RDLive For Windows (v1.1)

1. 概述

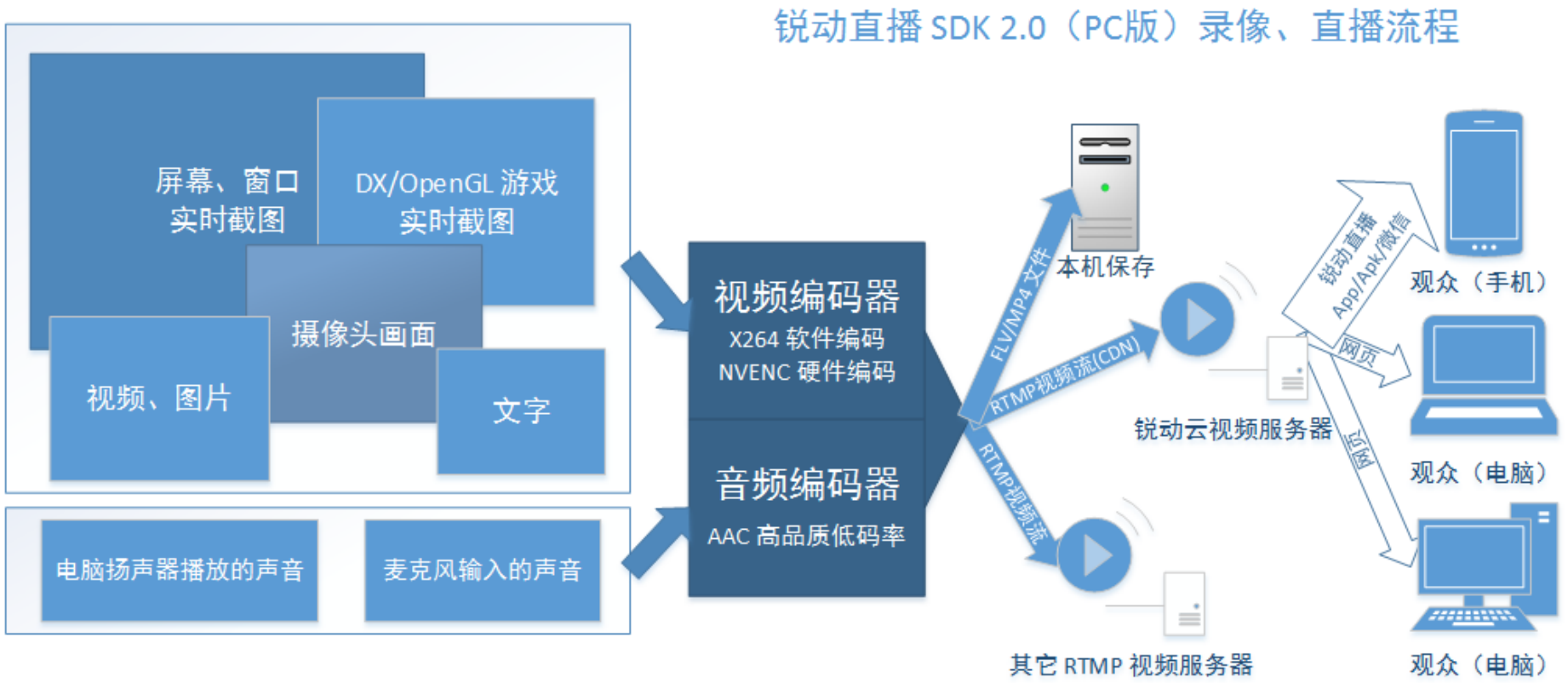
锐动录制/直播 SDK Window 版 （ v1.1 ）

本 SDK 是锐动直播SDK的PC版本，支持从 Windows XP 开始的各个 Windows 系统（不支持 CE 、Mobile 和 RT）。
SDK 的基本功能：把屏幕画面、摄像头、视频、游戏画面、音乐、文字等各种资源整合在一起，实时编码为视频文件和上传到视频服务器进行直播。
适合的应用场景：教学视频录制、在线教学、游戏录制/直播、美女直播、会议直播等各种需要高质量低码率音视频录制和直播的场合。

技术特点：使用 OpenGL 进行多个画面的整合，支持多个场景切换，使用 x264 或 NVENC 进行视频编码，可以在低码率下得到高质量的视频。

优势：本 SDK 是一系列 SDK 中的一部分，使用这些 SDK 可以迅速搭建起从主播到观众的整个应用流程。如果使用本公司提供的云视频服务器，更可以免去视频服务器架设、视频流分发等各种麻烦。即使不使用直播相关功能，本 SDK 也支持录像保存在本机，特别适用于屏幕录制和游戏录制。

录像和直播的基本流程框图：



画面编辑：

- 每一个画面来源在本 SDK 中被称为元件（Chip），类似于 PhotoShop 中的图层。
- 内置了完整的预览界面，并且支持使用鼠标进行选中、拖动、改变大小等各种基本操作。
- 支持多个场景，每个场景有自己的元件组合，可以快速切换场景。

