ANYKA
芯 随 你 动

版本: **1.0.2** **2015** 年 **01** 月

媒体录制库接口说明

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
V1.0.0	正式发布	2011 年 12 月
V1.0.1	更新描述	2011 年 12 月
V1.0.2	增加MediaLib_Rec_SetEncQP接口	2015 年 1 月

媒体录制库接口说明与媒体录制库对应的关系

Version	Corresponding media lib	Corresponding video driver lib	Corresponding audio driver lib
V1.0.2	V1.16.09	V1.10.02	V1.11.01
V0.3.1	V1.11.02	V1.7.09	V1.7.06
V0.2.2	V0.10.1	V0.7.4	V0.7.1
V0.2.1	V0.9.1	-	V0.7.0
V0.2.0	V0.8.0	V0.7.0	V0.6.0
V0.1.2	V0.5.3	V0.6.2	V0.3.3
V0.1.1	V0.4.0	V0.4.0	
V0.1.0	V0.3.0	V0.3.0	V0.2.0

目录

1 模块介绍

1.1 功能概述

本模块支持录像。

- 支持音频为 PCM 的 AVI 录像，视频编码方式可选 MPEG4/H263/MJPEG
- 音频为 AMR 的 3gp 录像，视频编码方式可选 MPEG4/H263
- 暂不支持音频为 ADPCM 的 AVI 录像

注：视频编码方式依赖于芯片的支持情况。

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

如果是首次使用视频驱动库，请首先阅读《音视频库总体使用说明》。

3.1 层次结构

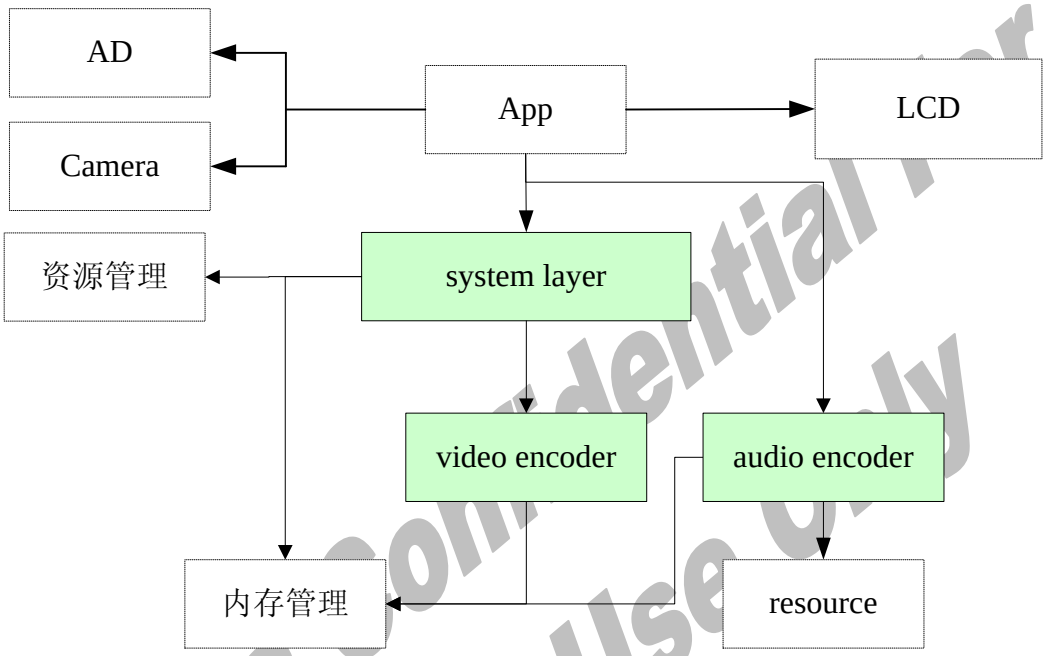

图 3-1 模块关系图

媒体录制库包含系统层库、视频编码库、音频编码库，集成后在系统中的位置如上图所示。媒体录制库依赖的模块都应该在使用前由系统初始化。App 应用模块实现录像过程显示界面，调用系统层模块、控制 LCD 实现显示；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。箭头标明在运行录像过程中模块的调用情况。

注意：与以往的音视频库不同的是，修改后的媒体录制库在视频编码后再解码的数据不再由媒体录制库调用 LCD 进行视频的输出，而是统一由 App 应用模块实现视频的输出。

3.2 集成步骤

3.2.1 获得媒体录制库相关文件

媒体录制库包括库二进制文件、头文件和相关文档。

3.2.2 实现媒体录制库依赖的系统API

请仔细阅读本文的第五部分，准备好媒体录制库所依赖的系统 API，为系统控制媒体录制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

把媒体录制库、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，调试和使用媒体录制库常见问题请参考本文的相关部分。

以录制 avi 文件为例：

1、调试录制无声视频，确定系统层和视频编码内部录像基本正常运作。

2、调试录制音频文件，确定系统层和音频编码内部录音基本正常运作。

3、调试录制有声视频，确定系统层、视频编码和音频编码相关接口正常运作。

4、调试其它类型媒体的有声录像，如 3gp 等，确定媒体录制库正常运作。

3.3 注意事项

1、存储空间满等判断需要调用者实现。

2、音频编码需要调用者实现。

4 接口说明

4.1 模块功能概述

本模块起录像的作用，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 节，依赖接口说明。

4.2.2 初始化结构体

参见《音视频库总体使用说明》。

4.2.3 媒体录制句柄

typedef T_pVOID T_MEDIALIB_STRUCT;

