点对点 SDK(C++版)说明(V1.0.0)

www.mediapro.cc

一、基本概念

音视频纯发送端：可以将自身音视频发送给对端，但不具备接收对端音视频流的能力。
音视频纯接收端：可以接收对端发送过来音视频，但不具备发送音视频给对端的能力。
音视频收发一体端：同时具备音视频发送和接收的能力。
在不同的业务场景里，通过设置不同的客户端类型来降低 SDK 资源的占用。比如投屏类应用，一端设置为纯发送端，另一端设置为纯接收端，这样发送端无需创建接收线程等资源，接收端也无需创建发送所需的资源。当需要全双工通讯时，设置双方为收发一体类型。
传输参数：本文传输参数是指视频通道的 FEC 上行 FEC 冗余度、上行 FEC Group 组大小、接收 Qos 丢包等待时间，音频通道在内部已经根据经验数据配置好了合适的值且不对外开放。对于音视频纯接收者，同样也可以设置 FEC 上行冗余度、上行 FEC Group 组大小，只是没有实际意义(因为不会发送数据，当然也不会进行 FEC 编码)。
主动 API 接口：由外层应用主动发起的调用，比如登录、下线、发送音视频等。
被动 API 接口：称为回调接口更为贴切，比如接收到来对端的音视频数据、底层帧率自适应建议的通知、底层 IDR 帧请求的通知。

二、日志接口

日志对于问题定位非常重要，SDK 提供了一组全局日志开关接口，用户只需要在应用中完成如下步骤即可激活 SDK 日志。

1、包含日志头文件

```
#include "SDLog.h"
```

2、系统启动时调用一次初始化

```
SDLOG_INIT(const char * outputPath,  int outputLevel)
```

参数：

@param: outputPath：日志文件输出的目录，若目录不存在，SDK 将自动创建，支持相对路径或绝对路径。

@param: outputLevel：日志输出的级别，只有等于或者高于该级别的日志会输出到文件，日志级别定义见 SDLog.h：SD_LOG_LEVEL_DEBUG、SD_LOG_LEVEL_INFO、SD_LOG_LEVEL_WARNING、SD_LOG_LEVEL_ERROR、SD_LOG_LEVEL_ALARM、SD_LOG_LEVEL_FATAL、SD_LOG_LEVEL_NONE，当指定为 SD_LOG_LEVEL_NONE 时，将不会生成日志文件。

3、系统退出时调用一次反初始化

```
SDLOG_CLOSE()
```

三、主动接口

以下接口均为线程安全，可以在多线程中调用，用户需要在其应用中包含头文件 **SDTerminal.h**

1、创建本地资源

```
int IFace_OnlineUser(BYTE byUserType, const char *strLocalIP, USHORT shLocalPort, const char *strRemoteIP, USHORT shRemotePort);
```

参数：

@byUserType，表示客户端的类型，定义位于"CmdProtocolStruct.h"，包括

```
#define      USER_TYPE_OTHER                  0   //保留未使用的类型
#define      USER_TYPE_AV_SEND_RECV           1   //同时进行音视频发送和接收
#define      USER_TYPE_AV_RECV_ONLY           2   //仅接收音视频
#define      USER_TYPE_AV_SEND_ONLY           3   //仅发送音视频
```

用户根据自身业务选择合适的客户端类型，以获得资源的最低占用。

@strLocalIP，绑定的本地 IP 地址，当设置为空字符串”“时（不是 NULL，而是空字符串），内部将使用 INADDR_ANY，交由操作系统选择一个网卡 IP。当存在多个网卡时，建议指定使用的哪一个网卡 IP，避免数据无法正确发出。

@shLocalPort，绑定的本地端口号，当客户端为 USER_TYPE_AV_RECV_ONLY 或者 USER_TYPE_AV_SEND_RECV 时，必须设置非 0 的本地端口，以便接收对方发出的流。当客户端为 USER_TYPE_AV_SEND_ONLY 时，允许设置本地端口为 0，由操作系统选择一个当前可用的端口发出数据。