画面来源：

- 桌面画面捕获
  - 使用 GDI 方式截取桌面窗口画面，支持多显示器，支持各种屏幕分辨率和色深(2bit 至 32bit)。
  - 支持在桌面画面捕获时，暂时禁用 Win7 Aero，以提高性能。
- 摄像头画面捕获
  - 支持各种 USB 摄像头、和 WDM 标准的视频采集卡。
- 游戏画面捕获 (开发中，2017 年 4 月份会更新)
  - 直接从游戏进程中获取 DirectX 以及 OpenGL 的渲染结果画面，不使用 GDI 截图的方式，能支持绝大多数游戏，包括常规的 GDI 截图无法截取到的画面。
  - 对游戏性能影响较低，比 GDI 截图消耗更小。
  - 兼容 DirectX 7/8/9/10/11 及 OpenGL 各个版本，各种游戏分辨率和画面颜色格式。
  - 支持以各种帧率捕获游戏画面。
- 视频和图片文件
  - 支持 FLV、MP4 等使用 H.264 编码的视频文件。(开发中，2017 年 4 月份会更新)
  - 支持 PNG、JPG、BMP 等常见格式的音频文件。
  - 支持 Flash、GIF 动画文件。(Flash 开发中，2017 年 4 月份会更新)
- 文字(开发中，2017 年 4 月或 5 月会更新)
  - 自定义文本内容、字体、大小、显示位置。

音频捕获：

- 支持 WinXP/Win7/8/10 下的声音捕获，且把录音设备抽象为“输出的声音（电脑播放的声音）”和“输入的声音（麦克风等）”这两类，方便二次开发。
- 支持软件混音和重采样，自定义录制声音的的声道数和采样格式。
- 通过软件修正可保证长时间录音时，录音数据的时长与实际时长基本无偏差。

音视频编码：

- x264 视频编码（视频画面质量高，码率控制好，但会消耗大量 CPU 和内存资源，根据编码质量的设置不同，消耗不同。）
- NVENC 硬件视频编码（视频画面质量高，码率控制较好，几乎不消耗 CPU 和内存资源，但需要 Nvidia 显卡的支持）

- AAC 音频编码（码率低，音质损失小）

文件保存和直播：

- 保存视频为 FLV、MP4、AVI 格式文件。
- 指定 RTMP 地址直播。
- 使用锐动云服务器直播。锐动云服务器可以自动地把直播的视频分发到全国各地的服务器，在国内任何地域，都没有访问速度的困绕。

2. 怎么开始？

这个栏目下有几段对 SDK 初步使用的简单介绍：

开发环境

初始化

设置画面内容

控制预览画面

开始录制和直播

处理回调通知

2.1 开发环境

SDK 自身的开发环境：

开发工具：Visual C++ 2008 + SP1 + DirectX SDK (June 2010) + DX8 SDK
操作系统：Windows 10，测试：Windows XP/7/8/10

SDK 用户的开发环境：

开发工具：任何一种可以使用 DLL 和标准 API 的编程语言
操作系统：Windows XP/7/8/10
硬件需求：显卡需要安装驱动程序，以提供 OpenGL 支持。其它不限

常见开发工具的开发过程举例：

Visual 2008(MFC)：

注：SDK 中提供了 Visual 2008(MFC) 编写的完整功能的演示程序及其源代码可以参考，或直接以演示程序为基础进行修改开发新的应用。
请打开 Demo\DemoFull_VC2008(Qt4)\DemoFull_VC2008(MFC).vcproj

1. 安装 Visual 2008，安装 SP1 补丁。
2. 新建 “MFC 应用程序”项目（假设项目名称为 Live ）。
3. 复制 SDK 中的 Include、Lib 文件夹到项目文件夹中 Live。
4. 打开项目中的 stdafx.h，在里面加入：
   ```
   #include "..\Include\RDLiveSDK.h"
   #pragma comment ( lib, "..\\Lib\\RDLiveSDK.lib " )
   ```
5. 在 CLiveApp::InitInstance() 中 或者 CLiveDlg::OnInitDialog() 中，调用 SDK 的初始化函数 RDLive_Init()。
6. 设置好项目的输出文件夹后直接编译，以检查 .h 和 .lib 文件的路径是否正确，如果不正确请检查并修改。
7. 复制 SDK 的 Debug 中的文件到项目的 Debug 输出文件夹，和复制 SDK 的 Release 中的文件到项目的 Release 的输出文件夹。
8. 继续编写其它代码……

Qt 4.8.x 或 Qt5.x：

注：SDK 中提供了 Qt 4.8.x 编写的完整功能的演示程序及其源代码可以参考，或直接以演示程序为基础进行修改开发新的应用。
推荐使用 C/C++ 开发的程序员使用 VC + Qt 做为开发环境，因为 VC 的 IDE 强大方便，Qt 搭建华丽的界面简单直接。
VC 请打开 Demo\DemoFull_VC2008(Qt4)\DemoFull_VC2008(Qt4).vcproj
Qt Creator 请打开 Demo\DemoFull_VC2008(Qt4)\DemoFull_QtCreator(Qt4).pro ()

首先要安装 Qt，下载 Qt 的地址：https://download.qt.io/official_releases/qt/　请选择一个合适的版本下载并安装。

使用 Qt Creator 作为 IDE：

Qt Creator 可以使用 MinGW 编译器，也可以使用 VC 的编译器，或者其它的编译器。
如果电脑上安装了 VC，那么可以直接使用 VC 做为 IDE，或者使用 Qt Creator 把编译器设置为 VC 的编译器。
要是打算完全脱离 VC，那么可以下载并安装 MinGW 的编译器。
如果使用 MinGW 编译器，请下载文件名中带 "mingw" 的版本的 Qt 安装包，例如 qt-opensource-windows-x86-mingw482-4.8.7.exe 或者 qt-opensource-windows-x86-mingw530-5.8.0.exe 等。
如果 VC 编译器，请看下面的**使用 Visual Studio 作为 IDE**，对不同的版本 VC 版本下载对应的 Qt。
下载并安装 Qt Creator：
如果使用 4.8.x 版的 Qt，建议使用 3.4.x 及之前的 Qt Creator 版本，因为可以不用在 Qt.io 网站上去注册用户，例如 qt-creator-opensource-windows-x86-3.4.2.exe。
如果使用 5.x 和更新的 Qt 版本，请下载较新的 Qt Creator 版本，例如 qt-creator-opensource-windows-x86-4.2.0.exe 。

