回帧头数据
返回值	表示是否设置成功	

原　　型	T_S32 _SD_Encode_SetFramHeadFlag(T_VOID *audio_encode, int flag)
返回值说明	AK_TRUE 设置成功 AK_FALSE 设置失败
注意事项	这个接口只在 AAC 编码的时候用的
调用示例	

4.4 DA buffer控制接口

注：这套接口主要是给 spotlight 媒体库调用，平台一般不需要使用。

4.4.1 T_S32 DABuf_Init(T_DABUF_INIT *buf_init)

原　　型	T_S32 DABuf_Init(T_DABUF_INIT *buf_init)	
功能概述	创建 DA buffer	
参数说明	buf_init	创建时，需要输入的信息结构指针
返回值	创建是否成功	
返回值说明	AK_TRUE　　创建成功 AK_FALSE　　创建失败	
注意事项	注意：要传入回调函数的指针	
调用示例	播放器初始化的时候调用	

4.4.2 T_S32 DABuf_Destroy(T_VOID)

原　　型	T_S32 DABuf_Destroy(T_VOID)
功能概述	销毁 DA buffer
返回值	销毁是否成功
返回值说明	AK_TRUE　　销毁成功 AK_FALSE　　销毁失败
注意事项	
调用示例	退出播放器的时候调用

4.4.3 T_S32 DABuf_Clear(T_VOID)

原　　型	T_S32 DABuf_Clear(T_VOID)
功能概述	清空 DA buffer
返回值	清空是否成功
返回值说明	AK_TRUE　　清空成功 AK_FALSE　　清空失败
注意事项	
调用示例	例如 seek 操作后调用

4.4.4 T_S32 DABuf_SetVolume(T_U32 volume)

原　　型	T_S32 DABuf_SetVolume(T_U32 volume)	
功能概述	设置播放音量大小	
参数说明	volume	输入声音大小
返回值	设置是否成功	
返回值说明	AK_TRUE　　设置成功 AK_FALSE　　设置失败	
注意事项		
调用示例	见典型调用示例	

4.4.5 T_S32 DABuf_SendData(T_VOID *pbuf, T_U32 len)

原　　型	T_S32 DABuf_SendData(T_VOID *pbuf, T_U32 len)	
功能概述	发送数据到 DA buffer	
参数说明	pbuf	要写到 DA buffer 的数据指针
	len	要写到 DA buffer 的数据长度
返回值	发送是否成功	
返回值说明	AK_TRUE　　发送成功 AK_FALSE　　发送失败	
注意事项		

原　　型	T_S32 DABuf_SendData(T_VOID *pbuf, T_U32 len)	
调用示例	见典型调用示例	

4.4.6 T_S32 DABuf_GetData(T_VOID **pbuf, T_U32 *len)

原　　型	T_S32 DABuf_GetData(T_VOID **pbuf, T_U32 *len)	
功能概述	从 DA buffer 获取要播放的数据	
参数说明	pbuf	获取的数据指针
	len	获取的数据长度
返回值	获取是否成功	
返回值说明	AK_TRUE　　获取成功 AK_FALSE　　获取失败	
注意事项		
调用示例	例如要送数据给 DACbuffer 的地方	

4.4.7 T_S32 DABuf_SetSta(T_eDA_STA_TYPE sta_type)

原　　型	T_S32 DABuf_SetSta(T_eDA_STA_TYPE sta_type)	
功能概述	设置指定的状态类型的状态	
参数说明	sta_type	要设置的状态类型
返回值	设置是否成功	
返回值说明	AK_TRUE　　关闭成功 AK_FALSE　　关闭失败	
注意事项		
调用示例	见典型调用示例	

4.4.8 T_S32 DABuf_GetSta(T_eDA_STA_TYPE sta_type)

原　　型	T_S32 DABuf_GetSta(T_eDA_STA_TYPE sta_type)	
功能概述	获取指定的状态类型的状态	
参数说明	sta_type	要获取的状态类型
返回值	获取的状态	
返回值说明		
注意事项		
调用示例	见典型调用示例	

4.5 音效处理接口

4.5.1 T_VOID *_SD_Filter_Open (const T_AUDIO_FILTER_INPUT *filter_input)

原　　型	T_VOID *_SD_Filter_Open(const T_AUDIO_FILTER_INPUT *filter_input)	
功能概述	打开音效处理设备	
参数说明	filter_input	音效处理的输入结构
返回值	T_VOID *	
返回值说明	返回音效处理结构体指针，返回空表示失败	
注意事项		
调用示例	见典型调用示例	

4.5.2 T_S32 _SD_Filter_Control(T_VOID *audio_filter, T_AUDIO_FILTER_BUF_STRC *audio_filter_buf)

原　　型	T_S32 _SD_Filter_Control(T_VOID *audio_filter, T_AUDIO_FILTER_BUF_STRC *audio_filter_buf)	
功能概述	音效处理	
参数说明	audio_filter	_SD_Filter_Open 调用时返回的结构体指针。
	audio_filter_buf	音效处理输入输出 buffer 结构体
返回值	T_S32	

原　　型	T_S32 _SD_Filter_Control(T_VOID *audio_filter, T_AUDIO_FILTER_BUF_STRC *audio_filter_buf)
返回值说明	一次调用返回的音效处理后的数据长度，字节为单位
注意事项	
调用示例	见典型调用示例

4.5.3　T_S32 _SD_Filter_Close(T_VOID *audio_filter)

原　　型	T_S32 _SD_Filter_Close(T_VOID *audio_filter)	
功能概述	关闭音效处理设备	
参数说明	audio_filter	_SD_Filter_Open 调用时返回的结构体指针。
返回值	T_S32	
返回值说明	返回 0，关闭失败；返回 1，关闭成功	
注意事项		
调用示例	见典型调用示例	

4.5.4 T_S32 _SD_Filter_SetParam(T_VOID *audio_filter, T_AUDIO_FILTER_IN_INFO *info)

原　　型	T_S32 _SD_Filter_SetParam(T_VOID *audio_filter, T_AUDIO_FILTER_IN_INFO *info)	
功能概述	设置音效参数	
参数说明	audio_filter	_SD_Filter_Open 调用时返回的结构体指针。
	info	音效参数结构体
返回值	T_S32	
返回值说明	返回 0，关闭失败；返回 1，关闭成功	
注意事项		
调用示例	例如 EQ 的时候，要改变 eq 模式	

