AnyChat 外部编解码器

编程指南

（版本：V2.0）

广 州 佰 锐 网 络 科 技 有 限 公 司

GuangZhou BaiRui Network Technology Co.,Ltd.

http://www.bairuitech.com http://www.anychat.cn

2014 年 06 月

目 录

一、系统概述

AnyChat Platform Core SDK 支持加载外部音视频的编码、解码模块（简称 AnyChat Codec 库），增强系统的可扩展性，同时也满足一些特定应用环境的特殊需求，如与特定的硬件产品集成时，可以调用硬件编解码器，实现更高效率的即时通讯应用。

AnyChat Platform Core SDK 内核已经实现了部分常用的音视频编码、解码模块，如 H.264 视频编码、解码算法，ARM-WB、AAC、SPEEX、CELT 等音频编码、解码算法。上层应用可以自己定义新的音视频编码、解码算法，在 AnyChat 平台初始化时，通过相关 API 接口可以加载新的编码、解码模块，同时上层应用也可以重新实现 SDK 内置的编码、解码算法，在 AnyChat 平台初始化时，发现加载的音视频编码、解码算法与内置的算法相同，则会采用上层应用提供的相关算法来替代内核的算法。

单个音视频编码、解码模块（AnyChat Codec 库）最多可以同时包含：一个音频编码器，一个音频解码器，一个视频编码器以及一个视频解码器。当上层应用需要实现多个编码器，或是多个解码器时，可以在多个库中分别实现，然后分别向 AnyChat 内核进行注册。

AnyChat Platform Core SDK 根据预先定义的 CODEC 编号（CodecId）来区分不同的 CODEC，当用户需要实现的 CODEC 不在已定义的 CODEC 列表中时，可以自行定义新的 CODEC 编号，当 AnyChat 内核收到采用新的 CODEC 编码的数据时，会根据 CODEC 编号查询已注册的新解码器，然后调用新的 CODEC 进行解码。

当上层应用实现的 CODEC 编号与 AnyChat 内核预置的 CODEC 相同时，且输入(或输出)为标准码流时，可以与 AnyChat 内置 CODEC 同步工作(如采用 AnyChat Codec 库进行编码，用 AnyChat 内置库进行解码)。

当上层应用实现的 CODEC 编号不存在于 AnyChat 内置 CODEC 列表中时，需要在编码端、解码端同时注册新的 CODEC，否则将会出现解码失败的情况。

当在编码、解码的过程中出现异常，需要上层重置 CODEC，则可在 OutPacket 上增加标志：OutPacket->dwFlags |= MEDIACODEC_FLAGS_NEEDREINIT

二、接口与应用

2.1 API 接口定义

外部音视频的编码、解码模块提供标准 C 语言接口，上层应用需要按下面的标准实现 AnyChat Codec 库，实现的目标文件是动态库（Windows 平台为 dll 文件，Linux、Android 平台为.so 文件），这样 AnyChat 平台才能正确注册并调用。

```
#if defined(WIN32)
#	if defined(MEDIACODEC_EXPORTS)
#		define MEDIACODEC_API extern "C" __declspec(dllexport)
#	else
#		define MEDIACODEC_API extern "C" __declspec(dllimport)
#	endif
#else
#	define MEDIACODEC_API extern "C"
#endif

// 获取库基本信息
MEDIACODEC_API DWORD MediaCodec_GetLibraryInfo(DWORD* dwCapability, DWORD* dwMainVer, DWORD* dwSubVer, CHAR* lpCompileTime, DWORD dwBufLen);
// 获取编解码器基本信息
MEDIACODEC_API DWORD MediaCodec_GetCodecInfo(DWORD dwCodecType, DWORD* dwCodecId, CHAR* lpCodecName, DWORD dwBufLen);

// 打开音频编码器
MEDIACODEC_API DWORD MediaCodec_AudioCodec_InitEncoder(MediaCodecContext * lpContext);
// 对音频进行编码
MEDIACODEC_API DWORD MediaCodec_AudioCodec_Encode(MediaCodecContext* lpContext, MediaPacket* InPacket, MediaPacket* OutPacket);
// 关闭音频解码器
MEDIACODEC_API DWORD MediaCodec_AudioCodec_CloseEncoder(MediaCodecContext * lpContext);

// 打开音频解码器
MEDIACODEC_API DWORD MediaCodec_AudioCodec_InitDecoder(MediaCodecContext * lpContext);
// 对音频流进行解码
MEDIACODEC_API DWORD MediaCodec_AudioCodec_Decode(MediaCodecContext* lpContext,
```

```
MediaPacket* InPacket, MediaPacket* OutPacket);
// 关闭音频解码器
MEDIACODEC_API DWORD MediaCodec_AudioCodec_CloseDecoder(MediaCodecContext *
lpContext);

// 打开视频编码器
MEDIACODEC_API DWORD MediaCodec_VideoCodec_InitEncoder(MediaCodecContext *
lpContext);
// 对视频进行编码
MEDIACODEC_API DWORD MediaCodec_VideoCodec_Encode(MediaCodecContext* lpContext,
MediaPacket* InPacket, MediaPacket* OutPacket);
// 关闭视频编码器
MEDIACODEC_API DWORD MediaCodec_VideoCodec_CloseEncoder(MediaCodecContext *
lpContext);

// 打开视频解码器
MEDIACODEC_API DWORD MediaCodec_VideoCodec_InitDecoder(MediaCodecContext *
lpContext);
// 对视频流进行解码
MEDIACODEC_API DWORD MediaCodec_VideoCodec_Decode(MediaCodecContext* lpContext,
MediaPacket* InPacket, MediaPacket* OutPacket);
// 关闭视频解码器
MEDIACODEC_API DWORD MediaCodec_VideoCodec_CloseDecoder(MediaCodecContext *
lpContext);
```

