ANYKA
芯 随 你 动

版本: **1.0.2 2012** 年 **01** 月

媒体播放库接口说明

声　　明

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
V1.0.1	正式发布	2011-12
V1.0.2	增加获取 meta 信息的接口	2012-1

媒体播放库接口说明与媒体播放库对应的关系

Version	Corresponding media lib	Corresponding video driver lib	Corresponding audio driver lib
V1.0.2	V1.12.00	V1.7.10	V1.7.09
V1.0.1	V1.11.02	V1.7.09	V1.7.08
V0.3.2	V1.11.02	V1.7.09	V1.7.06
V0.3.1	V0.10.2	V0.7.6	V0.7.4
V0.2.0	V0.8.0	V0.7.0	V0.6.0
V0.1.5	V0.5.3	V0.6.2	V0.3.3
V0.1.4	V0.2.3	V0.2.3	
V0.1.2	V0.1.0	V0.1.0	V0.1.0

目录

1 模块介绍

1.1 功能概述

1.1.1 音频使用DSP编解码的芯片系列

AK32XX、AK36XX 以及 AK78XX 系列芯片：

本模块实现了以下与电影播放相关功能：

- 支持视频编码 MPEG4（SP Level 0－3）或 H.263 （Baseline），音频编码 MP3/AMR/AAC
- 支持播放 AVI/AKV/3GP/MP4 格式的媒体
- 预览媒体中某个画面，播放过程中快速播放、暂停、继续、停止
- 视频播放最大分辨率为 CIF（H×V：352×288），AK78XX 支持到 VGA（H×V：640×480），AK36XX 支持到 VGA
- 播放 QCIF 或以下 fps <= 30，QCIF 以上 fps <= 25
- 支持最高 25fps@600Kbps(SRAM) 或 30fps@870.4Kbps(SDRAM) 或 25fps@1Mbps(SDRAM)的媒体播放
- 音频限制：AAC 码率<128kbps, 采样率<48000; AMR 采样率 8000

本模块实现了以下与音频播放相关功能：

- 支持 amr/aac/mp3 /wav/wma 的播放，与原有音频库的功能/性能指标一致。
- AAC/mp3 支持 bitrate <＝ 320kbps， Samplerate <= 48KHZ；
- AMR 支持所有编码模式；
- WMA 现阶段还不支持，将来支持 bitrate <＝ 320kbps Samplerate <= 48KHZ；
- MIDI 暂不支持
- 支持重采样

1.1.2 音频使用软编解码的芯片系列

AK88XX 芯片：

文件格式支持列表：

文件格式	常用音视频组合	
AVI	XVID（MPEG4 SP/ASP）	MP3
	H263	MP3
	MJPEG	PCM
MP4	MPEG4 SP/ASP	AAC
3GP	H263	AMR
FLV	FLV263	MP3
RM/RMVB	RealVideo	COOK
ASF/WMV/WMA	WMV	WMA
MKV	H264	AC3
F4V	H264	AAC
音频格式		
WAV		PCM/ADPCM
MP3		MP3
AMR		AMR
ADTS/ADIF		AAC
APE		APE
FLAC		FLAC
MIDI		MIDI
OGG		VORBIS
AC3		AC3

音视频限制情况列表：

视频编解码类型	芯片支持尺寸	LCD 限制
H263	CIF	

视频编解码类型	芯片支持尺寸	LCD 限制
	（352×288）	
MPEG4 SP	VGA （640×480）	
FLV263	VGA （640×480）	
MJPEG		1008×1008
H264	D1 （720×576）	
RealVideo	1024×1024	1008×1008
MPEG4 ASP	1280×1280	1008×1008

音频编解码类型	采样率	声道数	比特率（bps）
PCM	48000		
ADPCM			
MP3			
AMR	8000	1	
AAC	48000		320K
FLAC			
MIDI			

APE 限制情况

级别	支持情况		
<=4000	支持		
5000	不支持		

AK98XX 芯片：

文件格式支持同 AK88XX。

视频支持尺寸列表：

视频编解码类型	芯片支持尺寸
H263	4CIF（704×576）
MPEG4 SP	D1（720×576）
FLV263	D1（720×576）
MJPEG	1280×1280
H264	720P （1280×720）
RealVideo	720P （1280×720）
MPEG4 ASP	1280×1280

AK37XX 芯片：

文件格式支持上音频同 AK88XX，视频仅支持 AVI、MP4、3GP、FLV。

视频支持尺寸列表：

视频编解码类型	芯片支持尺寸
H263	4CIF（704×576）
MPEG4 SP	D1（720×576）
FLV263	D1（720×576）
MJPEG	1280×1280