4.6 单解码器对外接口

前面提到，音频库支持多种格式的解码，视用户需求，提供的库都能同时支持多种音频格式的解码，音频库提供的功能较多，导致层次较复杂，代码有一定的冗余。

所谓单解码器接口，是专门用于 AK10XX 芯片上的。由于 AK10XX 芯片内存限制，系统实现音频播放时，往往需要使用内存交换技术，将片内内存与闪存进行交换。有时候用户不想使用内存交换技术，只实现一种格式的解码即可。这样就需要只实现一种经过精简，空间需求尽可能小的解码器。目前只实现了两个单解码器库，即 MP3 和 AMR。这两种格式可在 AK10XX 芯片上，不通过内存交换技术，也可以顺利实现解码。

为此种解码模式，专门增加了一个结构体，用于在打开解码设备时获取相关的音频信息，如采样率等。

```
typedef struct
    {
        T_S32 samplerate;
        T_S32 bitrate;
        T_S32 totaltime;
        T_S32 channels;
        T_U32 CBRflag;
        T_U32 audiolen;   //audio file length (bytes)
        T_U32 header;     //valid mp3 header saved for seek
    }T_AudioInfo;
```


结构名	T_AudioInfo	
定义概述	需要解码音频的信息	
成员说明	Samplerate	采样率
	bitrate	比特率/码率
	totaltime	歌曲总时间
	Channels	声道数
	CBRflag	是否 CBR 歌曲
	Audiolen	音频文件大小
	Header	有效的 mp3 头，用于 seek 操作

4.6.1 T_VOID *AudioLib_Open()

原　　型	T_VOID *AudioLib_Open(const T_AUDIO_DECODE_INPUT *audio_input, T_AUDIO_DECODE_OUT *audio_output,T_AudioInfo *audioinfo)	
功能概述	打开音频播放设备	
参数说明	audio_input	需要解码输入的信息
	audio_output	解码输出的信息
	Audioinfo	打开解码器同时获取媒体文件基本信息。暂时该信息对 mp3 解码器有效，对 AMR 无效，因 AMR 的基本信息是已经的（单声道，8000 采样率等）。
返回值	音频解码内部保留结构指针	
返回值说明	AK_NULL　　打开失败	
注意事项	输入和输出信息详见结构说明	
调用示例	见典型调用示例	

4.6.2 T_S32 AudioLib_Decode()

原　　型	T_S32 AudioLib_Decode(T_VOID *audio_decode, T_AUDIO_DECODE_OUT *AudioDecOut)	
功能概述	音频解码	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
	audio_output	解码输出的信息
返回值	解码数据长度	
返回值说明	解码长度，以 byte 为单位；负数时，表示解码失败或解码结束	
注意事项		
调用示例	见典型调用示例	

4.6.3 T_S32 AudioLib_Close()

原　　型	T_S32 AudioLib_Close (T_VOID *audio_decode)	
功能概述	关闭音频解码设备	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
返回值	关闭是否成功	
返回值说明	AK_TRUE　　关闭成功 AK_FALSE　　关闭失败	
注意事项		
调用示例	见典型调用示例	

4.6.4 T_BOOL AudioLib_DecSeek()

原　　型	T_BOOL AudioLib_DecSeek(T_VOID *audio_decode,T_U32 time_ms, T_AudioInfo *audioinfo)	
功能概述	Seek，以便进行定位播放或快进快退	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
	time_ms	需要定位播放的时间（即在音乐中什么时间点播放）
	audioinfo	做 seek 需要的一些信息。
返回值	关闭是否成功	
返回值说明	AK_TRUE　　SEEK 成功 AK_FALSE　　SEEK 失败	
注意事项	该接口只对 mp3 有效，对 AMR 无效。AMR 不能进行 seek	
调用示例		

4.7 快速响应的播放器打开接口

本套接口是为类似点读笔这样的特殊需求设计的，它不需要获取总时间，但是要求文件切换的响应速度要快。歌曲切换的时候可以不需要进行 close、open 的操作，但要**调用者保证切换的歌曲间的采样率、声道等信息的一致性**。具体接口的调用方法可以参考 5.7 节的接口调用示例

以下两个接口用来替换 4.2 中的 open 接口。4.2 节中的 open 接口，需要与媒体库的 demuxer 配合才能使用；这里的两个接口不需要媒体库的 demuxer，但是同时也获取不到总时间。

调用这两个接口之后，后续的音频播放就可以调用 4.2 中的 decode 和 close 接口。

4.7.1 T_VOID *_SD_Decode_Open_Fast(T_AUDIO_DECODE_INPUT *audio_input, T_AUDIO_DECODE_OUT *audio_output)

原　　型	T_VOID *_SD_Decode_Open_Fast(T_AUDIO_DECODE_INPUT *audio_input, T_AUDIO_DECODE_OUT *audio_output)	
功能概述	快速打开音频解码库，申请解码所需内存等。使用这个接口的时候，不需要依赖媒体库的 demuxer。平台可以直接在打开文件后，将文件开始的部分数据存储在 m_szData 指向的地址空间中，然后调用这个接口。	
参数说明	audio_input	需要解码输入的信息
	audio_output	解码输出的信息
返回值	音频解码内部保留结构指针	
返回值说明	AK_NULL　　打开失败	
注意事项	目前这个接口只适用与 ogg vorbis 播放，其它文件格式的还不支持。 调用这个接口之前，要预先读取文件开始的一段数据到一个 buffer 中，并将 m_szData 指向这个 buffer，及将数据长度值赋给 m_szDataLen。 读取的数据长度，要求如下： 对于 vorbis 格式，**至少 58 字节**；如果不够 58 字节，返回 AK_NULL。	
调用示例	见典型调用示例 第一次播放一种类型的歌曲时，快速打开解码设备。	