其中获取库基本信息（MediaCodec_GetLibraryInfo）和获取编解码器基本信息（MediaCodec_GetCodecInfo）两个 API 是必须要实现的，其它的 API 可根据实际的需要来实现，如上层应用只需要实现硬件的视频编码与解码，则音频的编码、解码 API 可以不用实现。

2.2 数据结构定义

```
// 库包含编解码器能力定义
#define MEDIACODEC_CAPABILITY_AUDIOENCODEC   0x00000001      ///< 音频编码器
#define MEDIACODEC_CAPABILITY_AUDIODECODEC   0x00000002      ///< 音频解码器
#define MEDIACODEC_CAPABILITY_VIDEOENCODEC   0x00000004      ///< 视频编码器
#define MEDIACODEC_CAPABILITY_VIDEODECODEC   0x00000008      ///< 视频解码器
#define MEDIACODEC_CAPABILITY_YUV420SPINPUT  0x00000100      ///< 视频编码器支持
```

```
YUV420SP 数据输入
#define MEDIACODEC_CAPABILITY_YUV420SPOUTPUT 0x00000200        ///< 视频解码器输出数据为 YUV420SP
#define MEDIACODEC_CAPABILITY_DIRECTRENDER    0x00000400        ///< 支持直接视频显示，硬件渲染

// 功能标志定义
#define MEDIACODEC_FLAGS_KEYFRAME    0x00000001   ///< 关键帧标志
#define MEDIACODEC_FLAGS_WANTSPSPPS  0x00000002  ///< 上层应用处理 SPS、PPS 标志，视频编码使用
#define MEDIACODEC_FLAGS_USEARMV6    0x00000100   ///< 强制使用 ARMv6 指令集
#define MEDIACODEC_FLAGS_NEEDREINIT  0x00000200   ///< 需要重新初始化 Codec

#define MEDIACODEC_MAXCODECNAME                    50

// 编解码器 ID 定义
enum MEDIA_CODEC_ID {
    MEDIA_CODEC_ID_NONE = 0,
    /* video codecs */
    MEDIA_CODEC_ID_H264,
    MEDIA_CODEC_ID_MJPEG,
    /* audio codecs */
    MEDIA_CODEC_ID_AMR_NB = 10,
    MEDIA_CODEC_ID_AMR_WB,
    MEDIA_CODEC_ID_MP3,
    MEDIA_CODEC_ID_AAC,
    MEDIA_CODEC_ID_MP2,
    MEDIA_CODEC_ID_CELT,
    MEDIA_CODEC_ID_SPEEX,
    /* user define codecs*/
    MEDIA_CODEC_ID_USERSTART = 100,              ///< 用户自定义编解码器起始 ID
};

// 音视频数据包结构
typedef struct MediaPacket {
    CHAR     *lpData;
    DWORD   dwSize;
    DWORD   dwFlags;
    DWORD   dwTimeStamp;
}MediaPacket,*LPMediaPacket;

// 编解码器上下文句柄结构
struct MediaCodecContext {
    DWORD   cbSize;                    ///< 结构体大小
```

```
    DWORD   dwCodecId;              ///< 编解码器 ID
    DWORD   dwFlags;                ///< 相关标志
    CHAR    szCodecName[MEDIACODEC_MAXCODECNAME];///< 编解码器的名称

    // 编解码器私有数据
    void*   lpPrivateData;          ///< 私有数据指针

    // 音频部分参数
    DWORD   dwChannels;             ///< 音频通道数
    DWORD   dwSamplesPerSec;        ///< 音频采样率
    DWORD   dwBitsPerSample;        ///< 音频量化位数
    DWORD   dwFrameSize;            ///< 音频编码器处理帧长

    // 视频部分
    DWORD   dwWidth;                ///< 视频宽度
    DWORD   dwHeight;               ///< 视频高度
    DWORD   dwFrameRate;            ///< 帧率
    DWORD   dwGopSize;              ///< 关键帧间隔
    AC_PIX_FMT  PixFmt;             ///< 视频帧格式

    // 编码设置
    DWORD   dwBitrate;              ///< 目标码率，单位：bps
    DWORD   dwQuality;              ///< 质量
    DWORD   dwPreset;               ///< 预设参数

    // 显示部分
    void*   lpGlobalContext;        ///< 全局环境，Android 中为 JavaVM*
    void*   lpSuface;               ///< 显示表面句柄

    DWORD   reserved[20];           ///< 保留
};
```