1.2 与其它模块关系

本模块依赖于以下模块：

- 资源管理模块
- 内存管理模块
- 系统功能模块

本模块需要调用以上模块相关接口进行资源读取与写入、内存分配释放等，具体见第 5 章“依赖接口说明”。

2 相关文档

《音视频库总体使用说明》

3 集成指南

如果是首次使用媒体播放库，请首先阅读《音视频库总体使用说明》。

3.1 层次结构

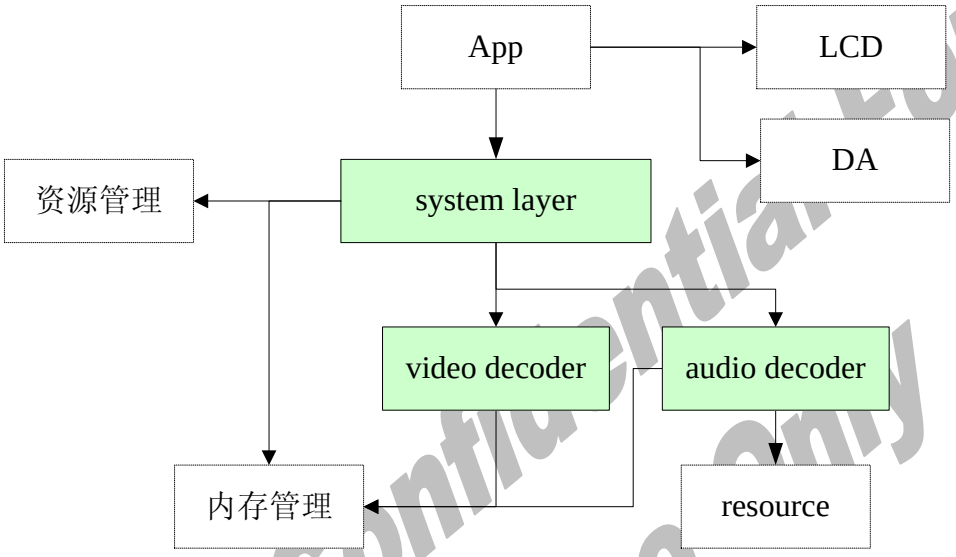

图 3-1 模块关系图

媒体播放库包含系统层库、视频解码库、音频解码库，集成后在系统中的位置如上图所示。媒体播放库依赖的模块都应该在使用前由系统初始化。App 应用模块实现播放，调用系统层模块、控制 LCD 及 DA 实现播放；内存管理模块用于申请和释放内存；资源管理模块提供基本的资源读写，定位等功能。箭头标明在运行播放过程中模块的调用情况。

注意与以往的音视频库不同的是，修改后的媒体播放库在音视频解码后的数据不再由媒体播放库调用 DA 和 LCD 进行音视频的输出，而是统一由 App 应用模块实现媒体的输出。

3.2 集成步骤

3.2.1 获得媒体播放库相关文件

媒体播放库包括库二进制文件、头文件和相关文档。

3.2.2 实现媒体播放库依赖的系统API

请仔细阅读本文的第五部分，准备好媒体播放库所依赖的系统 API，为系统控制媒体播放库提供足够的灵活性，需要在集成时根据实际情况由系统集成人员实现。

3.2.3 集成链接测试

把媒体播放库、依赖的系统 API 实现和目标系统进行链接。

3.2.4 联合调试

在目标系统上联合调试一般分成以下步骤，调试和使用媒体播放库常见问题请参考本文的相关部分。

1）调试播放无声媒体，确定系统层和视频解码内部播放基本正常运作

2）调试播放音频媒体，确定系统层和音频解码内部播放基本正常运作

3）调试播放有声媒体，确定系统层、视频解码和音频解码相关接口正常运作

4）调试两路音频媒体播放，确定系统层和音频解码相关接口及混音正常运作

5）调试各种类型媒体的播放，确定媒体播放库正常运作

4 接口说明

4.1 模块功能概述

本模块起播放媒体的作用，请参考功能概述。

4.2 数据结构/格式

4.2.1 回调函数定义

见第 5 章“依赖接口说明”。

4.2.2 初始化结构体

T_MEDIALIB_INIT_INPUT，参见《音视频库总体使用说明》。

4.2.3 媒体播放句柄

typedef T_pVOID T_MEDIALIB_STRUCT;

用于媒体播放，存放播放库的所有信息，外部调用不需要知道具体定义，传入库后做强制转换。

4.2.4 媒体打开输入结构体

T_eMEDIALIB_MEDIA_TYPE 和 T_MEDIALIB_CB 参见《音视频库总体使用说明》。

```
typedef enum
{
    MEDIALIB_PLAY,
    MEDIALIB_PREVIEW,
    MEDIALIB_PLAY_WITHERR,
    MEDIALIB_PREVIEW_WITHERR,
    MEDIALIB_GET_INFO
}T_eMEDIALIB_OPEN_TYPE;
```

结构名	T_eMEDIALIB_OPEN_TYPE	
定义概述	打开类型	
成员说明	MEDIALIB_PLAY	正常播放媒体，解码器出错时退出
	MEDIALIB_PREVIEW	仅预览，解码器出错时退出
	MEDIALIB_PLAY_WITHERR	播放媒体，不检测解码错误
	MEDIALIB_PREVIEW_WITHERR	预览，不检测解码错误
	MEDIALIB_GET_INFO	仅取得媒体信息，不进行播放和预览

```
typedef enum
{
        MEDIALIB_ROTATE_0,
        MEDIALIB_ROTATE_90,
        MEDIALIB_ROTATE_180,
        MEDIALIB_ROTATE_270
}T_eMEDIALIB_ROTATE;

typedef struct
{
        T_eMEDIALIB_OPEN_TYPE   m_OpenType;
        T_eMEDIALIB_MEDIA_TYPE  m_MediaType;
        T_S32                   m_hMediaSource;
        T_AUDIO_OUT_INFO        m_AudioOutInfo;
        T_VIDEO_OUT_INFO        m_VideoOutInfo;
        T_eMEDIALIB_ROTATE      m_Rotate;
        T_U8                    m_AudioAttribute;
        T_BOOL                  m_bCapabilityTest;
        T_BOOL                  m_bAVDiffTask;
        T_BOOL                  m_bEncryptFile;
        T_MEDIALIB_CB           m_CBFunc;
}T_MEDIALIB_OPEN_INPUT;
```

结构名	T_MEDIALIB_OPEN_INPUT	
定义概述	打开时输入参数	
成员说明	m_OpenType	见 T_eMEDIALIB_OPEN_TYPE
	m_MediaType	媒体类型，输入未知时库内部会对媒体类型进行判断； 建议输入 MEDIALIB_MEDIA_UNKNOWN
	m_hMediaSource	媒体资源句柄
	m_AudioOutInfo	音频输出信息，仅对重采样有效
	m_VideoOutInfo	视频输出信息，暂时无用
	m_Rotate	旋转角度，仅对 AK37XX 及以后的芯片有效
	m_AudioAttribute	音频属性，AK78/88/98XX 系列不使用
	m_bCapabilityTest	是否是进行性能测试，如果为 TRUE，库中发现帧率和码率超过播放范围，仍然会继续播放；正常情况下应该设为 FALSE
	m_bAVDiffTask	音视频解码是否在不同的任务中调用，目前已不使用
	m_bEncryptFile	是否为汉策加密 AVI 视频文件
	m_CBFunc	见 T_MEDIALIB_CB