4.7.2 T_S32 _SD_Decode_ParseFHead(T_VOID *audio_decode)

原　　型	T_S32 _SD_Decode_ParseFHead(T_VOID *audio_decode)	
功能概述	音频解码之前，获取解码需要的一些数据	
参数说明	audio_decode	解码参数的保留结构，在 open 时获得的返回指针
返回值	标识函数执行成功与否	
返回值说明	>0 : 成功 <0: 失败 =0: 输入数据不够，等待继续填数据；调用者再次填了数据后，再次调用改接口会继续解码。	
注意事项	注意：调用这个接口之前要通过音频库的码流填充接口（见 4.1 节），给输入 buffer 填入足够的数据。	
调用示例	_SD_Decode_Open_Fast() 之后，_SD_Decode()之前，获取解码过程需要的一些数据。 或者在切换歌曲时，保证歌曲格式及采样率、声道等属性，与之前播放的歌曲一致时，直接调用这个接口，不需要进行 close、及再次 open。 具体使用方法，见典型调用示例	

4.8 其它接口

4.8.1 T_S8 *_SD_GetAudioCodecVersionInfo()

原　　型	T_S8 *_SD_GetAudioCodecVersionInfo(T_VOID)	
功能概述	获取当前音频库版本号	
参数说明	无	
返回值	T_S8*	
返回值说明	返回字符号，例如： AudioCodec Version V0.7.0	
注意事项		
调用示例		

4.8.2 T_S32 _SD_GetAudioSpectrum()

原　　型	T_S32 _SD_GetAudioSpectrum(T_S32 *data,T_U16 size, T_AUDIO_CB_FUNS *cbfun)	
功能概述	对传进来的时域音频 PCM 信号, 计算其频谱并原址返回. 该接口会调用 wma 解码器中的程序模块,所以在 WMA 解码模块开启时才能使用	
参数说明	data	输入时是 T_S32 的时域音频 PCM 数据，pcm 数据放在低位； 输出时是 T_S32 的频谱数据
	size	时域音频 PCM 数据的长度（样点数，不是字节数）
	cbfun	回调函数结构体,需要将 malloc,free 和 printf 传进来
返回值	T_S32	
返回值说明	AK_TRUE　　计算频域数据成功,频域数据在 data 中, 返回的数据样点数是 size/2，并且是对称的 AK_FALSE　　由于内存分配失败而回 FALSE	
注意事项		
调用示例		

4.8.3 T_S32 _SD_GetAudioSpectrum_euqNum()

原　　型	T_S32 _SD_GetAudioSpectrum_equNum(T_S32 *data, T_U16 size, T_AUDIO_CB_FUNS *cbfun)	
功能概述	对传进来的时域音频 PCM 信号, 计算其频谱并原址返回. 该接口会调用 wma 解码器中的程序模块,所以在 WMA 解码模块开启时才能使用	
参数说明	data	输入时是 T_S32 的时域音频 PCM 数据，pcm 数据放在低位； 输出时是 T_S32 的频谱数据
	size	时域音频 PCM 数据的长度（样点数，不是字节数）
	cbfun	回调函数结构体,需要将 malloc,free 和 printf 传进来
返回值	T_S32	

原　　型	T_S32 _SD_GetAudioSpectrum_equNum(T_S32 *data, T_U16 size, T_AUDIO_CB_FUNS *cbfun)
返回值说明	AK_TRUE　　计算频域数据成功,频域数据在 data 中, 返回的数据样点数是 size，并且是对称的 AK_FALSE　　由于内存分配失败而回 FALSE
注意事项	
调用示例	

4.8.4 T_S32 _SD_GetAudioSpectrumComplex()

原　　型	T_S32 _SD_GetAudioSpectrumComplex(T_S32 *data, T_U16 size, T_AUDIO_CB_FUNS *cbfun)	
功能概述	对传进来的时域音频 PCM 信号, 计算其频谱并原址返回. 该接口会调用 wma 解码器中的程序模块,所以在 WMA 解码模块开启时才能使用	
参数说明	data	输人输出数据，都是复数（实部和虚部都是 T_S32），并且都是实部、虚部、实部、虚部……这样的顺序排列 输入时是 T_S32 的时域音频 PCM 数据，pcm 数据放在实部的低位，虚部是 0。
	size	时域音频 PCM 数据的长度（复数样点数，不是字节数）
	cbfun	回调函数结构体,需要将 malloc,free 和 printf 传进来
返回值	T_S32	
返回值说明	AK_TRUE　　计算频域数据成功,频域数据在 data 中, 返回的数据样点数是 size，并且是对称的 AK_FALSE　　由于内存分配失败而回 FALSE	
注意事项		
调用示例		

4.8.5 T_S32 _SD_GetCodecTime()

原　　型	T_S32 _SD_GetCodecTime(T_VOID *audio_codec, T_U8 codec_flag)	
功能概述	获取编解码时间（非播放时间）	
参数说明	audio_codec	解码参数的保留结构，在 open 时获得的返回指针
	Codec_flag	编解码标志 0：解码 1：编码
返回值	T_S32	
返回值说明	获取的时间	
注意事项	目前 codec_flag 只能为 0，即只能获取解码时间，不能获取编码时间 注意：获取到的是解码时间，而不是播放时间	
调用示例		

4.8.6 T_S32 _SD_GetWMABitrateType()

原　　型	T_S32 _SD_GetWMABitrateType(T_VOID *audio_decode)	
功能概述	获取 wma 比特率类型，LPC/Mid/High rate 三种	
参数说明	audio_codec	解码参数的保留结构，在 open 时获得的返回指针
返回值	T_S32	
返回值说明	返回比特率类型，0/1/2 分别对应 LPC/Mid/High rate	
注意事项		
调用示例		

4.8.7 T_VOID _SD_LogBufferSave()

原　　型	T_VOID _SD_LogBufferSave(T_U8 *pBuf, T_U32 *len,T_VOID *audio_codec)	
功能概述	获取当前解码缓冲中的码流供分析，暂时只对 AAC 解码有效	
参数说明	pBuf	存储码流的缓冲
	len	pBuf 缓冲中数据的长度
	audio_codec	解码参数的保留结构，在 open 时获得的返回指针
返回值	T_VOID	

原　　型	T_VOID _SD_LogBufferSave(T_U8 *pBuf, T_U32 *len,T_VOID *audio_codec)
返回值说明	
注意事项	
调用示例	