用于媒体录制，存放媒体录制库的所有信息，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

```
typedef T_BOOL (*MEDIALIB_EXFUNC_YUVENC)(T_U8 *srcYUV,     //输入 YUV 地址
        T_U8 *dstStream,        //输出码流地址
        T_U32 *pSize,           //输入 buf 大小，输出码流长度
        T_U32 pic_width,        //图像宽
        T_U32 pic_height,       //图像高
        T_U32 quality);         //编码质量
typedef struct
{
        T_eMEDIALIB_REC_TYPE            m_MediaRecType;
        T_S32                           m_hMediaDest;
```

```
    T_S32                       m_hIndexFile;
    T_AUDIO_RECORD_INFO         m_AudioRecInfo;
    T_VIDEO_RECORD_INFO         m_VideoRecInfo;
    T_MEDIALIB_CB               m_CBFunc;
    T_BOOL      m_bCaptureAudio;
    T_BOOL      m_bIdxInMem;    //flag indicating Index is saved in memory
    T_BOOL      m_bHighQuality;
    T_U32  m_IndexMemSize;      //index size set by system
    T_U16  m_SectorSize;   //add for a sector size of file system, no greater than 8192;
    T_U16  m_RecordSecond;      //add for time limit record
    MEDIALIB_EXFUNC_YUVENC m_ExFunEnc;//encode function, such as yuv2jpeg
    MEDIALIB_EXFUNC_SETENCTASK  m_ExFunSetEncTask;
    T_S32  m_hTmpFile;          //add for time limit record
    T_S32  m_Reserved;          //reserved
} T_MEDIALIB_REC_OPEN_INPUT;
```

结构名	T_MEDIALIB_REC_OPEN_INPUT	
定义概述	打开类型	
成员说明	m_MediaRecType	录像媒体的类型
	m_hMediaDest	录像媒体文件句柄
	m_hIndexFile	如果 m_bIdxInMem 为 false，本变量为临时存放 index 的文件句柄
	m_AudioRecInfo	录音后信息
	m_VideoRecInfo	录像视频信息
	m_CBFunc	回调函数结构体
	m_bCaptureAudio	是否带录音
	m_bIdxInMem	索引是否在内存中
	m_bHighQuality	是否高清录像

	m_IndexMemSize	如果 m_bIdxInMem 为 true，本变量表示存放 index 的空间大小，以字节为单位； 当 m_MediaRecType 为 MEDIALIB_REC_AVI_NORMAL 时： 如果 m_IndexMemSize 为 0 表示不写 index，录制出的 AVI 文件无 index 信息； 如果 m_IndexMemSize 为 0XFFFFFFFF（即(T_U32)-1），表示在录制过程中不写入 index，而在录制结束时生成 index 信息到文件最后；注意：选择此方式时文件保存过程会比较长。
	m_SectorSize	文件系统 sector 的 size，如：2048、4096
	m_RecordSecond	选择录制循环录像时，设置需保存的录制时间，仅在 m_MediaRecType 为 MEDIALIB_REC_AVI_CYC 时有效
	m_ExFunEnc	如果不选择内部编码方式，需要输入外部编码函数指针，如 motion jpeg 编码的函数
	m_ExFunSetEncTask	选择外部编码（m_ExFunEnc 非空）且需要编码与写文件并行时，传入设置并行回调的参数；如 jpeg 编码库的 Img_SetJPEGTaskFunc 函数；
	m_hTmpFile	选择录制循环录像时，临时文件的句柄，仅在 m_MediaRecType 为 MEDIALIB_REC_AVI_CYC 时有效
	m_Reserved	保留


```
typedef struct
{
    T_U32  m_Type;              //音频编码类型
    T_U16  m_nChannel;          //立体声(2)、单声道(1)
    T_U16  m_BitsPerSample;     //16 bit 固定(16)
    T_U32  m_nSampleRate;       //采样率(8000)
```

```
    T_U16  m_ulDuration;            //每个音频包持续时间
}T_AUDIO_RECORD_INFO;
```

结构名	T_AUDIO_RECORD_INFO	
定义概述	录音输入参数	
成员说明	m_Type	录音音频媒体类型
	m_nChanne	声道数
	m_BitsPerSample	采样位数
	m_nSampleRate	采样率
	m_ulDuration	每个音频包持续时间

```
typedef enum
{
    MEDIALIB_REC_AVI_NORMAL,        //录像保存为 avi 文件格式
    MEDIALIB_REC_AVI_CYC,           //avi 循环录像模式，目前已不提供
    MEDIALIB_REC_3GP,               //录像保存为 3gp 文件格式
    MEDIALIB_REC_AVI_SEGMENT        //分段录像模式，目前不提供
}T_eMEDIALIB_REC_TYPE;

typedef enum
{
    MEDIALIB_V_ENC_H263,            //视频编码方式为 H263
    MEDIALIB_V_ENC_MJPG,            //视频编码方式为 Motion JPEG
    MEDIALIB_V_ENC_MPEG             //视频编码方式为 MPEG4
}T_eMEDIALIB_V_ENC_TYPE;

typedef struct
{
    T_U16  m_nWidth;
```

```
    T_U16  m_nHeight;
    T_U16  m_nFPS;
    T_U16  m_nKeyframeInterval;
    T_U32  m_nvbps;
    T_eMEDIALIB_V_ENC_TYPE  m_eVideoType;
}T_VIDEO_RECORD_INFO;
```