4.2.5 媒体打开输出结构体

```
typedef enum
{
        MEDIALIB_OK,
        MEDIALIB_FORMAT_ERR,
        MEDIALIB_PARAM_ERR,
        MEDIALIB_SYSTEM_ERR,
        MEDIALIB_SUPPORT_ERR
}T_eMEDIALIB_STATE;
```

结构名	T_eMEDIALIB_STATE	
定义概述	打开结果	
成员说明	MEDIALIB_OK	打开成功
	MEDIALIB_FORMAT_ERR	格式错误
	MEDIALIB_PARAM_ERR	参数错误
	MEDIALIB_SYSTEM_ERR	内存不足或读资源出错等
	MEDIALIB_SUPPORT_ERR	不支持的格式

```
typedef struct
{
    T_eMEDIALIB_STATE       m_State;
    T_U32                   m_ulAudioDecBufSize;
    T_U32                   m_ulVideoDecBufSize;
    T_eMEDIALIB_ROTATE      m_Rotate;
    T_U8                    m_AudioAttribute;
}T_MEDIALIB_OPEN_OUTPUT;
```

结构名	T_MEDIALIB_OPEN_OUTPUT	
定义概述	打开时输入参数	
成员说明	m_State	见 T_eMEDIALIB_STATE
	m_ulAudioDecBufSize	音频解码缓冲区的期望大小，以字节为单位
	m_ulVideoDecBufSize	视频解码缓冲区的期望大小，以字节为单位
	m_Rotate	是否进行图像旋转
	m_AudioAttribute	音频解码类型，0：dsp 解码；1：软解码

4.2.6 媒体信息结构体

```
typedef enum
{
        MEDIALIB_VIDEO_UNKNOWN,
        MEDIALIB_VIDEO_MPEG4,
        MEDIALIB_VIDEO_H263,
        MEDIALIB_VIDEO_WMV,
        MEDIALIB_VIDEO_FLV263,
        MEDIALIB_VIDEO_H264,
        MEDIALIB_VIDEO_RV,
        MEDIALIB_VIDEO_MJPEG
}T_eMEDIALIB_VIDEO_CODE;

typedef enum
{
        MEDIALIB_AUDIO_UNKNOWN,
        MEDIALIB_AUDIO_AMR,
        MEDIALIB_AUDIO_MP3,
        MEDIALIB_AUDIO_AAC,
        MEDIALIB_AUDIO_PCM,
        MEDIALIB_AUDIO_WMA,
        MEDIALIB_AUDIO_MIDI,
        MEDIALIB_AUDIO_ADPCM,
        MEDIALIB_AUDIO_APE,
        MEDIALIB_AUDIO_FLAC,
        MEDIALIB_AUDIO_RA,
        MEDIALIB_AUDIO_VORBIS,
        MEDIALIB_AUDIO_AC3
}T_eMEDIALIB_AUDIO_CODE;

typedef struct
{
```

```
    T_U32  m_SampleRate;
    T_U16  m_Channels;
    T_U16  m_BitsPerSample;
    T_U32  m_Type;
    T_U32  m_BitRate;
}T_MEDIALIB_AUDIO_INFO;
```

结构名	T_MEDIALIB_AUDIO_INFO	
定义概述	音频信息	
成员说明	m_SampleRate	采样率
	m_Channels	声道数
	m_BitsPerSample	每个样本的比特数
	m_Type	音频类型
	m_BitRate	音频比特率

```
typedef struct
{
    T_U16  m_uOriWidth;    //not 16 align, in media resource
    T_U16  m_uOriHeight;   //not 16 align, in media resource
    T_U16  m_uWidth;
    T_U16  m_uHeight;
    T_U16  m_uFPS;
    T_U32  m_ulBitrate;
}T_MEDIALIB_VIDEO_INFO;
```

结构名	T_MEDIALIB_VIDEO_INFO	
定义概述	视频信息	
成员说明	m_uOriWidth	原始宽度

结构名	T_MEDIALIB_VIDEO_INFO	
	m_uOriHeight	原始高度
	m_uWidth	16 倍数对齐的图像宽
	m_uHeight	16 倍数对齐的图像高
	m_uFPS	视频帧率
	m_ulBitrate	视频比特率

```
typedef struct
{
        T_eMEDIALIB_MEDIA_TYPE  m_MediaType;
        T_BOOL                  m_bHasVideo;
        T_BOOL                  m_bHasAudio;
        T_BOOL                  m_bAllowSeek;
        T_U32                   m_ulTotalTime_ms;

        T_eMEDIALIB_VIDEO_CODE  m_VideoCode;
        T_eMEDIALIB_AUDIO_CODE  m_AudioCode;
        T_MEDIALIB_VIDEO_INFO   m_VideoInfo;
        T_MEDIALIB_AUDIO_INFO   m_AudioInfo;

        T_MEDIALIB_META_INFO    *m_pMetaInfo;
}T_MEDIALIB_MEDIA_INFO;
```

结构名	T_MEDIALIB_MEDIA_INFO	
定义概述	视频信息	
成员说明	m_MediaType	见 T_eMEDIALIB_MEDIA_TYPE 定义
	m_bHasVideo	是否包含视频
	m_bHasAudio	是否包含音频
	m_bAllowSeek	是否允许定位操作
	m_ulTotalTime_ms	播放总时间，以毫秒为单位
	m_VideoCode	视频编码方式
	m_AudioCode	音频编码方式
	m_VideoInfo	见 T_MEDIALIB_VIDEO_INFO
	m_AudioInfo	见 T_MEDIALIB_AUDIO_INFO
	m_pMetaInfo	Meta 信息指针