4.8.8 T_S32 _SD_SetInbufMinLen()

原　　型	T_U32 _SD_SetInbufMinLen(T_VOID *audio_decode, T_U32 len)	
功能概述	设置解码缓冲最少输入多少数据，才开始解码。 备注：这个接口只对非 spotlihgt 系列平台适用。	
参数说明	audio_codec	解码参数的保留结构，在 open 时获得的返回指针
	Len	要设置的长度。 当解码缓冲的工作模式为_SD_BM_NORMAL 时，解码判断到输入 buffer 中剩余数据为 len 长度才进行解码，否则返回 0。 如果退出的时候工作模式还是为_SD_BM_NORMAL，那么会有 len 长的码流丢弃，不能解码。 设置的长度，不能超过 buffer 总长。
返回值	T_U32	
返回值说明	实际设置的 inbufminlen 长度	
注意事项	这个接口必须在_SD_Decode_Open()或_SD_Decode_Open_Fast()之后调用，才能生效	
调用示例	应用场景：对 flash 播放等不能保证输入能填满 buffer，也不知道什么时候能设置 buffer 工作模式为_SD_BM_ENDING，又不想播放结束的时候丢失太多声音，那么就可以自己设置这个最小输入数据缓冲长度。 如果用户不调用这个接口设置，音频库内部会设置默认值，这个默认值比较大。退出的时候工作模式如果不能设置为_SD_BM_ENDING 的话，可能导致丢失几秒声音。	

4.8.9 T_S32 _SD_SetBufferMode()

原　　型	T_S32 _SD_SetBufferMode(T_VOID *audio_decode, T_AUDIO_BUFFER_MODE buf_mode)	
功能概述	设置解码缓冲工作模式. 备注：这个接口只对非 spotlihgt 系列平台适用。	
参数说明	audio_codec	解码参数的保留结构，在 open 时获得的返回指针
	buf_mode	输入 buffer 工作模式 bufmode = _SD_BM_NORMAL，缓冲解码，解码器会要求输入 buffer 中缓冲有一定长度的码流后才进行解码 bufmode = _SD_BM_LIVE，即时解码，不要求输入 buffer 中有多少码流，被调用即无条件解码，如果解码过程中发现码流不足，解码失败。 注意：如果设置为 Normal 模式，在解码结束时（即上层确定码流已经完成，但音频输入缓冲中仍有未解完的码流）,需要将 bufmode 设置为_SD_BM_ENDING，才能将音频输入 buffer 中最后剩余的码流解码。
返回值	T_S32	
返回值说明	无	
注意事项		
调用示例		

4.8.10 T_VOID _SD_SetHandle()

原　　型	T_VOID _SD_SetHandle(T_VOID *audio_decode, T_VOID *pHandle)	
功能概述	设置解码句柄，将其传给解码器，以便在调用回调使用	
参数说明	audio_codec	解码参数的保留结构，在 open 时获得的返回指针
	pHandle	传入的句柄
返回值	T_S32	
返回值说明	返回音频库内部解码结构的指针，空表示失败	

原　　型	T_VOID _SD_SetHandle(T_VOID *audio_decode, T_VOID *pHandle)
注意事项	
调用示例	

4.8.11 T_S32 _SD_Decode_SetDigVolume()

原　　型	T_S32 _SD_Decode_SetDigVolume(T_VOID *audio_decode, T_U32 volume)	
功能概述	设置解码出来的声音的音量。 适用环境：解码不需调用媒体库，而平台又没有相应的音量处理功能时。例如蓝牙立体声播放的时候。	
参数说明	audio_codec	解码参数的保留结构，在 open 时获得的返回指针
	volume	目标音量值：1024 为标准解码输出音量，小于这个值缩小原始音量，大于这个值放大原始音量。
返回值	T_S32	
返回值说明	AK_TRUE：音量设置成功 AK_FALSE：音量设置失败	
注意事项	注意：目前这个接口只对有特殊要求的平台开放，例如蓝牙立体声平台。	
调用示例		

4.8.12 T_S32 _SD_Decode_Seek()

原　　型	T_S32 _SD_Decode_Seek(T_VOID *audio_decode, T_AUDIO_SEEK_INFO *seek_info)	
功能概述	音频 seek	
参数说明	audio_codec	解码参数的保留结构，在 open 时获得的返回指针
	seek_info	Seek 后需要传给音频库的一些信息，具体见 T_AUDIO_SEEK_INFO 这个结构体的说明
返回值	T_S32	

原　　型	T_S32 _SD_Decode_Seek(T_VOID *audio_decode, T_AUDIO_SEEK_INFO *seek_info)
返回值说明	AK_TRUE：seek 成功 AK_FALSE：seek 失败
注意事项	
调用示例	

4.8.13 T_S32 _SD_Get_Input_Buf_Info()

原　　型	T_S32 _SD_Get_Input_Buf_Info(T_VOID *audio_decode, T_AUDIO_INBUF_STATE type)	
功能概述	获取音频库输入 buf 相关信息。	
参数说明	audio_codec	解码参数的保留结构，在 open 时获得的返回指针
	type	表示要获取什么数据，可以为以下类型： _STREAM_BUF_LEN，获取输入 buf 的总长度； _STREAM_BUF_REMAIN_DATAk，获取 buf 中剩余未解码数据的长度； _STREAM_BUF_MIN_LEN，获取解码所需最小 buf 长度，即 inbufminlen。
返回值	T_S32	
返回值说明	大于或等于 0：获取到的长度信息 小于 0：传入的参数非法	
注意事项		

调用示例	获取剩余数据长度：
	len = _SD_Get_Input_Buf_Info(paudio, _STREAM_BUF_REMAIN_DATA);
	获取 buffer 总长度：
	len = _SD_Get_Input_Buf_Info(paudio, _STREAM_BUF_LEN);
	获取 inbufminlen 长度：
	len = _SD_Get_Input_Buf_Info(paudio, _STREAM_BUF_MIN_LEN);	