结构名	T_VIDEO_RECORD_INFO	
定义概述	录像输入参数	
成员说明	m_nWidth	宽
	m_nHeight	高
	m_nFPS	帧率
	m_nKeyframeInterval	关键帧周期
	m_nvbps	视频期望比特速率
	m_eVideoType	视频编码方式

4.2.5 媒体录像状态

```
typedef enum
{
    MEDIALIB_REC_OPEN,
    MEDIALIB_REC_STOP,
    MEDIALIB_REC_DOING,
    MEDIALIB_REC_SYSERR,
    MEDIALIB_REC_MEMFULL,
    MEDIALIB_REC_SYNERR
}T_eMEDIALIB_REC_STATUS;
```

结构名	T_eMEDIALIB_REC_STATUS
定义概述	录像状态信息

结构名	T_eMEDIALIB_REC_STATUS	
成员说明	MEDIALIB_REC_OPEN	打开状态
	MEDIALIB_REC_STOP	停止状态
	MEDIALIB_REC_DOING	正在录制
	MEDIALIB_REC_SYSERR	系统错误，如写资源出错等
	MEDIALIB_REC_MEMFULL	内存满或到达最大文件限制（如 2G bytes）
	MEDIALIB_REC_SYNERR	同步错误

4.2.6 媒体打开输出结构体

```
typedef struct
{
        T_eMEDIALIB_STATE       m_State;
        T_U32           m_ulAudioEncBufSize;  //音频编码缓冲区的期望大小
        T_U32           m_ulVideoEncBufSize;  //视频编码缓冲区的期望大小
}T_MEDIALIB_REC_OPEN_OUTPUT;
```

结构名	T_MEDIALIB_REC_OPEN_OUTPUT	
定义概述	打开时输出参数	
成员说明	m_State	见 T_eMEDIALIB_STATE
	m_ulAudioDecBufSize	音频编码缓冲区的期望大小，以字节为单位
	m_ulVideoDecBufSize	视频编码缓冲区的期望大小，以字节为单位

4.2.7 媒体信息结构体

T_eMEDIALIB_REC_STATUS 见 4.1.2.5。

```
typedef struct
{
        //fix
        T_U16           ori_width;
```

```
    T_U16       ori_height;
    T_U16       fps;
    T_U16       keyframeInterval;
    T_BOOL      bCaptureAudio;
    T_U16       record_second;          //add for time limit record
    //dynamic
    T_eMEDIALIB_REC_STATUS record_status;
    T_U32  total_frames;
    T_U32  total_video_frames;
    T_U32  info_bytes;
    T_U32  file_bytes;              //current file size
    T_U32  total_time_ms;           //record time in millisecond
}T_MEDIALIB_REC_INFO;
```

结构名	T_MEDIALIB_REC_INFO	
定义概述	录像信息	
成员说明	ori_width	宽
	ori_height	高
	fps	帧率
	keyframeInterval	关键帧周期
	bCaptureAudio	是否录音，1：录音，0：不录音
	record_second	循环录像设置的录制时间
	record_status	录像状态
	total_frames	音视频总帧数
	total_video_frames	视频总帧数
	info_bytes	文件头字节数
	file_bytes	当前文件占用字节数
	total_time_ms	当前录制的总时间，以毫秒为单位

4.2.8 视频码流信息结构体

```
typedef struct
{
        T_pDATA     pVideoData;
        T_U32       ulVideoLen;
        T_BOOL      bKeyframe;
        T_U32       ulTimestamp;
}T_MEDIALIB_REC_VIDEO_OUT;
```

结构名	T_MEDIALIB_REC_VIDEO_OUT	
定义概述	录像信息	
成员说明	pVideoData	视频码流地址
	ulVideoLen	视频码流长度
	bKeyframe	是否关键帧
	ulTimestamp	时间戳

4.2.9 分段录像参数结构体

```
typedef enum
{
        MEDIALIB_REC_EV_NORMAL,
        MEDIALIB_REC_EV_EMERGENCY,
        MEDIALIB_REC_EV_NOTHING
}T_eMEDIALIB_REC_EVENT;
```

结构名	T_eMEDIALIB_REC_EVENT	
定义概述	重新录制的模式	
成员说明	MEDIALIB_REC_EV_NORMAL	普通状况
	MEDIALIB_REC_EV_EMERGENCY	紧急状况
	MEDIALIB_REC_EV_NOTHING	不保存录像状况

4.2.10 录像质量参数设置结构体

```
typedef struct
{
        T_U8    enc_QP;
        T_U8    p_enc_QP;
        T_U8    MJPEG_Quality;
}T_MEDIALIB_REC_ENCQP_PAR;
```