T_MEDIALIB_META_INFO 参见《音视频库总体使用说明》。

4.2.7 解码输出结构体

T_AUDIO_DECODE_OUT 和 T_VIDEO_DECODE_OUT 参见《音视频库总体使用说明》。

4.2.8 媒体播放状态

```
typedef enum
{
    MEDIALIB_END,
    MEDIALIB_PLAYING,
    MEDIALIB_FF,
    MEDIALIB_FR,
    MEDIALIB_PAUSE,
    MEDIALIB_STOP,
```

MEDIALIB_ERR,

MEDIALIB_SEEK

}T_eMEDIALIB_STATUS;

结构名	T_eMEDIALIB_STATUS	
定义概述	视频信息	
成员说明	MEDIALIB_END	播放结束
	MEDIALIB_PLAYING	正在播放
	MEDIALIB_FF	快进播放
	MEDIALIB_FR	快退播放
	MEDIALIB_PAUSE	播放暂停
	MEDIALIB_STOP	播放停止
	MEDIALIB_ERR	播放出错
	MEDIALIB_SEEK	定位

4.3 接口函数列表

MediaLib_GetVersion、MediaLib_Init、MediaLib_Destroy，参见《音视频库总体使用说明》。

4.3.1 MediaLib_Open

原　　型	T_MEDIALIB_STRUCT MediaLib_Open(const T_MEDIALIB_OPEN_INPUT *open_input, T_MEDIALIB_OPEN_OUTPUT *open_output);	
功能概述	初始化播放器，并获得媒体资源的主要属性信息	
参数说明	open_input	见 T_MEDIALIB_OPEN_INPUT
	open_output	见 T_MEDIALIB_OPEN_OUTPUT
返回值说明	返回打开资源的播放句柄	
返回值说明	AK_NULL：打开失败； 其他：打开成功	

原　　型	T_MEDIALIB_STRUCT MediaLib_Open(const T_MEDIALIB_OPEN_INPUT *open_input, T_MEDIALIB_OPEN_OUTPUT *open_output);
注意事项	
调用示例	见典型调用示例

4.3.2 MediaLib_Close

原　　型	T_BOOL MediaLib_Close(T_MEDIALIB_STRUCT hMedia)	
功能概述	关闭播放器，释放资源	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
返回值说明	返回关闭播放器的结果	
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	

4.3.3 MediaLib_Preview

原　　型	T_BOOL MediaLib_Preview(T_MEDIALIB_STRUCT hMedia, T_pVOID pImgYUV, T_U32 ulMilliSec);	
功能概述	预览已打开视频媒体中某一帧的画面	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
	pImgYUV	存放解码后图像的首地址
	ulMilliSec	需要预览图像的位置，如果找不到则输出第一个关键帧
返回值说明	预览是否成功	
返回值说明	AK_TRUE　　预览成功 AK_FALSE　　预览失败	
注意事项	必须已经成功执行 MediaLib_Open 函数	

原　　型	T_BOOL MediaLib_Preview(T_MEDIALIB_STRUCT hMedia, T_pVOID pImgYUV, T_U32 ulMilliSec);
调用示例	见典型调用示例

4.3.4 MediaLib_GetInfo

原　　型	T_BOOL MediaLib_GetInfo(T_MEDIALIB_STRUCT hMedia, T_MEDIALIB_MEDIA_INFO *pInfo)	
功能概述	获得当前媒体的属性信息和状态信息	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
	pInfo	存放媒体信息的数据结构的指针
返回值说明	获取信息的结果	
返回值说明	AK_TRUE　　获取信息成功 AK_FALSE　　获取信息失败	
注意事项	1. 必须已经成功执行 MediaLib_Open 函数。 2. 必须保证 pInfo 不为空指针。	
调用示例	见典型调用示例	

4.3.5 MediaLib_Play

原　　型	T_S32 MediaLib_Play(T_MEDIALIB_STRUCT hMedia);	
功能概述	从当前位置启动播放	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
返回值说明	当前的媒体时间，毫秒为单位；返回负数表示失败	
返回值说明		
注意事项	必须已经成功执行 MediaLib_Open 函数	
调用示例	见典型调用示例	

4.3.6 MediaLib_Stop

原　　型	T_BOOL MediaLib_Stop(T_MEDIALIB_STRUCT hMedia)	
功能概述	停止当前播放的媒体，当前位置返回到开始位置	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
返回值说明	停止的结果	
返回值说明	AK_TRUE　　停止成功 AK_FALSE　　停止失败	
注意事项	必须已经成功执行 MediaLib_Open 函数	
调用示例	见典型调用示例	

4.3.7 MediaLib_Pause

原　　型	T_BOOL MediaLib_Pause(T_MEDIALIB_STRUCT hMedia)	
功能概述	暂停当前播放的媒体，保持当前位置	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
返回值说明	暂停的结果	
返回值说明	AK_TRUE　　暂停成功 AK_FALSE　　暂停失败	
注意事项	必须已经成功执行 MediaLib_Open 函数	
调用示例	见典型调用示例	

4.3.8 MediaLib_FastForward

原　　型	T_BOOL MediaLib_FastForward (T_MEDIALIB_STRUCT hMedia)	
功能概述	快速播放当前的媒体	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
返回值说明	快速播放是否成功	

原　　型	T_BOOL MediaLib_FastForward (T_MEDIALIB_STRUCT hMedia)
返回值说明	AK_TRUE　　快进成功 AK_FALSE　　快进失败
注意事项	必须已经成功执行 MediaLib_Open 函数
调用示例	

4.3.9 MediaLib_FastRewind

原　　型	T_BOOL MediaLib _FastRewind (T_MEDIALIB_STRUCT hMedia)	
功能概述	将媒体转换到快速后退状态	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
返回值说明	AK_TRUE　　快退成功 AK_FALSE　　快退失败	
注意事项	必须已经成功执行 MediaLib_Open 函数	
调用示例		