1. 运行 Qt Creator，打开“工具”菜单“选项”，点击左侧构建和运行，检查右边 Qt 版本、编译器、构建套件等配置是否正确。
2. 点击“New Project"，创建应用程序“Qt Widgets Application，点击 Choose 设置项目名称（假设项目名称为 Live ）、路径等，直到完成创建。
3. 复制 SDK 中的 Include、Lib 文件夹到项目文件夹中 Live。
4. 打开 Live.pro，添加下面的内容：
```
LIBS += ../Lib/RDLiveSDK.lib
PRECOMPILED_HEADER    = stdafx.h
```
5. 新建一个 stdafx.h 文件作为预编译头文件，并添加以下内容：
```
#ifndef WINVER
#define WINVER 0x0500
#endif
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0500
#endif
```

```
#include <Windows.h>
#include "..\Include\RDLiveSDK.h"
```

6. 在 Live::Live() 构造函数中，调用 SDK 的初始化函数 RDLive_Init()。
7. 设置好项目的输出文件夹后直接编译，以检查 .h 和 .lib 文件的路径是否正确，如果不正确请检查并修改。
8. 复制 SDK 的 Debug 中的文件到项目的 Debug 输出文件夹，和复制 SDK 的 Release 中的文件到项目的 Release 的输出文件夹。
9. 继续编写其它代码……

使用 Visual Studio 作为 IDE：

使用 VS2008 作为开发用的 IDE，请下载文件名中带 "vs2008" 的版本（需 Qt 4.8.x），例如 qt-opensource-windows-x86-vs2008-4.8.7.exe。
使用 VS2010 作为开发用的 IDE，请下载文件名中带 "vs2010" 的版本（需 Qt 4.8.x），例如 qt-opensource-windows-x86-vs2010-4.8.7.exe。
使用 VS2013 作为开发用的 IDE，请下载文件名中带 "msvc2013" 的版本（需 Qt 5.x），例如 qt-opensource-windows-x86-msvc2013-5.8.0.exe。
使用 VS2015 作为开发用的 IDE，请下载文件名中带 "msvc2015" 的版本（需 Qt 5.x），例如 qt-opensource-windows-x86-msvc2015-5.8.0.exe。

下载并安装 Qt for VS 的插件（vs-addin ）：

VS 可以不安装 vs-addin 也能使用 Qt，但强烈建议安装 vs-addin，否则就需要完全手工设置 Qt 的界面文件、资源文件的自定义编译方式。
vs-addin 下载地址：https://download.qt.io/official_releases/vsaddin/ ，
其中 Qt 4.8.x 和 VS 2008/2010 请使用 1.1.11 版本，Qt 5.x 和 VS 2013/2015 请使用 1.2.0 以上的版本。

1. 运行 VC，在菜单栏上找到 “Qt” 菜单“Qt Option”，检查 Qt 的版本设置等是否正确。
2. 新建项目，“Qt4 Projects” 的子项“Qt Application” （假设项目名称为 Live ）。
   在弹出的工程向导的最后一页，请勾选上“Use Precompiled Headers”，也就是使用预编译头文件，这就和普通的 VC 项目一样了。
3. 复制 SDK 中的 Include、Lib 文件夹到项目文件夹中 Live。
4. 打开项目中的 stdafx.h，在里面加入：
   ```
   #include "..\Include\RDLiveSDK.h"
   #pragma comment ( lib, "..\\Lib\\RDLiveSDK.lib " )
   ```
5. 在 Live::Live() 构造函数中，调用 SDK 的初始化函数 RDLive_Init()。
6. 设置好项目的输出文件夹后直接编译，以检查 .h 和 .lib 文件的路径是否正确，如果不正确请检查并修改。

7. 复制 SDK 的 Debug 中的文件到项目的 Debug 输出文件夹，和复制 SDK 的 Release 中的文件到项目的 Release 的输出文件夹。
8. 继续编写其它代码……

2.2 初始化

SDK 需要调用初始化函数后才能正确工作。除了初始化 SDK，设置分辨率、帧率，由于本 SDK 不是完全免费，因此初始化还包含了验证授权的操作。

需要用到的函数（最小集合）：

```
BOOL  WINAPI RDLive_Init( LPCWSTR szOrganizationName, LPCWSTR szApplicationName,
                fnRenderNotifyCB pCbRender, fnEncoderNotifyCB pCbEncoder, LPVOID pCbParam );

BOOL  WINAPI RDLive_ResetAccredit( LPCSTR szAppKey, LPCSTR szAppSecret );
FLOAT  WINAPI RDLive_GetAccreditDays( ERdLiveAccreditType eAccreditType );

BOOL  WINAPI Render_SetSize( INT iWidth, INT iHeight );
BOOL  WINAPI Render_SetFps( FLOAT fps );
```