音视频数据包结构（struct MediaPacket）是 AnyChat 内核与 CODEC 交换数据的基本结构，里面包含了数据域（lpData）、数据长度（dwSize）和标志域（dwFlags），所有数据域的内存均由 AnyChat 内核分配与管理（包括输入类型与输出类型），CODEC 不需要再分配，对于输出类型的数据包，CODEC 内部需要检查 AnyChat 内核分配的内存是否足够（通过 dwSize 来判定），否则可能导致数据越界，形成非法内存访问异常，当 CODEC 内部检查到 AnyChat 内核分配的输出内存大小不足时，需要返回出错代码。标志域需要根据实际情况进行设置，如当视频编码 CODEC 输出的数据是关键帧时，标志域需要设置“MEDIACODEC_FLAGS_KEYFRAME”标志。

编解码器上下文句柄结构（struct MediaCodecContext）是 AnyChat 内核与 CODEC 之间传递参数的重要数据结构，初始化 CODEC 之前，AnyChat 内核会先设置相关的变量（如视频类型 CODEC 的宽度、高度、码率等），初始化 CODEC 之后，CODEC 内部需要设置该结构的相关变量，将信息返回给 AnyChat 内核，如初始化音频编码 CODEC 之后，CODEC 需要设置 dwFrameSize 变量，便于 AnyChat 内核了解每次需要向 CODEC 提交多少数据进行编码，详细信息可参考后面的“音频 CODEC”和“视频 CODEC”等相关章节的具体说明。

编解码器上下文句柄结构（struct MediaCodecContext）中的“lpPrivateData”非常重要，可用于保存编解码器内部的状态参数，通常在初始化时分配内存，并初始化相关的状态，然后在编码（解码）的过程中使用，最后在关闭时释放内存，分配内存、释放内存均在 CODEC 内部完成，“lpPrivateData”只是起一个中间桥梁的作用，AnyChat 内核会保障 CODEC 调用（初始化、编解码、关闭）之间的互斥。

2.3 基本信息接口

1）查询库基本信息接口

接口定义： DWORD **MediaCodec_GetLibraryInfo**(DWORD* dwCapability, DWORD* dwMainVer, DWORD* dwSubVer, CHAR* lpCompileTime, DWORD dwBufLen)

返 回 值： 0

参　　数：

DWORD* dwCapability	库所实现的 CODEC 类型标志组合
DWORD* dwMainVer	库的主版本号，用户自定义
DWORD* dwSubVer	库的从版本号，用户自定义
CHAR* lpCompileTime	库的编译时间，已由外部分配好内存空间
DWORD dwBufLen	保存编译时间缓冲区的大小