4.3.10 MediaLib_GetStatus

原　　型	T_eMEDIALIB_STATUS MediaLib_GetStatus(T_MEDIALIB_STRUCT hMedia)	
功能概述	获得当前播放的状态信息	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
返回值说明	媒体播放的状态信息	
返回值说明	见 T_eMEDIALIB_STATUS	
注意事项	必须已经成功执行 MediaLib_Open 函数。	
调用示例	见典型调用示例	

4.3.11 MediaLib_DecodeVideoPack

原　　型	T_S32 MediaLib_DecodeVideoPack(T_MEDIALIB_STRUCT hMedia, T_VIDEO_DECODE_OUT *pVideoDecOut, T_S32 ulMilliSec);	
功能概述	根据当前媒体播放的状态，处理媒体播放时的视频同步解码	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
	pVideoDecOut	见 T_VIDEO_DECODE_OUT
	ulMilliSec	需要同步播放的时间，可根据音频时间或系统时间，再根据 Play 返回的时间换算获得
返回值说明	解出的图像对应的媒体时间	
返回值说明	返回负数可能是播完或出错，可根据 status 获得状态区分	
注意事项	1. 必须已经成功执行 MediaLib_Open 函数。 2. 必须在程序中反复调用本函数，并保证调用本函数的时间间隔足够短，否则不能保证音频和视频的同步。	
调用示例	见典型调用示例	

4.3.12 MediaLib_DecodeAudioPack

原　　型	T_S32 MediaLib_DecodeAudioPack(T_MEDIALIB_STRUCT hMedia, T_AUDIO_DECODE_OUT *pAudioDecOut);	
功能概述	处理媒体播放时的音频同步解码	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
	pAudioDecOut	见 T_AUDIO_DECODE_OUT
返回值说明	解码出的音频数据量，以字节为单位	
返回值说明	返回负数可能是播完或出错，可根据 status 获得状态区分	
注意事项	1. 必须已经成功执行 MediaLib_Open 函数。 2. 必须在程序中反复调用本函数，并保证调用本函数的时间间隔足够短，否则不能保证音频和视频的同步。 3. 如果该函数和 MediaLib_DecodeVideoPack 在同一任务中调用，Open 时必须将 m_bAVDiffTask 置为 FALSE，如果这两个函数再两个任务中调用，那么 m_bAVDiffTask 置为 TRUE	

原　　型	T_S32 MediaLib_DecodeAudioPack(T_MEDIALIB_STRUCT hMedia, T_AUDIO_DECODE_OUT *pAudioDecOut);
调用示例	见典型调用示例

4.3.13 MediaLib_SetPosition

原　　型	T_S32 MediaLib_SetPosition(T_MEDIALIB_STRUCT hMedia, T_S32 ulMilliSec, T_BOOL bSeekNext)	
功能概述	定位当前媒体的位置，单位毫秒	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
	msTime	定位的时间点
	bSeekNext	表示 seek 时向前还是向后搜索关键帧（目前已不使用，直接输入 0 即可）
返回值说明	实际定位的位置，以毫秒为单位	
返回值说明	返回负数表示定位失败	
注意事项	1. 必须已经成功执行 MediaLib_Open 函数 2. 定位后的时间点实际上是指定时间点最靠近的一个 I 帧位置，如果整个媒体只有开始一个 I 帧则会定位到媒体开始	
调用示例	见典型调用示例	

4.3.14 MediaLib_ReleaseInfoMem

原　　型	T_BOOL MediaLib_ReleaseInfoMem(T_pVOID hMedia)	
功能概述	释放部分与播放无关的资源，如 meta 信息占用的内存	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
返回值说明	释放是否成功	
返回值说明	AK_TRUE　　释放成功 AK_FALSE　　释放失败	

原　　型	T_BOOL MediaLib_ReleaseInfoMem(T_pVOID hMedia)
注意事项	1. 必须已经成功执行 MediaLib_Open 函数 2. MediaLib_GetInfo 调用之后，上层已获取媒体相关信息并处理，通过调用该函数释放部分与播放无关的资源，减少播放过程中对内存的占用 3. 该函数不调用不会影响播放，也不会造成内存泄漏，MediaLib_Close 时会释放所有资源
调用示例	见典型调用示例

4.3.15 MediaLib_SetDispRotate

原　　型	T_BOOL MediaLib_SetDispRotate(T_pVOID hMedia, T_eMEDIALIB_ROTATE rotate)	
功能概述	设置旋转方向	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
	rotate	见 T_eMEDIALIB_ROTATE
返回值说明	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	1. 必须已经成功执行 MediaLib_Open 函数 2. 并不是所有解码类型都可以中途设置旋转方向，目前仅 h263/flv263/mpeg4 sp/motion jpeg 可以中途设置旋转，其他格式需在 MediaLib_Open 时设置好，中途不允许改变	

调用示例	if (!MediaLib_SetDispRotate(hMedia, MEDIALIB_ROTATE_180))
	{
	　//rotate failed
	}	

4.3.16 MediaLib_GetAudioDecTime

原　　型	T_S32 MediaLib_GetAudioDecTime(T_pVOID hMedia)	
功能概述	获取音频解码时间	
参数说明	hMedia	MediaLib_Open 时返回的媒体播放句柄
返回值说明	返回音频解码时间	
返回值说明	返回负数表示获取失败，其他表示已解码时间，毫秒为单位	
注意事项	必须已经成功执行 MediaLib_Open 函数	
调用示例	T_S32 aDecTime = MediaLib_GetAudioDecTime(hMedia);	

4.3.17 MediaLib_CheckFile

```
typedef struct
{
        T_BOOL          m_bHasVideo;
        T_BOOL          m_bHasAudio;
}T_MEDIALIB_CHECK_OUTPUT;
```