@strRemoteIP，远端 IP 地址。当客户端为纯接收端 USER_TYPE_AV_RECV_ONLY 时，设置远端 IP 地址为空字符串即可（作为纯接收端，一般是不知道发送端的 IP 和端口的，内部将在收到远端数据后自动翻转 IP 和端口，从而获得可用于向远端发送数据的 IP 和端口）。

@shRemotePort，远端端口号。当客户端为纯接收端 USER_TYPE_AV_RECV_ONLY 时，设置远端端口号为 0 即可。

返回值：

返回 0 表示创建成功，返回负数则为失败，负数值为其错误码。

2、回收创建的资源

```
void IFace_OfflineUser();
```

参数：无

返回值：无

3、发送视频数据

```
void IFace_SendVideoStreamData(BYTE* buf, UINT unLen);
```

发送已编码的一帧视频码流，内部自带拆分功能，一次传入**带 H264 起始码**的一帧码流。SDK 内部管理时间戳。

参数：

@buf，码流存放区。

@unLen，码流长度。

返回值：无

4、发送音频数据

```
void IFace_SendAudioStreamData(BYTE* buf, UINT unLen);
```

发送已编码的一帧音频码流，一次传一帧 **ADTS** 码流。SDK 内部管理时间戳。

参数：

@buf，码流存放区。

@unLen，码流长度。

返回值：无

5、设置音视频传输参数

```
void IFace_SetTransParams(UINT unJitterBuffDelay, FEC_REDUN_METHOD_TYPE eRedunMethod,
UINT unRedunRatio, UINT unFecGroupSize, BOOL bEnableNack);
```

参数：

@unJitterBuffDelay，本客户端接收码流时的内部缓存时间（毫秒）,范围 0~600。设置为 0 时，将关闭内部接收 JitterBuff 功能。本参数仅影响接收，对于 USER_TYPE_AV_SEND_ONLY 不会生效。

@eRedunMethod，为上行 FEC 冗余度方法，包括 AUTO_REDUN 自动冗余度、FIX_REDUN 固定冗余度。

@unRedunRatio，固定冗余度时对应的上行冗余比率，比如设置为 30，则表示使用 30%冗余。最低允许 10，最高 100。

@unFecGroupSize，为上行 FEC 分组大小，512Kbps 以下建议 8，512Kbps~1Mbps 建议设置为 16，1Mbps~2Mbps 建议设置 24，2Mbps~4Mbp 建议设置 28，4Mbps 以上建议 36。

@bEnableNack，是否启用 NACK 功能，关于 NACK 请阅读相关文档，建议收发双方均开启，单方开启将不会生效。

返回值：无

注意：本函数需在 IFace_OnlineUser 之前调用，本 API 的使用若有疑问，请联系技术支持获得帮助。

6、获取当前丢包率数据

```
void IFace_GetVideoAudioUpDownLostRatio(float &fVideoUpLostRatio, float
&fVideoDownLostRatio, float &fAudioUpLostRatio, float &fAudioDownLostRatio)
```

参数：

@fVideoUpLostRatio，获取视频上行丢包率，范围 0~100，表示百分比。

@fVideoDownLostRatio，获取视频下行丢包率

@fAudioUpLostRatio，获取音频上行丢包率

@fAudioDownLostRatio，获取音频下行丢包率

7、获取当前码率数据

```
void IFace_GetVideoAudioUpDownBitrate(float &fVideoUpRate, float &fVideoDownRate, float
&fAudioUpRate, float &fAudioDownRate);
```

参数：

@fVideoUpRate，视频上行码率，单位 kbps

@fVideoDownRate,视频下行码率

@fAudioUpRate，音频上行码率

@fAudioDownRate，音频下行码率

8、获取当前视频通道的实时 RTT

```
UINT IFace_GetNetWorkDelayOnVideoChannel();
```

返回值：获得当前视频通道的实时 RTT 值，单位 ms

9、获取当前 SDK 版本信息

```
UINT IFace_GetVersion();
```

返回值：获得当前 SDK 的版本信息

四、被动接口

被动接口均使用 CSDTermCmdIFace 类虚函数方式提供，用户需要使用一个类继承 CSDTermCmdIFace 类并实现该虚函数。用户在其应用中包含头文件 SDTermCmdIFace.h