5 典型调用示例

5.1 解码接口调用示例

以下调用示例适用于 4.2 中的解码接口。

```
#define OUTBUF_LEN 2048
Main()
{
   T_AUDIO_DECODE_INPUT audio_input;
   T_AUDIO_DECODE_OUT audio_output;
   T_VOID *pAudioCodec;
   char *obuf;
   FILE *fp_in = NULL;
   FILE *fp_out = NULL;

   fp_in = fopen("c:\\test.mp3","rb");
   fp_out = fopen("c:\\test.pcm","wb");

   audio_input.cb_fun.Malloc = (MEDIALIB_CALLBACK_FUN_MALLOC)malloc;
   audio_input.cb_fun.Free = (MEDIALIB_CALLBACK_FUN_FREE)free;
   audio_input.cb_fun.printf = (MEDIALIB_CALLBACK_FUN_PRINTF)printf;
   audio_input.cb_fun.delay = AK_NULL;
   audio_input.m_info.m_BitsPerSample = 16;
   audio_input.m_info.m_Channels = 2;
   audio_input.m_info.m_SampleRate = 44100;
   audio_input.m_info.m_Type = _SD_MEDIA_TYPE_MP3;
   pAudioCodec = _SD_Decode_Open(&audio_input,& audio_output);
   if (AK_NULL  == pAudioCodec)
   {
      return;
   }
```

```
    obuf = audio_input.cb_fun.Malloc(OUTBUF_LEN);
    if  (AK_NULL  == obuf )
    {
       return;
    }
    while(1)
    {
       T_S32 nSize;
       Fill_Audiodata(fp_in, pAudioCodec);
       audio_output.m_pBuffer = obuf;
       audio_output.m_ulSize = OUTBUF_LEN;
       nSize  = _SD_Decode(pAudioCodec, &audio_output);
       if(nSize < 0)            break;
       else  fwrite(audio_output. m_pBuffer,1,nSize,fp_out);
    }
    Free(obuf);
    obuf = AK_NULL;
   _SD_Decode_Close(pAudioCodec);
   Fclose(fp_in);
   Fclose(fp_out);
}
```

5.2 码流数据填充示例

以下调用示例适用于 4.1 中的解码缓冲区控制接口。

1）方法一

```
    void Fill_Audiodata(FILE *fp, T_VOID *audio_decode)
    {
       T_AUDIO_BUFFER_CONTROL bufctrl;
          T_AUDIO_BUF_STATE bufstate;
          T_S32  read_len;

       bufstate  = _SD_Buffer_Check(audio_decode ,&bufctrl);
```

```
    if(_SD_BUFFER_FULL  ==  bufstate )
    {
       return;
    }
    else if(_SD_BUFFER_WRITABLE == bufstate)
    {
       read_len = fread(bufctrl. pwrite, 1,  bufctrl. free_len,  fp);
       if (read_len>0)
          _SD_Buffer_Update(audio_decode, read_len);
    }
    else
    {
       read_len = fread(bufctrl. pwrite, 1,  bufctrl. free_len,  fp);
       if (read_len>0)
          _SD_Buffer_Update(audio_decode, read_len);
       read_len = fread(bufctrl ->pstart,  1,  bufctrl ->start_len,  fp);
       if (read_len>0)
          _SD_Buffer_Update(audio_decode, read_len);
    }
}
```

2）方法二

```
T_S32 Fill_Audiodata(FILE *fp, T_VOID *audio_decode,T_U32  need_len)
{
   T_U32 *pbuf;
   T_U32  read_len;
   T_AUDIO_DECODE_STRUCT *pdecode = (T_AUDIO_DECODE_STRUCT
*)audio_decode;

   pbuf = _SD_Buffer_GetAddr(pdecode,need_len);
   if (AK_NULL == pbuf)
   {
```

```
            return  AK_FALSE;
        }

        read_len = fread(pbuf, 1,  need_len,  fp);
        if (read_len>0)
            _SD_Buffer_UpdateAddr(pdecode, read_len);
        return      AK_TURE;
    }
```

5.3 单解码器对外接口调用示例

以下调用示例适用于 4.6 的接口调用。

```
int main(int argc, char *argv[])
{
   TCHAR wave_file[MAX_PATH];
   TCHAR ofFileName[MAX_PATH];
   TCHAR outpcm_file[MAX_PATH];
   T_S32 fid;
   T_AudioInfo audioinfo;
   T_AUDIO_DECODE_INPUT audio_input;
   T_AUDIO_DECODE_OUT audio_output;
   T_pVOID pAudioDec;
   T_U8 OutBuf[4096];
   T_S32 nSize;
   FILE *fp_out = NULL;

   strcpy(ofFileName, argv[1]);
   strcpy(outpcm_file, argv[2]);

   fid = _open(ofFileName, _O_RDONLY | _O_BINARY);
   if(fid <= 0)
   {
```

```
    printf("open music file failed\r\n");
    return 0;
}
fp_out = fopen("outpcm_file","w");
if(NULL == fp_out)
{
    printf("open pcm file failed\n");
    return 0;
}

open_input.m_CBFunc.m_FunMalloc = malloc;
open_input.m_CBFunc.m_FunFree = free;
open_input.m_CBFunc.m_FunPrintf = printf;
open_input.m_CBFunc.m_FunRead = _read;
open_input.m_CBFunc.m_FunWrite = _write;
open_input.m_CBFunc.m_FunSeek = _lseek;
open_input.m_CBFunc.m_FunTell = _tell;

audio_input.m_info.haudio = fid;
audio_input.cb_fun.read = _read;
audio_input.cb_fun.seek = _lseek;
audio_input.cb_fun.tell = _tell;

audio_input.cb_fun.Free = free;
audio_input.cb_fun.Malloc = malloc;
audio_input.cb_fun.printf = printf;
audio_input.m_info.m_Type = _SD_MEDIA_TYPE_MP3;

audio_output.m_pBuffer = OutBuf;
pAudioDec = AudioLib_Open(&audio_input, &audio_output,&audioinfo);
if (NULL == pAudioDec)
{
```

```
        printf("##Playback: ERROR--Open audio error##\r\n");
        return 0;
    }

    while( 1 )
    {
        nSize = AudioLib_Decode(pMedialib->pAudioDec, &audio_output);
        if(nSize < 0) //decode finished
            break;
        fwrite(audio_output.m_pBuffer,1,nSize,outpcm_file); //write pcm to file
    }
    AudioLib_Close(pAudioDec);
    fclose(fp_out);
    return 0;
}
```