原　　型	T_eMEDIALIB_MEDIA_TYPE MediaLib_CheckFile(T_MEDIALIB_CB *pCBFunc, T_S32 hMediaSource, T_MEDIALIB_CHECK_OUTPUT *check_output)	
功能概述	获取文件的格式信息	
参数说明	pCBFunc	回调函数列表
	hMediaSource	文件句柄
	check_output	T_MEDIALIB_CHECK_OUTPUT 结构体
返回值说明	T_eMEDIALIB_MEDIA_TYPE 类型	
返回值说明	见 T_eMEDIALIB_MEDIA_TYPE	
注意事项	1. 该函数根据文件的格式信息简单判断文件是纯音频文件还是带视频的文件，并不能作为最终的播放依据 2. 对于 mp4 文件，能正确给出文件中的音频或视频是否存在，主要用于区分 mp4 文件是 m4a 文件还是带视频的文件	

原　　型	T_eMEDIALIB_MEDIA_TYPE MediaLib_CheckFile(T_MEDIALIB_CB *pCBFunc, T_S32 hMediaSource, T_MEDIALIB_CHECK_OUTPUT *check_output)

调用示例	T_MEDIALIB_CB CBFunc;
	T_S32 hMediaSource;
	T_eMEDIALIB_MEDIA_TYPE mediaType;
	T_MEDIALIB_CHECK_OUTPUT checkOutput;
	set_callback(&CBFunc);//设置回调
	hMediaSource = open_file();//打开文件，获取文件句柄
	mediaType = MediaLib_CheckFile(&CBFunc, hMediaSource, &checkOutput);

4.3.18 MediaLib_GetPicMetaInfo

原　　型	T_pVOID MediaLib_GetPicMetaInfo(T_MEDIALIB_CB *pCBFunc, T_S32 hMediaSource, T_U8 **picBuf, T_U32 *picLen)	
功能概述	获取 mp3 文件的图片信息	
参数说明	pCBFunc	回调函数列表
	hMediaSource	文件句柄
	picBuf	输出图片数据地址
	picLen	输出图片数据字节长度
返回值说明	T_pVOID	
返回值说明	本次操作分配的句柄	
注意事项	返回的句柄需保存，并在图片数据使用完之后调用 MediaLib_ReleasePicMetaInfo 释放所有空间	
调用示例		

4.3.19 MediaLib_ReleasePicMetaInfo

原　　型	T_VOID MediaLib_ReleasePicMetaInfo(T_MEDIALIB_CB *pCBFunc, T_pVOID pMetapic)	
功能概述	获取文件的格式信息	
参数说明	pCBFunc	回调函数列表
	pMetapic	MediaLib_GetPicMetaInfo 的返回值
返回值说明	无	
返回值说明		
注意事项	MediaLib_GetPicMetaInfo 执行之后，如果其返回值不为 NULL，必须调用本函数释放空间，否则内存无法回收	
调用示例		

4.3.20 MediaLib_SetDecryptParam

typedef T_S32 (*MEDIALIB_CALLBACK_FUN_DECRYPT_INIT)(T_U8 *pData); //128 bytes

typedef T_S32 (*MEDIALIB_CALLBACK_FUN_DECRYPT_DATA)(T_U8 *pData, T_U32 ulDataLen);

typedef struct

{

MEDIALIB_CALLBACK_FUN_DECRYPT_INIT m_FunDecryptInit;

MEDIALIB_CALLBACK_FUN_DECRYPT_DATA m_FunDecryptData;

}T_MEDIALIB_DECRYPT_PARAM;

结构名	T_MEDIALIB_DECRYPT_PARAM	
定义概述	视频信息	
成员说明	m_FunDecryptInit	解密初始化回调
	m_FunDecryptData	数据解密回调

原　　型	T_BOOL MediaLib_SetDecryptParam(T_MEDIALIB_DECRYPT_PARAM *decrypt_param)	
功能概述	设置解密参数	
参数说明	pCBFunc	回调函数列表
	pMetapic	MediaLib_GetPicMetaInfo 的返回值
返回值说明	无	
返回值说明		
注意事项	1. 必须在 MediaLib_Init 执行之后调用 2. 如果每个文件的参数不同，需要在每次 MediaLib_Open 之前设置正确的参数后调用；如果每个文件的参数相同，在 MediaLib_Init 执行之后设置一次即可； 3. 对于加密 AVI 文件，必须将 T_MEDIALIB_OPEN_INPUT 结构体的 m_bEncryptFile 置为 true；非加密 AVI 文件必须将 T_MEDIALIB_OPEN_INPUT 结构体的 m_bEncryptFile 置为 false；	

调用示例	extern void hc_init(char * buf);
	extern int dec_video(char *video_data, int video_length);
	
	{
	T_MEDIALIB_DECRYPT_PARAM decrypt_par;
	decrypt_par.m_FunDecryptInit =
	(MEDIALIB_CALLBACK_FUN_DECRYPT_INIT)hc_init;
	decrypt_par.m_FunDecryptData =
	
	(MEDIALIB_CALLBACK_FUN_DECRYPT_DATA)dec_video;
	MediaLib_SetDecryptParam(&decrypt_par);
	}	

4.3.21 MediaLib_GetID3MetaInfo

原　　型	T_pVOID MediaLib_GetID3MetaInfo(T_MEDIALIB_CB *pCBFunc, T_S32 hMediaSource, T_eMEDIALIB_MEDIA_TYPE MediaType, T_MEDIALIB_META_INFO *pMetaInfo)	
功能概述	获取文件中的 meta 信息	
参数说明	pCBFunc	回调函数列表
	hMediaSource	文件句柄
	MediaType	指定文件格式类型，上层无法判定时输入 MEDIALIB_MEDIA_UNKNOWN
	pMetaInfo	输出，见 T_MEDIALIB_META_INFO 结构体描述
返回值说明	T_pVOID	
返回值说明	本次操作分配的句柄 AK_NULL：获取失败； 其他：获取成功；	
注意事项	返回的句柄需保存，并在 meta 数据使用完之后调用 MediaLib_ReleaseID3MetaInfo 释放所有空间	

调用示例	T_MEDIALIB_META_INFO MetaInfo;
	T_pVOID handle;
	T_MEDIALIB_CB CBFunc;
	T_S32 hMediaSource = FileOpen(“filename”);
	SetCB(&CBFunc);//设置回调
	handle = MediaLib_GetID3MetaInfo(&CBFunc, hMediaSource, MEDIALIB_MEDIA_UNKNOWN, &MetaInfo);
	...//处理 meta 信息
	MediaLib_ReleaseID3MetaInfo(&open_input.m_CBFunc, handle);
	...	