结构名	T_MEDIALIB_REC_ENCQP_PAR	
定义概述	设置录像的质量参数	
成员说明	enc_QP	编码 I 帧的质量参数 QP 值(1-31)
	p_enc_QP	编码 P 帧的质量参数 QP 值(1-31)
	MJPEG_Quality	MJPEG 编码图像质量参数 quality 值(0-200)

在设置结构体 T_MEDIALIB_REC_ENCQP_PAR 参数时，要求：

1、 对于 H263 或 MPEG4 编码，需要按需求设置 enc_QP 和 p_enc_QP 值，而 MJPEG_Quality 是用于 MJPEG 编码使用的，只需要设置为有效值，值在范围(0-200)内即可，库内不会实际使用。

2、 对于 MJPEG 编码时，需要按需求设置 MJPEG_Quality 的值，enc_QP 和 p_enc_QP 只需要设置在有效范围内的值即可。

3、 对于 H263 或 MPEG4 编码，若其是全 I 帧编码，则需要按要求设置 enc_QP 的值，p_enc_QP 和 MJPEG_Quality 只需要保证在有效范围内即可。

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_Rec_Open

原　　型	T_MEDIALIB_STRUCT MediaLib_Rec_Open(const T_MEDIALIB_REC_OPEN_INPUT *rec_open_input, T_MEDIALIB_REC_OPEN_OUTPUT *rec_open_output)	
功能概述	初始化录制器，并申请相关资源	
参数说明	open_input	见 T_MEDIALIB_REC_OPEN_INPUT
	open_output	见 T_MEDIALIB_REC_OPEN_OUTPUT
返回值说明	返回打开相关资源的录像句柄	
返回值说明	AK_NULL：打开失败； 其它：打开成功	
注意事项	-	
调用示例	见典型调用示例	

4.3.2 MediaLib_Rec_Close

原　　型	T_BOOL MediaLib_Rec_Close(T_MEDIALIB_STRUCT hMedia)	
功能概述	关闭录制器，释放资源	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录像句柄
返回值说明	返回关闭录制器的结果	
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_Rec_GetInfo

原　　型	T_BOOL MediaLib_Rec_GetInfo(T_MEDIALIB_STRUCT hMedia, T_MEDIALIB_REC_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_Rec_Open 时返回的媒体录像句柄
	pInfo	存放媒体信息的数据结构的指针

原　　型	T_BOOL MediaLib_Rec_GetInfo(T_MEDIALIB_STRUCT hMedia, T_MEDIALIB_REC_INFO *pInfo)
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_Rec_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

4.3.4 MediaLib_Rec_Start

原　　型	T_BOOL MediaLib_Rec_Start(T_MEDIALIB_STRUCT hMedia)	
功能概述	从当前位置启动录像	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
返回值说明	返回启动的结果	
返回值说明	AK_TRUE　　启动成功 AK_FALSE　　启动失败	
注意事项	必须已经成功执行 MediaLib_Rec_Open 函数	
调用示例	见典型调用示例	

4.3.5 MediaLib_Rec_Stop

原　　型	T_BOOL MediaLib_Rec_Stop(T_MEDIALIB_STRUCT hMedia)	
功能概述	停止当前的录象	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
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_Rec_Open 函数	
调用示例	见典型调用示例	

4.3.6 MediaLib_Rec_GetStatus

原　　型	T_eMEDIALIB_REC_STATUS MediaLib_Rec_GetStatus(T_MEDIALIB_STRUCT hMedia)	
功能概述	获得当前录制的状态信息	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
返回值说明	媒体录制的状态信息	
返回值说明	见 T_eMEDIALIB_REC_STATUS	
注意事项	必须已经成功执行 MediaLib_Rec_Open 函数。	
调用示例	见典型调用示例	

4.3.7 MediaLib_Rec_ProcessAudio or MediaLib_Rec_WriteAudio

原　　型	T_BOOL MediaLib_Rec_ProcessAudio(T_MEDIALIB_STRUCT hMedia, T_U8 *pAudioData, T_U32 ulAudioSize) T_BOOL MediaLib_Rec_WriteAudio(T_MEDIALIB_STRUCT hMedia, T_U8 *pAudioData, T_U32 ulAudioSize)	
功能概述	将已编码的音频数据写入文件	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	pAudioData	音频编码后数据的指针
	ulAudioSize	音频编码后数据大小
返回值说明	编码音频数据并写入文件操作的结果	
返回值说明	AK_TRUE　处理成功 AK_FALSE　处理失败	
注意事项	1. 必须已经成功执行 MediaLib_Rec_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	

4.3.8 MediaLib_Rec_ProcessVideo

原　　型	T_S32 MediaLib_Rec_ProcessVideo(T_MEDIALIB_STRUCT hMedia, T_U8 *pVideoData, T_S32 ulMilliSec)
功能概述	将捕获来的视频数据编码，写入文件，并返回视频时间