5.4 音效处理接口调用示例

以下调用示例适用于 4.5 的音效处理接口

```
#define  ONCE_READ_LEN (32*2)
#define  IN_BUF_LEN    (1024*2)
#define  OUT_BUF_LEN   (2048*2)


main()
{
    T_VOID *pfilter;
    T_AUDIO_FILTER_INPUT s_ininfo;
    T_AUDIO_FILTER_BUF_STRC fbuf_strc;
    char *ibuf = AK_NULL;
    char *obuf = AK_NULL;
    FILE *fp_in = AK_NULL;
    FILE *fp_out = AK_NULL;
```

```
fp_in = fopen("c:\\test.pcm","rb");
fp_out = fopen("c:\\test.pcm","wb");

s_ininfo.cb_fun.Malloc = (MEDIALIB_CALLBACK_FUN_MALLOC)malloc;
s_ininfo.cb_fun.Free = (MEDIALIB_CALLBACK_FUN_FREE)free;
s_ininfo.cb_fun.printf = (MEDIALIB_CALLBACK_FUN_PRINTF)printf;
s_ininfo.cb_fun.delay = AK_NULL;
s_ininfo.m_info.m_BitsPerSample = 16;
s_ininfo.m_info.m_Channels = 2;
s_ininfo.m_info.m_SampleRate = 44100;
s_ininfo.m_info.m_Private.m_eq.eqmode = _SD_EQ_MODE_CLASSIC;
s_ininfo.m_info.m_Type = _SD_FILTER_EQ;

pfilter = _SD_Filter_Open(&s_ininfo);
if (AK_NULL == pfilter)
   return;

ibuf = malloc(IN_BUF_LEN);
obuf = malloc(OUT_BUF_LEN);
while(1)
{
   T_S32 nSize, readlen;

   fbuf_strc.buf_in = ibuf;
   fbuf_strc.buf_out = obuf;
   fbuf_strc.len_out = OUT_BUF_LEN;
   readlen = fread(ibuf, 1, ONCE_READ_LEN, fp_in);
   if (readlen <= 0)
      break;
   fbuf_strc.len_in = readlen;
   nSize = _SD_Filter_Control(pfilter, &fbuf_strc);
   fwrite(fbuf_strc.buf_out,1,nSize,fp_out);
```

```
    }
    free(ibuf);
    ibuf = AK_NULL;
    free(obuf);
    obuf = AK_NULL;
    _SD_Filter_Close(pfilter);
    fclose(fp_in);
    fclose(fp_out);
}
```