示例代码（C/C++）：

```
//初始化 SDK。公司名和产品名参数主要是为了生成保存各种文件的默认路径。
if ( FALSE == RDLive_Init( L"公司名", L" 应用名", NULL, NULL, NULL  ) )
{
	//错误处理
}
//验证 SDK 的授权信息。调用此接口时，如果已经联网，那么会从本公司网站异步获得最新的授权信息。
//注意，因为从网站获得最新授权信息是异步的，所以在更新成功后，会通过回调函数通知。
if ( FALSE == RDLive_ResetAccredit( L"在本公司网站上获得的 AppKey", L" 在本公司网站上获得的 AppSecret " ) )
{
	//错误处理
```

```
}
else
{
	//授权信息验证成功后，取得授权的剩余天数。
	 //当有回调通知授权信息更新时，可以再次调用 RDLive_GetAccreditDays 接口得到授权剩余天数。
	FLOAT fDays = RDLive_GetAccreditDays( eAccreditLocalSave );
}
//设置想要录制、直播的视频分辨率
if ( FALSE ==  Render_SetSize( 1280, 720 ) )
{
	//错误处理
}
//设置想要录制、直播的视频帧率
if ( FALSE ==  Render_SetFps( 25.0f ) )
{
	//错误处理
}
```

怎样得到 AppKey 和 AppSecret：

1. 打开网页 http://www.rdsdk.com/home/business/registers 注册开发者帐号。
2. 打开网页 http://dianbook.17rd.com/business/verify/login 登录刚才注册的帐号。
3. 点击网页左侧“应用管理”类目下的“我的应用”，再点击右侧的“新增”按钮，填入必要的资料。
4. 在“我的应用”栏目下，有刚才新增的应用信息，复制其中的 AppKey 和 AppSecret 到代码中。
5. 试用期结束时，可以点击“申请授权”，得到正式的使用授权。

2.3 设置画面内容

设置画面内容，包括创建场景、设置场景中元件的内容、设置元件的大小、位置等属性。为了快速入门，这里只做最简单的说明。

需要用到的函数（最小集合）：

```
HSCENE  WINAPI Render_CreateScene();
HCHIP  WINAPI Scene_CreateChip( HSCENE hScene );
BOOL  WINAPI Chip_Open( HCHIP hChip, LPCWSTR szTypeName, LPCWSTR szResource, BOOL bCannotReuse = FALSE, DWORD_PTR ptrParam = NULL );
BOOL  WINAPI Chip_SetVisible( HCHIP hChip, BOOL bVisible );
```

示例代码（C/C++）：

```
//创建一个场景。得到场景句柄，之后对该场景的操作，都通过句柄来完成。
HSCENE hScene = Render_CreateScene();
//创建一个元件。得到元件句柄。之后对该元件的操作，都通过句柄来完成。
HCHIP hChip = Scene_CreateChip( hScene );
//为元件设置图像来源。这里设置为录制第一个屏幕。
if ( FALSE == Chip_Open( hChip, L"Screen", L"0", FALSE,  NULL ) )
{
	//错误处理
}
//设置元件显示。新创建的元件默认是隐藏的，不会绘制，因此要设置为显示。
Chip_SetVisible( hChip, TRUE );
```

2.4 控制预览画面

为了便于用户操作，视频录制和直播软件最好是有预览画面，虽然没有预览画面也不影响录制和直播。而功能越复杂的视频软件，在预览画面上可以进行的操作也就越复杂，为了减少 SDK 用户的开发难度，SDK 内部集成了预览画面的显示，以及使用鼠标对预览画面内各个部分进行的操作。但 SDK 用户仍需要设置预览画面的显示区域，并且当用户设置不同的预览比例或者改变窗口大小时，都需要调用接口重设预览参数。

如下图所示，预览画面包括：当前场景、后台场景、场景滚动条。在预览画面之外，是程序界面的其它部分，当设置预览比例，或者改变主窗口大小时，预览区域的大小和位置都需要改变。

需要用到的函数（最小集合）：

```
BOOL  WINAPI Render_SetPreviewLayout( IGlRender_SPreviewLayout* pLayout  );
const IGlRender_SPreviewLayout* WINAPI Render_GetPreviewLayout();
HCURSOR WINAPI Render_SendPreviewMouseMessage( UINT uMsg, WPARAM wParam, int iX, int iY );
typedef  VOID    ( WINAPI *fnPreviewImageCB ) ( INT iWidth, INT iHeight, LPBYTE pImageBuffer, LPVOID pCbParam );
```

示例代码（C/C++）：

```
//假设已经存在以下变量：
//m_hMainWnd                        主界面的窗口句柄
//m_hPreviewWnd                 位于主界面中的显示预览画面的子窗口句柄
//OnPreviewBufferCB       用于接收预览图像的数据的回调函数
//声明一个设置预览画面布局的结构。
IGlRender_SPreviewLayout sLayout;
sLayout = *Render_GetPreviewLayout();   //获得当前的预览布局
//根据需要，修改预览布局结构中的相关成员。
sLayout.hMainWnd = m_hMainWnd;       //主窗口句柄
//注意：如果不是重设预览比例，而是用户改变了主窗口大小，这里的 fScale 应该设置为 0，其它的参数不变。
sLayout.fScale = 0.5f;       //预览的缩放比例是 50%
//设置计算重置主窗口的大小和位置时，位于屏幕中心。
sLayout.eReposWindow  = IGlRender_SPreviewLayout::eScreenCenter;
if ( bUseOpenGlWnd )       //如果使用 OpenGL 窗口进行预览
{
       //使用 OpenGL  窗口预览，实质是 SDK 会把创建的 OpenGL 预览渲染窗口设置为 sLayout.hPreviewWnd 的子窗口。
       sLayout.hPreviewWnd  = m_hPreviewWnd;         //显示预览画面的子窗口句柄
       //设置预览画面在子窗口上的实际显示区域，这里直接处理为整个子窗口都显示预览画面。
       GetClientRect( m_hPreviewWnd, sLayout.rtPreview );
        //把结构中的回调函数相关成员置 NULL
       sLayout.pCbFunction = NULL;
       sLayout.pCbParam= NULL;
}
else
```

```
{
        //不使用 OpenGL 窗口预览，而是能过回调函数传出预览画面的图像数据，由上层调用者自行绘制。
        sLayout.hPreviewWnd  = 0;        //显示预览画面的子窗口句柄设置为 0，pCbFunction  才会生效。
        //设置将要绘制预览画面的区域大小。
        GetClientRect( m_hPreviewWnd, sLayout.rtPreview );
        //设置回调，将会通过此回调函数来传出预览画面的图像数据。
        sLayout.pCbFunction = OnPreviewBufferCB;
        sLayout.pCbParam= this;
}
//设置预览画面布局。
if ( Render_SetPreviewLayout( &sLayout   ) )
{
        //如果设置布局成功，需要根据结构中返回的值重新设置主窗口的大小。
        MoveWindow( m_hMainWnd,
                sLayout.rtMainClient.left, sLayout.rtMainClient.top,                    //X,Y
                sLayout.rtMainClient.right - sLayout.rtMainClient.left,                 //宽度
                sLayout.rtMainClient.bottom - sLayout.rtMainClient.top );     //高度
}
```