详细说明：

该 API 接口必须实现，AnyChat 内核加载外部编解码动态库时，首先会调用该接口，获取当前加载的动态库实现了那些 API 接口（根据 dwCapability 返回值来判断）。

2）查询编解码器信息接口

接口定义：DWORD **MediaCodec_GetCodecInfo**(DWORD dwCodecType, DWORD* dwCodecId, CHAR* lpCodecName, DWORD dwBufLen)

返 回 值：查询的 CODEC 类型存在时返回 0，否则返回-1

参　　数：

DWORD dwCodecType	需要查询的 CODEC 类型
DWORD* dwCodecId	返回该类型的 CODEC 编号值
CHAR* lpCodecName	返回该 CODEC 的名称，已由外部分配好内存
DWORD dwBufLen	保存 CODEC 名称缓冲区的大小

详细说明：

该 API 接口必须实现，AnyChat 内核首先通过“库基本信息接口”获取到库的 CODEC 类型标志组合，然后再调用该接口详细获取指定类型的 CODEC 编号与名称。

当用户实现的 CODEC 编号不在 AnyChat 内置 CODEC 列表（enum MEDIA_CODEC_ID）中时，可自定义一个 ID，需要注意的是，自定义的 CODEC ID 必须大于或等于 MEDIA_CODEC_ID_USERSTART 宏定义的值，宏定义以内的值属于保留 ID，用于 AnyChat 今后的扩展，如果上层应用使用了该宏定义以内的值，那么今后 AnyChat 平台的升级将可能导致 AnyChat 内核与上层之间的一些冲突。

2.4 注册应用

当按本编程指南所列要求生成目标动态库文件后，可以使用 AnyChat Platform Core SDK 的相关 API 将该动态库向内核注册，注册完成后，可根据 AnyChat 的 log 日志判断是否注册成功。

只有向 AnyChat 内核注册 CODEC 成功之后，才能在 AnyChat 平台使用新的编解码器，通常注册代码所处的位置是初始化 SDK（BRAC_InitSDK）调用之后。

注册示例代码如下所示：

```
// 向SDK注册CODEC
CHAR* lpCodecLibName = "SampleCodec.dll";     // or "libsamplecocec.so"
BRAC_SetSDKOption(BRAC_SO_CORESDK_LOADCODEC,lpCodecLibName,strlen(lpCodecLibName));
```

注册时可指定绝对路径（含动态库文件名），也可只写动态库文件名，AnyChat 平台将会在当前目录，或是设置的 Core SDK 路径下查找对应的文件并加载。

注册 CODEC 成功之后，会在本地日志文件中输出相关信息。

2.5 解码数据通过 GPU 渲染

当希望视频数据解码之后，直接传递给 GPU 进行渲染，实现高分辨率的高效率显示，即显示部分在 Codec 中完成，则可以通过如下方式实现：

1、 在库能力标志上加上：MEDIACODEC_CAPABILITY_DIRECTRENDER，告知上层 Codec 具备 GPU 渲染的能力；

2、 在应用层开启 GPU 渲染模式，即调用 API：BRAC_SetSDKOption，设置 BRAC_SO_VIDEOSHOW_GPUDIRECTRENDER 为 1；

3、 AnyChat 内核在初始化解码器“MediaCodec_VideoCodec_InitEncoder”时，会初始化上下文句柄“MediaCodecContext”中的“lpGlobalContext”、“lpSuface”，告知解码器显示窗口，便于 GPU 渲染；若上层没有初始化前述两个变量，则表示上层只要求 Codec 进行解码，不需要显示；

4、 视频解码器解码到一帧数据之后，将数据直接传递给 GPU 渲染，同时返回-1，向上层反馈解码失败的信息，告知上层没有解码输出数据。

三、音频 CODEC

音频 CODEC 包含“音频编码器”和“音频解码器”两种类型，可分别实现，音频编码器实现将音频采样数据（PCM）编码为音频流，音频解码器实现将编码后的数据包解码为 PCM 采样数据。

音频 CODEC 需要使用帧长（dwFrameSize）变量，该变量所表示的意思定义为：编解码器一次能处理的单个通道音频采样帧数。根据 dwFrameSize 长度的音频帧所占用内存大小计算公式为：

dwByteSize = dwFrameSize × dwChannels × (dwBitsPerSample>>3)