5.5 编码调用示例

```
// 对于 ADPCM 录音，首先定义 ADPCM 文件头信息结构体
typedef struct ADPCM_HEADER {
    T_CHR  riff[4];        // "RIFF"
    T_S32  file_size;      // in bytes
    T_CHR  wavefmt_[8];    // "WAVE"
    T_S32  imachunk_size;      // in bytes (0x14 for IMA ADPCM)
    IMA_ADPCM_WAVEFORMAT ima_format;
    T_CHR  fact[4];        //"fact"
    T_S32  factchunk_size;  //fact chunk size
    T_S32  factdata_size;  //fact data size
    T_CHR  data[4];        // "data"
    T_S32  data_length;    // in bytes
} ADPCM_HEADER;
ADPCM_HEADER adpcmh = {
    {"RIFF"},                  //"RIFF"
       0,                      //file size
    {"WAVEfmt "},              //"WAVEfmt "
    0x14,                     //IMAADPCMWAVEFORMAT struct size(0x14 for IMA ADPCM)
    {0x11,                     //IMA adpcm format
    1,                        //channel 1=mono, 2=stereo
```

```
    8000,                   //default 8k sample rate
    0,                      //bytes_per_sec
    2,                      //block align
    16,                     //bits per sample
    2,                      //reserve bit
    505},                   //samples per packet
    {"fact"},               //"fact"
    4,                      //fact chuck size
    0,                      //the number of samples
    {"data"},               //"data"
    0                       //data len
};

T_U32 main()
{
    FILE *fp_src,*fp_dst;
    T_U16 recBufOut[REC_LEN];
    T_U8  temp_bufin[ONCE_REC_LEN+10];
    T_AUDIO_REC_INPUT audio_record_in;
    T_AUDIO_ENC_BUF_STRC enc_buf_strc;
    T_AUDIO_ENC_OUT_INFO enc_output;
    T_AUDIO_REC_STRUCT *audio_encode;
    T_VOID *precord;
    T_U32 retval;
    int readLen=0, encodelen=0;
    int encType = _SD_MEDIA_TYPE_AMR;

    if((fp_src = fopen("c:\\test.pcm","rb"))==NULL)
    {
        printf("open source file error\r\n");
        return 0;
    }
```

```
    if((fp_dst = fopen((“c:\\test.amr”,"wb"))==NULL)
    {
        printf("open destination file error\r\n");
        return 0;
    }
    audio_record_in.enc_in_info.m_Type = encType;
    audio_record_in.enc_in_info.m_nSampleRate    = 8000;
    audio_record_in.enc_in_info.m_BitsPerSample =16;
    audio_record_in.enc_in_info.m_nChannel       = 1;

    audio_record_in.cb_fun.Malloc = (MEDIALIB_CALLBACK_FUN_MALLOC)malloc;
    audio_record_in.cb_fun.Free = (MEDIALIB_CALLBACK_FUN_FREE)free;
    audio_record_in.cb_fun.printf = (MEDIALIB_CALLBACK_FUN_PRINTF)printf;

    /* find encType, write file */
    switch (encType)
    {
        case _SD_MEDIA_TYPE_AMR:
            audio_record_in.enc_in_info.m_private.m_amr_enc.mode = mode_bitrate_bits;
            fwrite("#!AMR\n", sizeof(char), 6, fp_dst); // 写文件头·
            audio_record_in.cb_fun.printf("arm encode mode %dk\r\n",
audio_record_in.enc_in_info.m_private.m_amr_enc.mode);
            break;
        case _SD_MEDIA_TYPE_AAC:
            audio_record_in.cb_fun.printf("aac encode \r\n");
            break;
        case _SD_MEDIA_TYPE_MP3:
            audio_record_in.enc_in_info.m_private.m_mp3.bitrate = 128;
            audio_record_in.cb_fun.printf("mp3 output bitrate %dk\r\n",
audio_record_in.enc_in_info.m_private.m_mp3.bitrate);
            break;
        case _SD_MEDIA_TYPE_ADPCM_IMA:
```

```
        audio_record_in.enc_in_info.m_private.m_adpcm.enc_bits = 4;
        audio_record_in.cb_fun.printf("adpcm encode bits %dk\r\n",
audio_record_in.enc_in_info.m_private.m_adpcm.enc_bits);

        // 预留 ADPCM 文件头的位置
        adpcmh.file_size = encodelen + sizeof(adpcmh) - 8;
        adpcmh.factdata_size = 0;
        adpcmh.data_length = encodelen;
        fwrite(&adpcmh, 1, sizeof(adpcmh), fp_dst);
        break;
    case _SD_MEDIA_TYPE_SPEEX:
        audio_record_in.cb_fun.printf("speex encode \r\n");
        audio_record_in.enc_in_info.m_private.m_speex.cbr = 0;
        audio_record_in.enc_in_info.m_private.m_speex.bitrate =5950;
        audio_record_in.enc_in_info.m_private.m_speex.dtx_disable = 0;
        break;
    default:
        audio_record_in.enc_in_info.m_private.m_amr_enc.mode = AMR_ENC_MR515;
        fwrite("#!AMR\n", sizeof(char), 6, fp_dst);
        break;
  }

  precord = _SD_Encode_Open(&audio_record_in,&enc_output);
  if(precord == AK_NULL)
  {
    return 0;
  }

  audio_encode = (T_AUDIO_REC_STRUCT *)precord;
  audio_encode->buf_in.buf_rp = 0;
  audio_encode->buf_in.buf_wp = 0;
  audio_encode->buf_out.buf_rp = 0;
```

```
audio_encode->buf_out.buf_wp = 0;
encodelen = 0;

while (1)
{
    enc_buf_strc.buf_in = temp_bufin;
    enc_buf_strc.buf_out = recBufOut;
    enc_buf_strc.len_out = sizeof(recBufOut);
    readLen = fread(enc_buf_strc.buf_in,sizeof(char),ONCE_REC_LEN,fp_src);
    if (readLen <=0 )
    {
        break;
    }
    enc_buf_strc.len_in = readLen;

    retval = _SD_Encode(precord,&enc_buf_strc);
    audio_record_in.cb_fun.printf("retval is %d!\r\n",retval);
    fwrite(audio_encode->buf_out.pbuf,sizeof(char),retval,fp_dst);
    if (retval > 0)
    {
        encodelen += retval;
    }
}

// 回写文件头
switch (encType)
{
case _SD_MEDIA_TYPE_SPEEX:
        memset(temp_bufin, 0, 160*2*wav_head.num_chans);
        enc_buf_strc.buf_in = temp_bufin;
        enc_buf_strc.len_in = 160*2*wav_head.num_chans;
        enc_buf_strc.buf_out = recBufOut;
```

```
            enc_buf_strc.len_out = sizeof(recBufOut);
            retval = _SD_Encode_Last(audio_encode,&enc_buf_strc);
            audio_record_in.cb_fun.printf("retval is %d!\r\n",retval);
            if (retval > 0)
            {
               fwrite(audio_encode->buf_out.pbuf,sizeof(char),retval,fp_dst);
               encodelen += retval;
            }
            break;
      case _SD_MEDIA_TYPE_AMR:
         break;
      case _SD_MEDIA_TYPE_AAC:
         break;
      case _SD_MEDIA_TYPE_MP3:
         break;
      case _SD_MEDIA_TYPE_ADPCM_IMA:
         adpcmh.ima_format.nChannels = wav_head.num_chans;
         adpcmh.ima_format.nSamplesPerSec = wav_head.sample_rate;
         {
            T_AUDIO_REC_STRUCT *padpcm = (T_AUDIO_REC_STRUCT *)precord;
            adpcmh.ima_format.nBlockAlign = padpcm-
>enc_out_info.m_Private.m_adpcm.nBlockAlign;
            adpcmh.ima_format.wFormatTag = padpcm->enc_out_info.wFormatTag;
            adpcmh.ima_format.wBitsPerSample = padpcm-
>enc_out_info.m_Private.m_adpcm.wBitsPerSample;
            adpcmh.ima_format.wSamplesPerBlock = padpcm-
>enc_out_info.m_Private.m_adpcm.nSamplesPerPacket;
            adpcmh.ima_format.nAvgBytesPerSec = padpcm-
>enc_out_info.m_Private.m_adpcm.nAvgBytesPerSec;
         }
         adpcmh.file_size = encodelen + sizeof(adpcmh) - 8;
         adpcmh.factdata_size = 0;
```

```
        adpcmh.data_length = encodelen;

        fseek(fp_dst, 0, SEEK_SET);
        fwrite(&adpcmh, 1, sizeof(adpcmh), fp_dst);
        break;
    default:
        break;
    }

    fclose(fp_src);
    fclose(fp_dst);
    _SD_Encode_Close(precord);

    return 0;
}
```