其它说明：

- 如果是设置为使用 OpenGL 显示预览，那么不需要再做额外的处理。
  如果是设置为使用回调函数接收预览图像的数据，那么就需要处理图像的输出和鼠标操作的输入。
- 预览图像的输出，自己绘制到窗口上：

```
VOID WINAPI OnPreviewBufferCB( INT iWidth, INT iHeight, LPBYTE pImageBuffer, LPVOID pCbParam )
{
        //pImageBuffer 是与 GDI 兼容的 BGR32 的数据，
        //可以设置到 HBITMP 再绘制到 HDC 上，也可以用 GDI+ 的 Image 或 Qt 的 QImage 等来作为数据的容器。
}
```

- 鼠标的输入：
  当鼠标在预览画面的绘制区域移动、按下、放开时，请调用接口 Render_SendPreviewMouseMessage。

Render_SendPreviewMouseMessage 的返回值是 HCURSOR，因为鼠标移动到预览区域的不同位置，光标需要被更改为不同的状。
调用者应该使用 SetCursor 把光标设置为返回的 HCURSOR 句柄。

2.5 开始录制和直播

在进行视频编码之前，需要设置编码的各项参数，例如编码器、码率等。

需要用到的函数（最小集合）：

```
BOOL  WINAPI Encoder_SetCurrent( EVideoEncoder eEncoder );
BOOL  WINAPI Encoder_SetPreset( EVideoEncoder eEncoder, EVideoPreset ePreset );
BOOL  WINAPI Encoder_SetBitrate( EVideoRateMode eMode, INT iBitrate, INT iBitrateMax, INT iVbvSize );
BOOL  WINAPI Encoder_SetAudioParams( WORD nChannels, EAudioInSamples eSample, WORD wBits, EAudioEncoder eEncoder, INT
iEncBitsRate );
INT    WINAPI Encoder_AddSaveFile( SEncoderSaveFile* pSaveFile );
BOOL  WINAPI Encoder_Start();
```

示例代码（C/C++）：

```
//设置视频编码器为 x264。
Encoder_SetCurrent( VE_X264 );
//设置 x264 编码器的 Perset 为 Medium。
Encoder_SetPreset( VE_X264, VP_x264_Medium  );
//设置码率为 1000Kbps，模式为“平均码率”。
Encoder_SetBitrate( VR_AverageBitrate, 1000, 0, 0 );
//设置录音的参数为 2 声道，22050hz 16bit 采样率，AAC 编码，码率 96Kbps。
Encoder_SetAudioParams( 2, Aud_Inp_Samp_22050, 16, Aud_Enc_AAC, 96 );
//设置保存文件和直播信息的相关结构。
SEncoderSaveFile sFile;
ZeroMemory( &sFile, sizeof( sFile ) );
```

```
sFile.eContainer  = Mix_FLV;       //设置文件格式为 FLV
sFile.szSavePath = L"D:\\Movie\\abc.flv";       //设置文件保存路径
//添加到视频输出列表。
if ( -1 == Encoder_AddSaveFile(  &sFile ) )
{
       //失败可能是由于没有获得授权，或者路径格式不正确
       DWORD dwError = GetLastError();
}
//再设置一个输出，为 RTMP 直播。
sFile.eContainer  = Mix_RTMP;       //设置类型为 RTMP 直播
sFile.szSavePath = L"rtmp://192.168.0.1/live/abc";       //设置直播的 URL
if ( -1 == Encoder_AddSaveFile(  &sFile ) )
{
       //失败可能是由于没有获得授权，或者 URL 格式不正确
       DWORD dwError = GetLastError();
}
//开始录制和直播。
if ( FALSE == Encoder_Start() )
{
        //通常是由于没有设置任何输出才导致返回失败。因为打开文件和连接网络的操作等等都是异步的，如果失败会通过回调函数进行通知。
}
```