实现音频编码 CODEC 时，需要在“MediaCodec_GetLibraryInfo”的 dwCapability 标志中加上 MEDIACODEC_CAPABILITY_AUDIOENCODEC 宏定义。

实现音频解码 CODEC 时，需要在“MediaCodec_GetLibraryInfo”的 dwCapability 标志中加上 MEDIACODEC_CAPABILITY_AUDIODECODEC 宏定义。

3.1 初始化音频编码器

接口定义： DWORD MediaCodec_AudioCodec_InitEncoder(MediaCodecContext* lpContext);

返 回 值： 初始化成功返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext　　CODEC 上下文句柄

详细说明：

AnyChat 内核会初始化上下文句柄中的“dwCodecId”、“dwChannels”、“dwSamplesPerSec”、“dwBitsPerSample”以及“dwBitrate”等参数，初始化完成之后，CODEC 内部需要初始化“dwFrameSize”变量，表示编码器每次编码时需要多少帧音频采样数据。

3.2 音频数据编码

接口定义： DWORD MediaCodec_AudioCodec_Encode(MediaCodecContext* lpContext, MediaPacket* InPacket, MediaPacket* OutPacket);

返 回 值：编码成功返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext	CODEC 上下文句柄
MediaPacket* InPacket	输入数据包，指向音频采样 PCM 数据
MediaPacket* OutPacket	输出数据包，返回编码后的音频流

详细说明：

AnyChat 内核会严格按初始化编码器时，编码器返回的“dwFrameSize”所表示的帧长输入数据，编码完成后，CODEC 内部将编码后的数据保存在“OutPacket”所指向的内存空间，并将“OutPacket”中的“dwSize”修改为实际输出的数据长度。

3.3 关闭音频编码器

接口定义：`DWORD MediaCodec_AudioCodec_CloseEncoder(MediaCodecContext* lpContext);`

返 回 值：成功关闭返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext	CODEC 上下文句柄

详细说明：

关闭音频编码，清理初始化编码器时分配的内存等。上下文句柄自身的内存由 AnyChat 内核分配和管理，CODEC 内部不需要释放。

3.4 初始化音频解码器

接口定义：`DWORD MediaCodec_AudioCodec_InitDecoder(MediaCodecContext* lpContext);`

返 回 值：初始化成功返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext	CODEC 上下文句柄

详细说明：

AnyChat 内核会初始化上下文句柄中的“dwCodecId”、“dwChannels”、“dwSamplesPerSec”以及“dwBitsPerSample”等参数，初始化完成之后，CODEC 内部需要初始化“dwFrameSize”变量，表示解码器每次解码后将输出多少帧音

频采样数据。

3.5 音频数据解码

接口定义： DWORD MediaCodec_AudioCodec_Decode(MediaCodecContext* lpContext, MediaPacket* InPacket, MediaPacket* OutPacket);

返 回 值：解码成功返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext	CODEC 上下文句柄
MediaPacket* InPacket	输入数据包，指向已编码的音频流
MediaPacket* OutPacket	输出数据包，返回音频采样 PCM 数据

详细说明：

AnyChat 内核会传入完整的一帧数据（与编码端输出对应），解码完成后，CODEC 内部将解码后的数据保存在“OutPacket”所指向的内存空间，并将“OutPacket”中的“dwSize”修改为实际输出的数据长度。

3.6 关闭音频解码器

接口定义： DWORD MediaCodec_AudioCodec_CloseDecoder(MediaCodecContext* lpContext);

返 回 值：成功关闭返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext	CODEC 上下文句柄

详细说明：

关闭音频解码，清理初始化解码器时分配的内存等。上下文句柄自身的内存由 AnyChat 内核分配和管理，CODEC 内部不需要释放。

四、视频 CODEC

视频 CODEC 包含“视频编码器”和“视频解码器”两种类型，可分别实现，视频编码器实现将视频帧数据（YUV420P）编码为视频流，视频解码器实现将编码后的数据包解码为视频帧数据（YUV420P）。

AnyChat 内核与视频 CODEC 进行交互时，所传递的视频帧格式默认均为 YUV420P 格式（Planar YUV 4:2:0, 12bpp, (1 Cr & Cb sample per 2x2 Y samples)），该格式是目前 ITU 相关标准推荐的视频编码输入格式。

实现视频编码 CODEC 时，需要在“MediaCodec_GetLibraryInfo”的 dwCapability 标志中加上 MEDIACODEC_CAPABILITY_VIDEOENCODEC 宏定义。