5.6 DA buffer控制接口调用示例

```
main()
{
    Media_Open(...); //实现见后续内容
    Media_Play(...);
    while (1)
    {
        Media_Handle(...)
        Media_Pause(...);
        Media_Play(...);
        if (fileEnd)
        {
            break;
        }
    }
```

```
}

Media_open(...)
{
   _SD_Decode_Open(...);
   if (bFadeEnable)
   {
      DABuf_SetFadetime(bFadeInTime, FADE_IN);
      DABuf_SetFadetime(bFadeOutTime, FADE_OUT);
   }

   fade_type = FADE_NORMAL;
   DABuf_SetFade(pMedialib->nSamplesPerSec, FADE_NORMAL); //这里要设置一下，不然可能导致设置音量为 0 时仍然有声音

   if (DABuf_SetPCMInfo(m_Channels, m_BitsPerSample) == 0)
   {
      return AK_NULL;
   }
}
Media_play(...)
{
   if (bNeedSeek)
   {
      //init da buf
      if (!DABuf_SetSta(DABUF_INIT))
      {
         return; //error
      }
      DABuf_Clear();
   }
   if (bFadeEnable)
```

```
    {
        if (DABuf_SetFade(nSamplesPerSec, FADE_IN))
        {
            fade_type = FADE_IN;
        }
        else
        {
            fade_type = FADE_NORMAL;
        }
    }
}
Media_Handle(...)
{
    while (!DABuf_GetSta(DABUF_FULL))
    {
        Fill_Audiodata(...);

        retval = _SD_Decode(...);
        if (retval < 0)
        {
            DABuf_SetSta(DABUF_END);
        }

        if (fade_type != FADE_NORMAL)
        {
            T_U32 volume;
            T_S32 ret = 0;

            ret = DABuf_GetVolume(&volume);
            if (AK_TRUE == ret)
            {
                DABuf_SetVolume(volume);
```

```
        }
        else// if (AK_FALSE == ret)
        {
            if (FADE_OUT == pMedialib->fade_type)
            {
                fade_type = FADE_NORMAL;
                DABuf_SetFade(pMedialib->nSamplesPerSec, FADE_NORMAL);
                break;
            }
            else
            {
                fade_type = FADE_NORMAL;
                DABuf_SetFade(pMedialib->nSamplesPerSec, FADE_NORMAL);
            }
        }
    }

    DABuf_SendData(AudioDecOut.m_pBuffer, retval);
  }
}

Media_pause()
or
Media_Stop()
{
  if (bFadeEnable)
  {
    if (DABuf_SetFade(nSamplesPerSec, FADE_OUT))
    {
      fade_type = FADE_OUT;
    }
    else
```

```
        {
            fade_type = FADE_NORMAL;
        }
    }
}
```

5.7 快速响应的播放器打开接口调用示例

```
void test_new_open(void)
{
    FILE *fInput = NULL;
    FILE *fOutput = NULL;
    int FrameLen = 0;
    T_AUDIO_DECODE_STRUCT *pdecode = NULL;
    T_AUDIO_DECODE_INPUT audio_input ={0};
    T_AUDIO_DECODE_OUT audio_output;
    T_U8 obuf[OUTBUF_LEN];
    T_U8 tmpbuf[4096];

    fInput = fopen("g:\\test_case\\vorbis\\16bit_q0.ogg","rb");
    if(fInput == NULL)
    {
        return;
    }
    fOutput = fopen("g:\\ogg_dec.pcm","wb");
    if(fOutput == NULL)
    {
        fclose(fInput);
        return;
    }

    // 传参数
    audio_input.cb_fun.Malloc = (MEDIALIB_CALLBACK_FUN_MALLOC)malloc;
```

```
audio_input.cb_fun.Free = (MEDIALIB_CALLBACK_FUN_FREE)free;
audio_input.cb_fun.delay = 0;
audio_input.cb_fun.printf = (MEDIALIB_CALLBACK_FUN_PRINTF)printf;
audio_input.m_info.m_Type = _SD_MEDIA_TYPE_OGG_VORBIS;
audio_input.m_info.m_InbufLen = 1024*128;
audio_input.m_info.m_BitsPerSample = 16;
audio_input.m_info.m_Channels = 1;
audio_input.m_info.m_SampleRate = 32000;

// 打开
if (_SD_MEDIA_TYPE_OGG_VORBIS == audio_input.m_info.m_Type)
{
    // ogg vorbis 的新接口进行打开音频库
    int size;
    int ret;
    // read the first page data to m_szData
    size = fread(&tmpbuf[0], 1, sizeof(tmpbuf), fInput);
    audio_input.m_info.m_szData = &tmpbuf[0];
    audio_input.m_info.m_szDataLen = size;
    pdecode = _SD_Decode_Open_Fast(&audio_input, &audio_output);
    if (AK_NULL == pdecode)
        return;
    // read first three page data at least
    fseek(fInput, 0, SEEK_SET);
    while (0 ==ret)
    {
        size = FillAudioData((void *)pdecode, fInput); //给音频库的码流 buffer 填数据，该函数需要调用者自己实现，实现方法可以参考 5.2 的调用示例
        ret = _SD_Decode_ParseFHead(pdecode);
        if (ret <0)
            return;
    }
```

```
}

// 解码
while (1)
{
    int readsize;

    if ( ((readsize=FillAudioData((void *)pdecode, fInput)) <= 0)
        && (FrameLen<=0) )
    {
        break;
    }

    audio_output.m_pBuffer = obuf;
    audio_output.m_ulSize = sizeof(obuf);
    FrameLen = _SD_Decode(pdecode, &audio_output);
    if (FrameLen > 0)
    {
        fwrite(obuf, FrameLen, 1, fOutput);
    }
    if (FrameLen<0)
        break;
}

// 切换新文件，注意新文件与之前的文件采样率、声道等信息要一致。
if(fInput != NULL)   fclose(fInput);
fInput = fopen("g:\\test_case\\vorbis\\16bit_q10.ogg","rb");
if(fInput == NULL)    return;

if (_SD_MEDIA_TYPE_OGG_VORBIS == audio_input.m_info.m_Type)
{
    int size, ret=0;
```

```
    _SD_Buffer_Clear(pdecode);
    while (0 ==ret)
    {
        size = FillAudioData((void *)pdecode, fInput);
        ret = _SD_Decode_ParseFHead(pdecode);
        if (ret < 0)
        {
            audio_input.cb_fun.printf("## _SD_Decode_ParseFHead() fail\r\n");
            _SD_Decode_Close(pdecode);
            return;
        }
    }
}
// 解码
while (1)
{
    int readsize;

    if ( ((readsize=FillAudioData((void *)pdecode, fInput)) <= 0)
        && (FrameLen<=0) )
    {
        break;
    }

    FrameLen = _SD_Decode(pdecode, &audio_output);
    if (FrameLen > 0)
    {
        fwrite(obuf, FrameLen, 1, fOutput);
    }
    if (FrameLen<0)
        break;
}
```

```
    if(fInput != NULL)    fclose(fInput);
    if(fOutput != NULL)   fclose(fOutput);

    _SD_Decode_Close(pdecode);
}
```