原　　型	T_S32 MediaLib_Rec_ProcessVideo(T_MEDIALIB_STRUCT hMedia, T_U8 *pVideoData, T_S32 ulMilliSec)	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	pVideoData	捕获的 YUV 视频数据指针
	ulMilliSec	音频时间或系统时间，以毫秒为单位
返回值说明	视频时间	
返回值说明	返回负值是出错，可根据 status 获得状态区分	
注意事项	1. 必须已经成功执行 MediaLib_Rec_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	

4.3.9 MediaLib_Rec_EncodeVideo

原　　型	T_S32 MediaLib_Rec_EncodeVideo(T_MEDIALIB_STRUCT hMedia, T_U8 *pVideoData, T_S32 ulMilliSec)	
功能概述	将捕获来的视频数据编码	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	pVideoData	捕获的 YUV 视频数据指针
	ulMilliSec	音频时间或系统时间，以毫秒为单位
返回值说明	视频时间	
返回值说明	返回负值是出错，可根据 status 获得状态区分； 返回正值编码成功； 返回 0 表明未编码，根据视频时间与 ulMilliSec 的差值决定	
注意事项	1. 必须已经成功执行 MediaLib_Rec_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. 调用完毕后，如果返回正值，必须调用 MediaLib_Rec_WriteVideo 函数进行写文件	
调用示例	见典型调用示例	

4.3.10 MediaLib_Rec_WriteVideo

原　　型	T_S32 MediaLib_Rec_WriteVideo(T_MEDIALIB_STRUCT hMedia)	
功能概述	根据 MediaLib_Rec_EncodeVideo 的结果写入视频数据到文件	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
返回值说明	视频时间	
返回值说明	返回负值是出错，可根据 status 获得状态区分	
注意事项	1. 必须已经成功执行 MediaLib_Rec_Open 函数。 2. 必须在 MediaLib_Rec_EncodeVideo 正常执行完之后尽快调用，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	

4.3.11 MediaLib_Rec_GetVideo

原　　型	T_S32 MediaLib_Rec_GetVideo (T_MEDIALIB_STRUCT hMedia, T_MEDIALIB_REC_VIDEO_OUT *pRecVideoOut)	
功能概述	获取 MediaLib_Rec_EncodeVideo 编码出的视频码流	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	pRecVideoOut	见 T_MEDIALIB_REC_VIDEO_OUT 结构体定义
返回值说明	获取是否成功	
返回值说明	负值：获取失败； 0：没有已编码出的视频帧可获取； 正值：获取成功；	
注意事项	1. 必须已经成功执行 MediaLib_Rec_Open 函数。 2. 必须在 MediaLib_Rec_EncodeVideo 正常执行完且 MediaLib_Rec_WriteVideo 执行前调用才有效。	
调用示例	T_MEDIALIB_REC_VIDEO_OUT RecVideoOut; MediaLib_Rec_WriteVideo (hMedia, &RecVideoOut);	

4.3.12 MediaLib_Rec_Restart

原　　型	T_BOOL MediaLib_Rec_Restart(T_MEDIALIB_STRUCT hMedia, T_S32 hFile, T_eMEDIALIB_REC_EVENT rec_event)	
功能概述	重新开始录制	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	hFile	新录制文件的句柄
	rec_event	普通录像时忽略该变量，传入任意值皆可；分段录像时见 T_eMEDIALIB_REC_EVENT
返回值说明	Restart 是否成功	
返回值说明	AK_TRUE　　成功 AK_FALSE　　失败	
注意事项	1. 必须已经成功执行 MediaLib_Rec_Open 函数。 2. 必须在前一次录制结束调用过 MediaLib_Rec_Stop 之后。	
调用示例	见典型调用示例	

4.3.13 MediaLib_Rec_SetVideoBitrate

原　　型	T_BOOL MediaLib_Rec_SetVideoBitrate(T_MEDIALIB_STRUCT hMedia, T_U32 ulBitrate)	
功能概述	录制过程中改变视频比特率	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	ulBitrate	新的视频比特率
返回值说明	设置是否成功	
返回值说明	AK_TRUE　　成功 AK_FALSE　　失败	
注意事项	1. 必须已经成功执行 MediaLib_Rec_Open 函数。 2. 必须已经成功执行 MediaLib_Rec_Start 函数。	
调用示例	见典型调用示例	

4.3.14 MediaLib_Rec_SetVideoMaxInsert

原　　型	T_BOOL MediaLib_Rec_SetVideoMaxInsert(T_MEDIALIB_STRUCT hMedia, T_U32 ulMaxInsertNum)	
功能概述	录制开始或录制过程中改变最大插帧数	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	ulMaxInsertNum	最大插帧数，若 ulMaxInsertNum 为 0 时表示不可插帧
返回值说明	设置是否成功	
返回值说明	AK_TRUE　　成功 AK_FALSE　　失败	
注意事项	1. 必须已经成功执行 MediaLib_Rec_Open 函数。 2. 在 MediaLib_Rec_Open 时 I 帧间隔设置为 0 时有效。	
调用示例	最大插帧为 2 秒的帧数：（最坏情况会导致录像有 2 秒的停顿） MediaLib_Rec_SetVideoMaxInsert(hMedia, 2*fps);	