4.3.22 MediaLib_ReleaseID3MetaInfo

原　　型	MediaLib_ReleaseID3MetaInfo(T_MEDIALIB_CB *pCBFunc, T_pVOID pID3Meta)	
功能概述	获取文件的格式信息	
参数说明	pCBFunc	回调函数列表
	pID3Meta	MediaLib_GetPicMetaInfo 的返回值
返回值说明	无	
返回值说明		
注意事项	MediaLib_GetID3MetaInfo 执行之后，如果其返回值不为 NULL，必须调用本函数释放空间，否则内存无法回收	
调用示例	见 MediaLib_GetID3MetaInfo 调用示例	

4.4 典型调用范例

```
T_VOID play_media(char* filaname);
T_VOID main(int argc, char* argv[])
{
        T_MEDIALIB_INIT_CB init_cb;
        T_MEDIALIB_INIT_INPUT init_input;

        init();    // initial file system, memory, lcd and etc.

        init_cb_func_init(&init_cb);        //initial callback function pointer

        init_input.m_ChipType = MEDIALIB_CHIP_UNKNOWN;
        init_input.m_AudioI2S = I2S_UNUSE;

        if (MediaLib_Init(&init_input, &init_cb) == AK_FALSE)
        {
                return;
```

```
        }
        //above only call one time when system start

        //play film or music
        play_media(argv[1]);

        //below only call one time when system close
        MediaLib_Destroy();
        return;
}

T_VOID play_media(char* filaname)
{
        T_S32 fid = 0;
        T_MEDIALIB_STRUCT hMedia;
        T_MEDIALIB_OPEN_INPUT open_input;
        T_MEDIALIB_OPEN_OUTPUT open_output;
        T_AUDIO_DECODE_OUT      AudioDecOut;
        T_VIDEO_DECODE_OUT      VideoDecOut;
        T_MEDIALIB_MEDIA_INFO media_info;
        T_U8 ImgYUV[352*288*2];
        T_U32 preview_time = 0;
        T_U32 sync_time = 0, begin_time = 0, start_system_time = 0;
        T_S32 ret_audio = 0, ret_video = 0;
        T_eMEDIALIB_STATUS player_status;

        fid = FileOpen("/test.3gp");
        if(fid <= 0)
        {
                printf("open file failed\r\n");
                return;
        }
```

```
open_input.m_OpenType = MEDIALIB_OPEN_PLAY;
open_input.m_MediaType = MEDIALIB_MEDIA_UNKNOWN;
open_input.m_hMediaSource = fid;
open_input.m_Scale = MEDIALIB_SCALE_1X;
open_input.m_Rotate = MEDIALIB_ROTATE_0;
open_input.m_AudioAttribute = 0;
open_input.m_bCapabilityTest = AK_FALSE;
open_input.m_bAVDiffTask = AK_FALSE;
open_input.m_AudioOutInfo.m_SampleRate = 44100;
open_input.m_AudioOutInfo.m_Channels = 1;
open_input.m_AudioOutInfo.m_BitsPerSample = 16;
open_input.m_VideoOutInfo.m_OutWidth = 0;
open_input.m_VideoOutInfo.m_OutHeight = 0;

open_cb_func_init(&(open_input.m_CBFunc));     //initial callback function pointer;

hMedia = MediaLib_Open(&open_input, &open_output);

if (AK_NULL == hMedia)
{
        FileClose(fid);
        return;
}

if (MediaLib_GetInfo(hMedia, &media_info) == AK_FALSE)
{
        MediaLib_Close(hMedia);
        FileClose(fid);
        return;
}
```

```
        MediaLib_ReleaseInfoMem(hMedia);

        if (media_info.m_bHasVideo)
        {
                preview_time = media_info.m_ulTotalTime_ms * 30 / 100;
                if (MediaLib_Preview(hMedia, ImgYUV, preview_time) == AK_TRUE)
                {
                        display(ImgYUV, media_info.m_VideoInfo.m_uWidth,
media_info.m_VideoInfo.m_uHeight);
                }

                VideoDecOut.m_ulSize = open_output.m_ulVideoDecBufSize;
        }
        else
        {
                VideoDecOut.m_pBuffer = AK_NULL;
                VideoDecOut.m_ulSize = 0;
        }

        if (media_info.m_bHasAudio)
        {
                AudioDecOut.m_pBuffer = malloc(open_output.m_ulAudioDecBufSize);
                if (AK_NULL == AudioDecOut.m_pBuffer)
                {
                        MediaLib_Close(hMedia);
                        free(VideoDecOut.m_pBuffer);
                        FileClose(fid);
                        return;
                }
                AudioDecOut.m_ulSize = open_output.m_ulAudioDecBufSize;
        }
        else
```

```
        {
                AudioDecOut.m_pBuffer = AK_NULL;
                AudioDecOut.m_ulSize = 0;
        }

        begin_time = MediaLib_SetPosition(hMedia, 5000, AK_FALSE);    //play from 00:00:05, maybe can't seek there
        if (begin_time < 0)
        {
                MediaLib_Close(hMedia);
                free(AudioDecOut.m_pBuffer);
                FileClose(fid);
                return;
        }

        begin_time = MediaLib_Play(hMedia);
        if (begin_time < 0)
        {
                MediaLib_Close(hMedia);
                free(VideoDecOut.m_pBuffer);
                free(AudioDecOut.m_pBuffer);
                FileClose(fid);
                return;
        }

        if (media_info.m_bHasAudio && media_info.m_bHasVideo)//play with audio and video
        {
                while (1)
                {
                        //return audio output data length
                        ret_audio = MediaLib_DecodeAudioPack(hMedia, &AudioDecOut);
                        if (ret_audio > 0)
```

```
                        {
                                //audio output
                                audio_data_to_da(AudioDecOut.m_pBuffer,
AudioDecOut.m_ulDecDataSize);
                        }
                        sync_time = begin_time + get_audio_time_ms();    //from 00:00:00
                        ret_video = MediaLib_DecodeVideoPack(hMedia, &VideoDecOut,
sync_time);
                        if (ret_audio < 0 || ret_video < 0)
                        {
                                player_status = MediaLib_GetStatus(hMedia);
                                if (MEDIALIB_ERR == player_status)
                                {
                                        printf("error\r\n");
                                        break;
                                }
                                else if (MEDIALIB_END == player_status)
                                {
                                        printf("end\r\n");
                                        break;
                                }
                        }
                        //display video
                        display(VideoDecOut.m_pBuffer, VideoDecOut.m_uDispWidth,
VideoDecOut.m_uDispHeight);
                }
        }
        else if (media_info.m_bHasVideo)//only video
        {
                start_system_time = get_system_time_ms();
                while (1)
                {
```

```
                        sync_time = (get_system_time_ms() - start_system_time) + begin_time;
                        ret_video = MediaLib_DecodeVideoPack(hMedia, &VideoDecOut,
sync_time);
                        if (ret_video < 0)
                        {
                                player_status = MediaLib_GetStatus(hMedia);
                                if (MEDIALIB_ERR == player_status)
                                {
                                        printf("error\r\n");
                                        break;
                                }
                                else if (MEDIALIB_END == player_status)
                                {
                                        printf("end\r\n");
                                        break;
                                }
                        }
                        //display video
                        display(VideoDecOut.m_pBuffer, VideoDecOut.m_uDispWidth,
VideoDecOut.m_uDispHeight);
                }
        }
        else//only audio
        {
                while (1)
                {
                        //return audio output data length
                        ret_audio = MediaLib_DecodeAudioPack(hMedia, &AudioDecOut);
                        if (ret_audio < 0)
                        {
                                player_status = MediaLib_GetStatus(hMedia);
                                if (MEDIALIB_ERR == player_status)
```

```
                                {
                                        printf("error\r\n");
                                        break;
                                }
                                else if (MEDIALIB_END == player_status)
                                {
                                        printf("end\r\n");
                                        break;
                                }
                        }
                        //audio output
                        audio_data_to_da(AudioDecOut.m_pBuffer,
AudioDecOut.m_ulDecDataSize);
                }
        }

        MediaLib_Close(hMedia);

        free(VideoDecOut.m_pBuffer);
        free(AudioDecOut.m_pBuffer);
        FileClose(fid);
        return;
}
```