2.6 处理回调通知

SDK 中有两个最重的要回调通知，分别是场景编辑的回调，和视频编码存储上传的回调。

场景编辑的回调（IRender_ENotify eNotify）主要是以下内容：

- 场景添加、删除、切换。
- 场景中元件的添加、删除、切换。

- 场景中元件的大小、位置、层次等改变。
- 场景中元件其它属性的改变。

视频编码存储上传的回调（IEncoder_ENotify）主要是以下内容：

- 视频编码开始、停止（或由于出错被迫停止）。
- 视频编码的时间、帧率。
- 视频上传的速率、丢弃未上传的数据。
- 视频上传或文件写入过程中出错。

回调通知处理的注意事项：

- 回调函数被调用时，通常不在程序的界面线程中，因此最好不要在回调函数中直接更新界面。
- 不能直接在回调函数中调用 Encoder_Start()、Encoder_End()、RDLive_Init()、RDLive_Uninit()，因为可能会引起死锁。但可以把消息交给其它线程处理，并且不要在回调函数中等待完成。
- 如果使用 Win32 API 和 MFC 开发程序，应该在回调函数中使用 PostMessage() 把通知发送到界面线程中进行处理。
- 对于 Qt 应该使用自定义的 signals/slots 把通知交给界面线程处理。
- 对于其它类型的语言和开发环境，可以使用 PostMessage() 或其它的从非界面线程发送通知到界面线程的方式。

需要用到的函数（最小集合）：

```
BOOL  WINAPI RDLive_Init( LPCWSTR szOrganizationName, LPCWSTR szApplicationName,
                fnRenderNotifyCB pCbRender, fnEncoderNotifyCB pCbEncoder, LPVOID pCbParam );

typedef  VOID    ( WINAPI *fnRenderNotifyCB ) ( IRender_ENotify eNotify, HSCENE hScene, INT iValue, LPVOID pCbParam );
typedef  VOID  ( WINAPI*
fnEncoderNotifyCB )( IEncoder_ENotify  eNotify,  DWORD  dwValue,  DWORD_PTR  ptrValue,  LPVOID  pCbParam  );
```

示例代码（C/C++）：

- **MFC 示例，使用 Win32 API 也类似：**

```
//先预定义两个自定义消息，实际上将把作为一个消息范围来使用。
#define WM_GLRENDER_NOTIFY_BASE (WM_USER+0x100)
#define WM_ENCODER_NOTIFY_BASE (WM_USER+0x200)
//对话框初始化
BOOL CMyDialog::OnInitDialog()
{
	//……其它初始化……
	//在对话框或窗口的初始化中，调用初始化，并且设置回调。
	RDLive_Init( L"公司名", L"产品名 ", RenderNotifyCB, EncoderNotifyCB, this );
	//……其它初始化……
}
//场景编辑的回调，作为类成员函数时，声明时应该加上 static 标识，否则不能作为回调函数使用。
VOID WINAPI CMyDialog::RenderNotifyCB( IRender_ENotify eNotify, HSCENE hScene, INT iValue, LPVOID pCbParam )
{
	CMyDialog* pThis = (CMyDialog*)pCbParam;	//回调函数的回传参数，就是调用 Render_SetNotifyCallBack 时传入的 this 指针。
	//送到消息到窗口，实际发送的消息是 WM_GLRENDER_NOTIFY_BASE + eNotify。
	::PostMessage( pThis->m_hWnd, WM_GLRENDER_NOTIFY_BASE + eNotify, (WPARAM)hScene, (LPARAM)iValue );
}
//视频编码存储上传的回调，作为类成员函数时，声明时应该加上 static 标识，否则不能作为回调函数使用。
VOID WINAPI CMyDialog::EncoderNotifyCB( IEncoder_ENotify eNotify, DWORD dwValue, DWORD_PTR ptrValue, LPVOID pCbParam )
{
	CMyDialog* pThis = (CMyDialog*)pCbParam;	//回调函数的回传参数，就是调用 Render_SetNotifyCallBack 时传入的 this 指针。
	//送到消息到窗口，实际发送的消息是 WM_ENCODER_NOTIFY_BASE + eNotify。
	::PostMessage( pThis->m_hWnd, WM_ENCODER_NOTIFY_BASE + eNotify, (WPARAM)dwValue, (LPARAM)ptrValue);
}
//重写了 CWnd 的消息处理函数。
LRESULT CMyDialog::WindowProc( UINT message, WPARAM wParam, LPARAM lParam )
{
	if ( message >= WM_GLRENDER_NOTIFY_BASE
		&& message < WM_GLRENDER_NOTIFY_BASE + eNotify_Count )
	{
```

```
                //当消息值的范围是 IRender_ENotify 时，调用 IRender_ENotify  通知的处理函数。
                OnRenderNotifyCB( IRender_ENotify( message  - WM_GLRENDER_NOTIFY_BASE ), HSCENE( wParam  ), INT( lParam   ) );
        }
        else if ( message >= WM_ENCODER_NOTIFY_BASE
                && message < WM_ENCODER_NOTIFY_BASE + eEncNotify_Count )
        {
                //当消息值的范围是 IEncoder_ENotify 时，调用 IEncoder_ENotify 通知的处理函数。
                OnEncoderNotifyCB( IEncoder_ENotify( message  - WM_ENCODER_NOTIFY_BASE  ), DWORD( wParam  ),
DWORD_PTR( lParam   ) );
        }
        return CDialog::WindowProc( message , wParam , lParam );
}
//IRender_ENotify  通知的处理函数。
VOID CMyDialog::OnRenderNotifyCB( IRender_ENotify eNotify, HSCENE hScene, INT iValue )
{
        switch( eNotify )     //根据通知消息，进行不同的处理。
        {
        case .....
        }
}
//IEncoder_ENotify 通知的处理函数。
VOID CMyDialog::OnEncoderNotifyCB( IEncoder_ENotify eNotify, DWORD  dwValue,  DWORD_PTR  ptrValue )
{
        switch( eNotify )     //根据通知消息，进行不同的处理。
        {
        case .....
        }
}
```