4.3.15 MediaLib_Rec_UpdateIndexFile

原　　型	T_BOOL MediaLib_Rec_UpdateIndexFile(T_MEDIALIB_STRUCT hMedia, T_S32 hIndexFile)	
功能概述	重新开始录制之前更新 index 临时文件句柄	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	hIndexFile	新的 index 临时文件句柄
返回值说明	更新是否成功	
返回值说明	AK_TRUE　　成功 AK_FALSE　　失败	
注意事项	1. 必须已经成功执行 MediaLib_Rec_Open 函数。 2. 在 MediaLib_Rec_Stop 之后且 MediaLib_Rec_Restart 之前调用有效。	
调用示例	hIndexFile = File_Open(filename); MediaLib_Rec_UpdateIndexFile(hMedia, hIndexFile);	

4.3.16 MediaLib_Rec_SetDropFrame

原　　型	T_BOOL MediaLib_Rec_SetDropFrame (T_MEDIALIB_STRUCT hMedia, T_U32 ulDropNum)	
功能概述	设置丢帧状况	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	ulDropNum	每次丢帧数
返回值说明	设置是否成功	
返回值说明	AK_TRUE　　成功 AK_FALSE　　失败	
注意事项	1. 必须已经成功执行 MediaLib_Rec_Open 函数。 2. 在 MEDIALIB_REC_DOING 状态下设置才有效。 3. ulDropNum 不为 0 时，进入丢帧模式，每次丢 ulDropNum 帧后才编码一帧有效帧；直到 ulDropNum 设置为 0 时恢复正常录制模式。	

调用示例	T_U32 DropFrameNum = JudgeDrop();//判定丢帧数，如：根据文件写缓冲的情况决定
	MediaLib_Rec_SetDropFrame (hMedia, DropFrameNum);	

4.3.17 MediaLib_Rec_SetBufferingInfo

原　　型	T_BOOL MediaLib_Rec_SetBufferingInfo(T_MEDIALIB_STRUCT hMedia, T_U32 ulRemainSize)	
功能概述	设置文件写缓冲信息	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	ulRemainSize	文件写缓冲剩余空间
返回值说明	设置是否成功	
返回值说明	AK_TRUE　　成功 AK_FALSE　　失败	
注意事项	1. 必须已经成功执行 MediaLib_Rec_Open 函数。 2. 在 MEDIALIB_REC_DOING 状态下设置才有效。 3. 处于丢帧模式下，ulRemainSize 设置有效；且必须在丢帧模式下根据文件系统反馈情况进行动态更新。	

原　　型	T_BOOL MediaLib_Rec_SetBufferingInfo(T_MEDIALIB_STRUCT hMedia, T_U32 ulRemainSize)

调用示例	T_U32 UseSize = 0, T_U32 BufSize = 0;
	MediaLib_Rec_SetDropFrame (hMedia, 1);
	if (FS_GetAsynBufInfo(&UseSize, &BufSize))//通过文件系统相应函数获取
	{
	　　MediaLib_Rec_SetBufferingInfo(hMedia, BufSize - UseSize);
	}

4.3.18 MediaLib_Rec_SetEncQP

原　　型	T_BOOL MediaLib_Rec_SetEncQP(T_MEDIALIB_STRUCT hMedia, T_MEDIALIB_REC_ENCQP_PAR *encQP_param)	
功能概述	设置编码质量参数	
参数说明	hMedia	MediaLib_Rec_Open 时返回的媒体录制句柄
	encQP_param	设置编码质量参数的结构体指针
返回值说明	设置是否成功	
返回值说明	AK_TRUE　　成功 AK_FALSE　　失败	
注意事项	1. 必须已经成功执行 MediaLib_Rec_Open 函数。 2. 若调用 MediaLib_Rec_SetEncQP()来手动设置编码质量参数值，则在调用 MediaLib_Rec_Open ()时设置的固定码率是无效的。若需要重新使用固定码率的方式，则需要调用 MediaLib_Rec_SetVideoBitrate ()接口。	
调用示例	T_MEDIALIB_REC_ENCQP_PAR　enc_qp_par; MediaLib_Rec_SetEncQP(hMedia, &enc_qp_par);	