实现视频解码 CODEC 时，需要在“MediaCodec_GetLibraryInfo”的 dwCapability 标志中加上 MEDIACODEC_CAPABILITY_VIDEODECODEC 宏定义。

4.1 初始化视频编码器

接口定义： DWORD MediaCodec_VideoCodec_InitEncoder(MediaCodecContext* lpContext);

返 回 值： 初始化成功返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext　　CODEC 上下文句柄

详细说明：

AnyChat 内核会初始化上下文句柄中的“dwCodecId”、“dwWidth”、“dwHeight”、“dwFrameRate”、“dwGopSize”、“PixFmt”、“dwQuality”、“dwPreset”以及“dwBitrate”等参数。

大部分的 H.264 硬件编码器只在第一帧才输出 SPS、PPS 等初始化解码器所需要的重要参数信息，而 AnyChat 需要定期传输 SPS、PPS 等信息（随关键帧一起传输），则可按如下方式设置标志位：

lpContext->dwFlags |= MEDIACODEC_FLAGS_WANTSPSPPS;

设置 MEDIACODEC_FLAGS_WANTSPSPPS 标志后，AnyChat 内核将会保存第一帧的 SPS、PPS 等信息，当后续编码器输出关键帧时，则会自动将 SPS、PPS 等信

息放在关键帧之前再传输。

视频的码率单位为：bps，当设置为 0 时，表示采用质量因子模式，可参考“dwQuality”参数值来初始化编码器。

4.2 视频数据编码

接口定义： DWORD MediaCodec_VideoCodec_Encode(MediaCodecContext* lpContext, MediaPacket* InPacket, MediaPacket* OutPacket);

返 回 值：编码成功返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext	CODEC 上下文句柄
MediaPacket* InPacket	输入数据包，指向视频帧 YUV 数据
MediaPacket* OutPacket	输出数据包，返回编码后的视频流

详细说明：

AnyChat 内核每次输入一帧完整的图像数据，编码完成后，CODEC 内部将编码后的数据保存在“OutPacket”所指向的内存空间，并将“OutPacket”中的“dwSize”修改为实际输出的数据长度。

当视频编码 CODEC 输出的数据是关键帧时，标志域需要设置“MEDIACODEC_FLAGS_KEYFRAME”标志（当初始化编码器时设置 MEDIACODEC_FLAGS_WANTSPSPPS 标志后，AnyChat 内核会自动探测帧类型并设置关键帧标志）。

AnyChat 内核要求编码器能定期输出关键帧（如 H.264 的 IDR 帧，同时包含 PPS、SPS 等 NAL 信息），输出间隔可参考 CODEC 上下文句柄中的 dwGopSize 参数值，如果 CODEC 不能定期输出关键帧，则可能导致对方接收到视频流之后无法正常解码。

4.3 关闭视频编码器

接口定义： DWORD MediaCodec_VideoCodec_CloseEncoder(MediaCodecContext* lpContext);

返 回 值：成功关闭返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext　　CODEC 上下文句柄

详细说明：

关闭视频编码，清理初始化编码器时分配的内存等。上下文句柄自身的内存由 AnyChat 内核分配和管理，CODEC 内部不需要释放。

4.4 初始化视频解码器

接口定义：DWORD MediaCodec_VideoCodec_InitDecoder(MediaCodecContext* lpContext);

返 回 值：初始化成功返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext　　CODEC 上下文句柄

详细说明：

AnyChat 内核会初始化上下文句柄中的“dwCodecId”、“dwWidth”、“dwHeight”以及“PixFmt”等参数。

4.5 视频数据解码

接口定义：DWORD MediaCodec_VideoCodec_Decode(MediaCodecContext* lpContext, MediaPacket* InPacket, MediaPacket* OutPacket);

返 回 值：解码成功返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext　　CODEC 上下文句柄

MediaPacket* InPacket　　输入数据包，指向已编码的视频流

MediaPacket* OutPacket　　输出数据包，返回解码后的 YUV 数据

详细说明：

AnyChat 内核会传入完整的一帧数据（与编码端输出对应），解码完成后，CODEC 内部将解码后的数据保存在“OutPacket”所指向的内存空间，并将

“OutPacket”中的“dwSize”修改为实际输出的数据长度。

4.6 关闭视频解码器

接口定义： DWORD MediaCodec_VideoCodec_CloseDecoder(MediaCodecContext* lpContext);