4.5 注意事项

1）MediaLib_SetFastSpeed、MediaLib_FastForward、MediaLib_FastRewind、MediaLib_Preview、MediaLib_SetDispRotat 函数，对于纯音频媒体资源播放时是不起作用的。

2）MediaLib_GetAudioDecTime 对于无音频的媒体资源播放时不起作用。

5 依赖接口说明

从 3.1 层次关系可以看出媒体播放库所依赖的外部接口主要有资源管理、内存管理及其他一些媒体播放库需要的功能接口，该类函数由目标平台实现。参见《音视频库总体使用说明》。

6 常见问题

6.1 播放常见问题

1）调用 MediaLib_Open 失败

检查是否成功调用 MediaLib_Init；该文件不是媒体文件或暂时不支持此类文件的播放。

2）播放时视频画面一直没变化

请确定音视频同步解码函数被不断调用，该函数要求约 30ms 被调用一次。

3）播放低帧率（<5fps）的媒体资源，音视频同步效果差

这属于正常现象。由于 fps 太低，音视频同步点太少导致。

4）定位误差偏大，例如定位到第 10 秒播放，实际从第 9 秒开始播放

这属于正常现象。如果定位指定的位置不是一个视频关键帧，根据视频压缩的特性必须从之前最接近的一个关键帧开始依次每帧解码才能得到指定位置的视频帧。不同媒体资源关键帧的分布差别很大，从关键帧依次解码耗费的时间可能很长造成响应时间过长，为此只能从最接近的关键帧初开始播放。

5）某些媒体资源不能快进/快退，或者快进/快退马上就结束

这属于正常现象。某些媒体资源只有很少的视频关键帧，甚至某些 3GP 媒体只有一个关键帧。快进/快退是播放媒体中的关键帧，所以产生该现象。可以通过媒体信息得知文件是否可以快进/快退/定位，见 T_MEDIALIB_MEDIA_INFO。

6）媒体播放库没有提供音量控制功能

由于系统通常需要综合其他事件控制音频播放，媒体播放库不控制音量，可调用音频库的音量控制 API 或驱动提供的 API 实现。请参考音频库的文档或驱动相关文档。

7）画面的缩放和旋转功能暂由 LCD 实现。

8）文件不能播放

* 文件本身损坏

* 文件格式不支持

* 视频编码方式不支持或比特率等特性超出支持范围

* 音频编码方式不支持或采样率等特性被限制

* 内存不足，分配失败

9）播放不同步或音频卡

一般码率或帧率较大时会出现，原因是芯片处理能力不够

10）播放到中途退出

* 解码器报错

* 读文件失败或 SD 卡被拔出

11）音频输出异常

可能是码流有误或上层设置错误的采样率

12）总时间不准确

VBR 特性的 mp3/ac3 等纯音频文件或文件被损坏

13）视频显示异常

* 图像严重花屏，可能是码流本身有误

* 图像画面块效应严重，可能画面质量本身不好，或是 RM 文件芯片无滤波功能

6.2 其它问题

参见《音视频库总体使用说明》。