- **Qt 4.8.x 示例：**

```
//对话框的构造函数
QMyDialog::QMyDialog( QWidget* parent )
{
```

```
	ui.setupUi(this);
	//……其它初始化……
	qRegisterMetaType<int>("IRender_ENotify");		//注册数据类型
	qRegisterMetaType<void*>("HSCENE");		//注册数据类型
	 //连接槽函数
	connect( this, SIGNAL(signalRenderNotifyCB( IRender_ENotify, HSCENE, int )), this, SLOT(OnRenderNotifyCB( IRender_ENotify, HSCENE, int )) );
	qRegisterMetaType<int>("IEncoder_ENotify");		//注册数据类型
	//连接槽函数
	connect( this, SIGNAL(signalEncoderNotifyCB( IEncoder_ENotify, ulong, ulong )), this, SLOT(OnEncoderNotifyCB( IEncoder_ENotify, ulong, ulong )) );
	//设置回调函数
	Render_SetNotifyCallBack( RenderNotifyCB, EncoderNotifyCB, this );
}
//场景编辑的回调，作为类成员函数时，声明时应该加上 static 标识，否则不能作为回调函数使用。
VOID WINAPI QMyDialog::RenderNotifyCB( IRender_ENotify eNotify, HSCENE hScene, INT iValue, LPVOID pCbParam )
{
	QMyDialog* pThis = (QMyDialog*)pCbParam;	//回调函数的回传参数，就是调用 Render_SetNotifyCallBack 时传入的 this 指针。
	emit pThis->signalRenderNotifyCB( eNotify, hScene, iValue );
}
//视频编码存储上传的回调，作为类成员函数时，声明时应该加上 static 标识，否则不能作为回调函数使用。
VOID WINAPI QMyDialog::EncoderNotifyCB( IEncoder_ENotify eNotify, DWORD dwValue, DWORD_PTR ptrValue, LPVOID pCbParam )
{
	QMyDialog* pThis = (QMyDialog*)pCbParam;	//回调函数的回传参数，就是调用 Render_SetNotifyCallBack 时传入的 this 指针。
	emit pThis->signalEncoderNotifyCB( eNotify, dwValue, ptrValue );
}
//IRender_ENotify  通知的处理函数。
VOID QMyDialog::OnRenderNotifyCB( IRender_ENotify eNotify, HSCENE hScene, INT iValue )
{
	switch( eNotify )	//根据通知消息，进行不同的处理。
	{
	case .....
```

```
        }
}
//IEncoder_ENotify 通知的处理函数。
VOID QMyDialog::OnEncoderNotifyCB( IEncoder_ENotify eNotify, ulong dwValue,  ulong  ptrValue )
{
        switch( eNotify )     //根据通知消息，进行不同的处理。
        {
        case .....
        }
}
```

3. 初始化 SDK

初始化 SDK

RDLive_Init	初始化 SDK，其它的接口都需要在初始化成功之后才能调用。
RDLive_Uninit	反初化 SDK，关闭所有打开的资源。
RDLive_ResetAccredit	验证授权，SDK 根据授权的验证结果，启用对应的功能。
RDLive_GetAccreditDays	取得剩余的授权天数。
RDLive_DefaultDir	取得指定类型的默认文件夹路径。
fnRenderNotifyCB	场景回调函数。场景控制、元件设置等信息状态变化时，向上层发起通知的回调函数。
fnEncoderNotifyCB	编码回调函数。编码器进行音视频编码、文件保存、直播等过程中，发生错误或状态改变时，向上层发起通知的回调函数。
ERdLiveAccreditType	SDK 授权的的类型，可以使用 RDLive_GetAccreditDays 获得指定类型授权的剩余天数。

ERecDefaultDirs	各种默认路径的文件夹类型。
IRender_ENotify	场景回调函数向上层发起通知的类型。
IEncoder_ENotify	编码回调函数向上层发起通知的类型。

3.1 RDLive_Init

初始化 SDK，其它的接口都需要在初始化成功之后才能调用。

语法

```
BOOL WINAPI RDLive_Init(
        IN              LPCWSTR szOrganizationName,
        IN              LPCWSTR szApplicationName,
        IN              fnRenderNotifyCB pCbRender,
        IN              fnEncoderNotifyCB pCbEncoder,
        IN              LPVOID pCbParam
);
```

参数

szOrganizationName

公司、组织名称。如果为 NULL 或空字符串，则默认使用 L"17rd"。

szApplicationName

应用程序名称。如果为 NULL 或空字符串，则默认使用 L"RDLive"。

pCbRender

场景回调函数。当场景控制、元件设置等信息状态变化时，向上层发起通知的回调函数指针。

详细信息请参阅 IRender_ENotify 和 fnRenderNotifyCB 的说明。

pCbEncoder

编码回调函数。编码器进行音视频编码、文件保存、直播等过程中，发生错误或状态改变时，向上层发起通知的回调函数指针。

详细信息请参阅 IEncoder_ENotify 和 fnEncoderNotifyCB 的说明。

pCbParam

回调参数。在调用回调函数时，会把该参数原样传出。通常应用程序把 this 指针作为回调参数。

返回值

成功返回 TRUE，失败返回 FALSE。

如果失败，可以使用 GetLastError() 取得错误代码。

备注

必须首先调用 RDLive_Init 接口之后，才能调用其它接口。

传入的“公司名”、“应用名”参数，主要用于在调用 RDLive_DefaultDir 接口获取各个默认文件夹路径时，把公司名等拼接在路径中。

请参阅

3.2 RDLive_Uninit

反初化　SDK，关闭所有打开的资源。

语法

```
VOID WINAPI RDLive_Uninit();
```

参数

无

返回值

无

备注

在程序关闭之前，应该调用本函数进行反初始化。

不要在退出程序所引起的应用程序主窗口对象析构或 DLL 的 DLL_PROCESS_DETACH 时才反初始化，因为这个时候可能有的对象实例已经被迫析构了，而有的线程可能还在工作，所以反初始化 SDK 时有可能造成崩溃或死锁。