4.4 典型调用范例

```
T_VOID record_media();
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_AK8801;
        init_input.m_AudioI2S = I2S_UNUSE;

        if (MediaLib_Init(&init_input, &init_cb) == AK_FALSE)
        {
            return;
        }
        //above only call one time when system start

        //record
        record_media(argv[1]);

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

T_VOID record_media(char *filename)
{
        T_S32 fid;

        T_MEDIALIB_REC_OPEN_INPUT rec_open_input;
        T_MEDIALIB_REC_OPEN_OUTPUT rec_open_output;

        T_eMEDIALIB_REC_STATUS rec_status;
        char press_key;
        char tmp_buf[1600];
        T_MEDIALIB_REC_INFO RecInfo;
```

```
        T_VOID *hMedia;
//      T_pDATA pVideo;
        T_S32 v_time;
        T_U8 pYUV1[352*288*2];
        T_U32 audio_tytes = 0;
        T_U32 fps = 10;
        T_U32 max_seconds = 30*60;// half an hour

        fid = FileOpen(filename);
        if(fid <= 0)
        {
                printf("open file failed\r\n");
                return;
        }

        memset(&rec_open_input, 0, sizeof(T_MEDIALIB_REC_OPEN_INPUT));
        rec_open_input.m_hMediaDest = fid;
        rec_open_input.m_bCaptureAudio = 1;
        rec_open_input.m_bHighQuality = 1;
        rec_open_input.m_bIdxInMem = 1;
        rec_open_input.m_IndexMemSize = (fps+2)*16 * max_seconds;
        rec_open_input.m_RecordSecond = 0;
        rec_open_input.m_MediaRecType = MEDIALIB_REC_AVI_NORMAL;//or
MEDIALIB_REC_3GP;
// set video open info
        rec_open_input.m_VideoRecInfo.m_nFPS = fps;
        rec_open_input.m_VideoRecInfo.m_nWidth = 352;
        rec_open_input.m_VideoRecInfo.m_nHeight = 288;
        rec_open_input.m_VideoRecInfo.m_nKeyframeInterval = 19;
        rec_open_input.m_VideoRecInfo.m_nvbps = 600*1024;
        rec_open_input.m_VideoRecInfo.m_eVideoType = MEDIALIB_V_ENC_MJPG;//or
MEDIALIB_V_ENC_H263;
```

```
        rec_open_input.m_SectorSize = 2048;
        rec_open_input.m_ExFunEnc = VD_EXfunc_YUV2JPEG;//set mjpeg encode function

// set audio open info
        rec_open_input.m_AudioRecInfo.m_BitsPerSample = 16;
        rec_open_input.m_AudioRecInfo.m_nChannel = 1;
        rec_open_input.m_AudioRecInfo.m_nSampleRate = 8000;
        rec_open_input.m_AudioRecInfo.m_ulDuration = 1000;
        rec_open_input.m_AudioRecInfo.m_Type = _SD_MEDIA_TYPE_PCM;

        memset(&rec_open_input.m_CBFunc,0,sizeof(T_MEDIALIB_CB));
        rec_open_input.m_CBFunc.m_FunPrintf =
(MEDIALIB_CALLBACK_FUN_PRINTF)printf;
        rec_open_input.m_CBFunc.m_FunMalloc =
(MEDIALIB_CALLBACK_FUN_MALLOC)VD_Malloc;
        rec_open_input.m_CBFunc.m_FunFree =
(MEDIALIB_CALLBACK_FUN_FREE)VD_Free;
        rec_open_input.m_CBFunc.m_FunRead =
(MEDIALIB_CALLBACK_FUN_READ)VD_FileRead;;
        rec_open_input.m_CBFunc.m_FunSeek =
(MEDIALIB_CALLBACK_FUN_SEEK)VD_FileSeek;
        rec_open_input.m_CBFunc.m_FunTell =
(MEDIALIB_CALLBACK_FUN_TELL)VD_FileGetCurPos;
        rec_open_input.m_CBFunc.m_FunWrite =
(MEDIALIB_CALLBACK_FUN_WRITE)VD_FileWrite;
        rec_open_input.m_CBFunc.m_FunMMUInvalidateDCache = MMU_InvalidateDCache ;
        rec_open_input.m_CBFunc. m_FunCleanInvalidateDcache =
MMU_CleanInvalidateDcache ;

        rec_open_input.m_CBFunc.m_FunMapAddr = VD_EXfunc_MapAddr;
        rec_open_input.m_CBFunc.m_FunUnmapAddr = VD_EXfunc_UnmapAddr;
        rec_open_input.m_CBFunc.m_FunDMAFree =
(MEDIALIB_CALLBACK_FUN_FREE)DMA_free;
```

```
        rec_open_input.m_CBFunc.m_FunDMAMalloc =
(MEDIALIB_CALLBACK_FUN_MALLOC)DMA_malloc;
        rec_open_input.m_CBFunc.m_FunVaddrToPaddr = VD_EXfunc_VaddrToPaddr;
        rec_open_input.m_CBFunc.m_FunRegBitsWrite =
        (MEDIALIB_CALLBACK_FUN_REG_BITS_WRITE)VD_EXfunc_RegBitsWrite;

        hMedia = MediaLib_Rec_Open(&rec_open_input,&rec_open_output);
        if (AK_NULL == hMedia)
        {
                printf("##MOVIE: Media_Open Return NULL\r\n");
                FileClose(fid);
                return;
        }

        if (MediaLib_Rec_GetInfo(hMedia, &RecInfo) == AK_FALSE)
        {
                MediaLib_Rec_Close(hMedia);
                FileClose(fid);
                return;
        }

        if (AK_FALSE == MediaLib_Rec_Start(hMedia))
        {
                MediaLib_Rec_Close(hMedia);
                FileClose(fid);
                return;
        }

        g_MovieRecHandle.m_StartSystemPts = 0;

        buf_change = 0;
```

```
        if (AK_FALSE == MediaLib_Rec_Start(hMedia))
        {
                MediaLib_Rec_Close(hMedia);
                FileClose(fid);
                return;
        }

        if (g_MovieRecHandle.m_MediaInfo.bCaptureAudio)
        {
                _SD_REC_SetBufAddr(rec_buf2, ONCE_REC_LEN);
        }

        VD_Cam_CapStart();

rec_loop:
        g_MovieRecHandle.m_StartSystemPts = get_tick_count();

        MediaLib_Rec_GetStatus(g_MovieRecHandle.m_hMedia);

        //record
        While(1)        //this loop can be replaced by limit times loop,or break while receive
some external signal
                //to avoid the dead loop.
        {
                if (g_MovieRecHandle.m_MediaInfo.bCaptureAudio)
                {
                        Fwl_Printf("Audio recorde start\r\n");

                        //Record audio
                        if (rec_buf_check == 1)
                        {
                                if (buf_change == 0)
```

```
                {

                    _SD_REC_SetBufAddr(rec_buf1,ONCE_REC_LEN);

        if(MediaLib_Rec_ProcessAudio(g_MovieRecHandle.m_hMedia, rec_buf2,
        ONCE_REC_LEN) == AK_FALSE)
                {

                        Fwl_Printf("MediaLib_Rec_ProcessAudio error\r\n");
                        return MOVIE_REC_SYSERR;
                }
                buf_change = 1;
        }
        else
        {

                _SD_REC_SetBufAddr(rec_buf2,ONCE_REC_LEN);

                 if(MediaLib_Rec_ProcessAudio(g_MovieRecHandle.m_hMedi
                 a,rec_buf1, ONCE_REC_LEN) == AK_FALSE)
                {

                        Fwl_Printf("MediaLib_Rec_ProcessAudio error\r\n");
                        return MOVIE_REC_SYSERR;
                }
                buf_change = 0;
        }

        rec_buf_check = 0;
}

//Record video
while (!VD_Cam_CapComplete())
```

```
                {
                        ;
                }
                if (modify_video_bitrate)
                {
                MediaLib_Rec_SetVideoBitrate(g_MovieRecHandle.m_hMedia, new_bitrate);
                }
                pYUV2 = (T_pDATA)VD_Cam_CapGetData();
                VD_Cam_CapStart();
                pYUV1 = pYUV2;
                rec_pts = get_tick_count() - g_MovieRecHandle.m_StartSystemPts;
                v_time = MediaLib_Rec_ProcessVideo(g_MovieRecHandle.m_hMedia, pYUV1,
ec_pts);
                if (v_time < 0)
                {
                        Fwl_Printf("MediaLib_Rec_ProcessVideo error\r\n");
                        break;
                }
        VD_Cam_ShowFrame(pVideo,g_MovieRecHandle.m_MediaInfo.ori_width,
                                g_MovieRecHandle.m_MediaInfo.ori_height);
                                }
        MediaLib_Rec_Stop(g_MovieRecHandle.m_hMedia);
        FileClose(fid)
        fid = FileOpen(filename_new);
        if(fid <= 0)
        {
           MediaLib_Rec_Close(g_MovieRecHandle.m_hMedia);
                printf("open file failed\r\n");
                return;
        }

        if (continue_rec)//record again
```

```
        {
        if (MediaLib_Rec_Restart(g_MovieRecHandle.m_hMedia, fid, 0))
        {
                goto rec_loop;
        }
        }
        MediaLib_Rec_Close(g_MovieRecHandle.m_hMedia);
}
```