1、收到远端发送的视频数据

```
virtual void OnRemoteVideo(BYTE byIndex, BYTE* data, UINT unLen, UINT unPTS, VideoFrameInfo tFrameInfo);
```

参数：

@byIndex，保留，暂未使用。

@data，指向接收的码流帧存放区域

@unLen，接收码流帧的长度

@unPTS，当前码流帧的时间戳，时基 1KHZ。

@tFrameInfo，当前码流的详细描述，其定义如下：

```
typedef struct VideoFrameInfo
{
    UINT unWidth;
    UINT unHeight;
    BOOL bPacketLost;
    BOOL bKeyFrame;
}VideoFrameInfo;
```

模块内部会解析 SPS 获得当前码流的宽和高并告知外层（可能外层根本不需要这个）。另外两个 BOOL 变量是用于外层实现丢帧冻结机制。bPacketLost 表示当前帧是否接收完整，若网络丢包且 FEC 未能恢复时，该标志将置位。bKeyFrame 表示当前帧是否为 IDR 关键帧。如何使用这两个标志实现丢帧冻结可以联系技术支持获得帮助。需要说明的是，当没有丢包发生时，本函数的输出与对方调用 IFace_SendVideoStreamData 函数的输入完全一致。

返回值：无

说明：SDK 内部是在独立于网络接收线程之外的线程中调用本接口，所以外层可以将一定耗时的操作（比如解码）放置在此。

2、收到服务器下发的音频数据

```
virtual void OnRemoteAudio(BYTE byIndex, BYTE* data, UINT unLen, UINT unPTS, AudioFrameInfo tFrameInfo);
```

参数：

@byIndex，保留，暂未使用。

@data，指向接收的码流帧存放区域

@unLen，接收码流帧的长度

@unPTS，当前码流帧的时间戳

@tFrameInfo，当前码流的详细描述，其定义如下：

```
typedef struct AudioFrameInfo
{
    UINT unCodecType;
    UINT unSampleRate;
```

```
    UINT unChannelNum;
}AudioFrameInfo;
```

音频帧为 ADTS 格式，其每个包头部均附带了采样率、通道数、编码格式等信息。SDK 暂时未做解析，tFrameInfo 内容暂不可参考。

返回值：无

说明：SDK 内部是在独立于网络接收线程之外的线程中调用本接口，所以外层可以将一定耗时的操作（比如解码）放置在此。

3、帧率自适应建议输出

```
virtual BOOL OnAutoBitrateProcess(UINT unFrameDropInterval) { return FALSE; }
```

参数：

@unFrameDropInterval，接口告知外层，当前推荐使用的均匀丢帧间隔，以间接实现码率自适应。比如 unFrameDropInterval 为 3 时，表示告知外层需要每 3 帧内丢弃 1 帧，此时若外层帧率为 30fps，则实际送编码帧率为 20fps。unFrameDropInterval 的取值范围为{10, 7, 6, 5, 4, 3, 2，0}，当取值为 0 时，表示不做丢帧处理。

返回值：当外层实现了帧率自适应功能时返回 TRUE，否则返回 FALSE。

说明：SDK 内部将根据上行丢包率、NACK 等情况得到一个推荐的丢帧系数，并通过本回调函数通知外层实现具体的丢帧动作。通过外层丢帧的方式，我们可以间接实现码率自适应。帧率自适应相比通过编码器实现调控码率而言，其通用性更好（软编码、硬编码均可适应），且不易察觉到码率调整带来的画质突变。当用户不希望实现帧率自适应时，忽略本函数即可。

4、收到远端编码 IDR 帧请求

```
virtual void OnRemoteIdrRequest()
```

参数：无

返回值：无

说明：当远端接收失败时，将会通知本端，请求编码 IDR 帧，以便尽快回复因丢包导致的花屏或卡顿。注意本接口为同步调用方式，因此外层不应在其中执行耗时操作，应尽快返回。当用户不希望实现 IDR 请求机制时，忽略本函数即可。