返 回 值： 成功关闭返回 0，否则返回出错代码，或-1

参　　数：

MediaCodecContext* lpContext　　CODEC 上下文句柄

详细说明：

关闭视频解码，清理初始化解码器时分配的内存等。上下文句柄自身的内存由 AnyChat 内核分配和管理，CODEC 内部不需要释放。

五、应用举例

下面应用一个简单的例子，演示一个 AnyChat Codec 库的框架代码，实现音频的 CELT 编码、解码模块（不包含音频编解码算法部分）。

如需要完整的示例工程源代码，请与佰锐科技技术支持中心联系，我们将安排专业工程师提供技术协助。

```
// 获取库基本信息
DWORD MediaCodec_GetLibraryInfo(DWORD* dwCapability, DWORD* dwMainVer, DWORD* dwSubVer, CHAR* lpCompileTime, DWORD dwBufLen)
{
    // 版本信息
    *dwMainVer = 1;
    *dwSubVer = 0;
    // 编译时间
    if(dwBufLen && lpCompileTime)
        _snprintf(lpCompileTime,dwBufLen,"%s %s",__DATE__,__TIME__);
    // 库的编解码能力（非常重要，库实现某一种编解码器时，才能加上对应的标志）
    *dwCapability = MEDIACODEC_CAPABILITY_AUDIOENCODEC | MEDIACODEC_CAPABILITY_AUDIODECODEC;
    return 0;
}
// 获取编解码器基本信息
DWORD MediaCodec_GetCodecInfo(DWORD dwCodecType, DWORD* dwCodecId, CHAR* lpCodecName, DWORD dwBufLen)
{
    DWORD ret = -1;
    if(dwCodecType & MEDIACODEC_CAPABILITY_AUDIOENCODEC)
    {
        *dwCodecId = MEDIA_CODEC_ID_CELT;
        if(lpCodecName && dwBufLen)
            _snprintf(lpCodecName, dwBufLen, "%s", "CELT Audio Encoder");
        ret = 0;
    }
    else if(dwCodecType & MEDIACODEC_CAPABILITY_AUDIODECODEC)
    {
        *dwCodecId = MEDIA_CODEC_ID_CELT;
        if(lpCodecName && dwBufLen)
            _snprintf(lpCodecName, dwBufLen, "%s", "CELT Audio Decoder");
        ret = 0
    }
```

```
    return ret;
}

// 打开音频编码器
DWORD MediaCodec_AudioCodec_InitEncoder(MediaCodecContext* lpContext)
{
    return CCeltCodec::InitEncoder(lpContext);
}
// 对音频进行编码
DWORD MediaCodec_AudioCodec_Encode(MediaCodecContext* lpContext, MediaPacket* InPacket,
MediaPacket* OutPacket)
{
    return CCeltCodec::Encode(lpContext, InPacket, OutPacket);
}
// 关闭音频解码器
DWORD MediaCodec_AudioCodec_CloseEncoder(MediaCodecContext* lpContext)
{
    return CCeltCodec::CloseEncoder(lpContext);
}

// 打开音频解码器
DWORD MediaCodec_AudioCodec_InitDecoder(MediaCodecContext* lpContext)
{
    return CCeltCodec::InitDecoder(lpContext);
}
// 对音频流进行解码
DWORD MediaCodec_AudioCodec_Decode(MediaCodecContext* lpContext, MediaPacket* InPacket,
MediaPacket* OutPacket)
{
    return CCeltCodec::Decode(lpContext, InPacket, OutPacket);
}
// 关闭音频解码器
DWORD MediaCodec_AudioCodec_CloseDecoder(MediaCodecContext* lpContext)
{
    return CCeltCodec::CloseDecoder(lpContext);
}
```

六、技术支持

在您使用 AnyChat SDK 的过程中，遇到任何困难，请与我们联系，我们将热忱为您提供帮助。

您可以通过如下方式与我们取得联系：

1、 在线论坛：http://bbs.anychat.cn/

2、 知识中心：http://www.anychat.cn/faq/

3、 官方网站：http://www.anychat.cn

4、 电子邮件：service@bairuitech.com

5、 24 小时客服电话：+86 （020） 85276986、38109065、38103410