在处理音视频数据时有两套实现方案：

1、MediaLib_Rec_ProcessAudio 与 MediaLib_Rec_ProcessVideo 函数配套使用；

2、MediaLib_Rec_WriteAudio 与 MediaLib_Rec_EncodeVideo 及 MediaLib_Rec_WriteVideo 配套使用。

典型调用范例中采用的是方案 1，如果采用方案 2，那么将典型调用范例中的 MediaLib_Rec_ProcessAudio 改为 MediaLib_Rec_WriteAudio；调用 MediaLib_Rec_ProcessVideo 之处用以下代码替代：

```
if (MediaLib_Rec_EncodeVideo(g_MovieRecHandle.m_hMedia, pYUV1, ec_pts) > 0)
{
v_time = MediaLib_Rec_WriteVideo g_MovieRecHandle.m_hMedia);
}
```

5 依赖接口说明

从 3.1 层次关系可以看出媒体录制库所依赖的外部接口主要有资源管理、内存管理及其它一些媒体录制库需要的功能接口，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 录制常见问题

1、调用 MediaLib_Rec_Open 失败

检查是否成功调用 MediaLib_Init；检查各输入参数是否正确，注意观察打印。

2、录像时看到的画面卡

（1）确认输入的视频帧率是否合适，一般来说合理范围是 10－25，根据编码、采集、写文件速度，需要调节帧率到合适的值，太大或太小都容易卡；

（2）检查写文件速度是否均匀，如：某次写入耗时较长。

3、录像时前几帧画面是绿色的

在正式编码写文件前，可预采集几帧不做处理。

4、录像文件不能播放

在录制时已经出错，需要观察录制时的打印。

5、音视频不同步导致录像退出

出现打印：#system time is %d, record time is %d#，表示调用 ProcessVideo 的频率不够；一般是 camera 采集帧率不够或策略性的降低调用 ProcessVideo 频率导致，可以通过 MediaLib_Rec_SetVideoMaxInsert 加大最大插帧数来解决。

6.2 其它问题

参见《音视频库总体使用说明》。