对于有窗口的程序，应该是在程序的消息循环结束之前进行 SDK 的反初始化，也就是程序的主窗口收到了关闭消息，但还没有关闭完成的时候。例如 MFC 窗口的 OnClose() 被调用时、Qt 窗口的 closeEvent() 被调用时、Win32 API 程序窗口收到 WM_CLOSE 消息时等等。

对于没有窗口和消息循环的程序，则应该是在 main() 函数返回之前，而不要在某个类对象的析构中进行。

请参阅

RDLive_Init
fnRenderNotifyCB
fnEncoderNotifyCB
IRender_ENotify
IEncoder_ENotify
RDLive_DefaultDir

3.3 RDLive_ResetAccredit

验证授权，SDK 根据授权的验证结果，启用对应的功能。

语法

```
BOOL WINAPI RDLive_ResetAccredit(
        IN              LPCSTR szAppKey,
```

```
        IN              LPCSTR szAppSecret
);
```

参数

szAppKey
在本公司网站 http://www.rdsdk.com 注册开发者帐号，并新建应用后获得的 AppKey。

szAppSecret
在本公司网站 http://www.rdsdk.com 注册开发者帐号，并新建应用后获得的 AppSecret。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

应该在 RDLive_Init() 调用返回成功后，再调用本函数。

SDK 在验证授权完成之前，不能保存视频文件，不能上传视频流，但其它的各个接口都能正常调用。

验证时会异步从网络服务器获取授权信息，验证完成通过 fnEncoderNotifyCB 回调发起 eEncNotify_AcceditDone 通知。

请参阅

RDLive_Init
fnEncoderNotifyCB
RDLive_Uninit
RDLive_GetAccreditDays
ERdLiveAccreditType

3.4 RDLive_GetAccreditDays

取得剩余的授权天数。

语法

```
FLOAT WINAPI RDLive_GetAccreditDays(
        IN          ERdLiveAccreditType eAccreditType
);
```

参数

eAccreditType
需要得到剩余天数的授权功能。

为以下的值之一：
eAccreditLocalSave: 本地保存视频文件。
eAccreditRtmpLive: RTMP 直播视频流。
eAccreditCloudLive: 使用锐动云服务器直播视频流。

返回值

天数。

备注

应该在调用 RDLive_ResetAccredit 并得到 eEncNotify_AcceditDone 通知后再调用本函数，否则返回值为 0。

请参阅

RDLive_ResetAccredit
ERdLiveAccreditType
RDLive_Init
RDLive_Uninit

3.5 RDLive_DefaultDir

取得指定类型的默认文件夹路径。

语法

```
LPCWSTR WINAPI RDLive_DefaultDir(
        IN            ERecDefaultDirs eDir
);
```

参数

eDir

需要得到默认路径的文件夹类型。

为以下的值之一：

RDir_Application	程序本身所在的文件夹。
RDir_Profile	存放配置文件的默认文件夹。例如：C:\Users\UserName\AppData\Local\17rd\RDLive
RDir_Videos	存放视频文件的默认文件夹。例如：C:\Users\UserName\Videos\RDLive
RDir_Audios	存放音频文件的默认文件夹。例如：C:\Users\UserName\Music\RDLive
RDir_Images	存放图片文件的默认文件夹。例如：C:\Users\UserName\Pictures\RDLive

返回值

成功返回路径字符串指针，失败返回 NULL。

备注

返回的指针不需要也不应该由调用者释放指向的内存。

请参阅

ERecDefaultDirs
RDLive_Init
RDLive_Uninit

3.6 fnRenderNotifyCB

场景回调函数。场景控制、元件设置等信息状态变化时，向上层发起通知的回调函数。

语法

```
VOID WINAPI fnRenderNotifyCB(
        IRender_ENotify eNotify,
        HSCENE hScene,
        INT iValue,
        LPVOID pCbParam
);
typedef VOID ( WINAPI * fnRenderNotifyCB )(
        IRender_ENotify eNotify,
        HSCENE hScene,
        INT iValue,
        LPVOID pCbParam
);
```

参数

eNotify

枚举，通知类型值。

详细信息请参阅 IRender_ENotify 的说明。

hScene

当前通知相关的场景的句柄，可能为 NULL，与 eNotify 通知的类型有关。

iValue

回调的值，和 eNotify 相关，通知类型不同的时候，值的含义不同。
详细信息请参阅 IRender ENotify 的说明。

pCbParam

回调函数的回传参数，就是在调用 RDLive_Init 时传入的 pCbParam。
请参阅 RDLive_Init 的说明。

返回值

无

备注

回调函数被调用时，通常不在程序的界面线程中，因此最好不要在回调函数中直接更新界面。

不能直接在回调函数中调用 Encoder_Start()、Encoder_End()、RDLive_Init()、RDLive_Uninit()，因为可能会引起死锁。但可以把消息交给其它线程处理，并且不要在回调函数中等待完成。

如果使用 Win32 API 和 MFC 开发程序，应该在回调函数中使用 PostMessage() 把通知发送到界面线程中进行处理。

对于 Qt 应该使用自定义的 signals/slots 把通知交给界面线程处理。

对于其它类型的语言和开发环境，可以使用 PostMessage() 或其它的从非界面线程发送通知到界面线程的方式。

请参阅

RDLive_Init
RDLive_Uninit
IRender_ENotify
Encoder_Start
Encoder_End
fnEncoderNotifyCB
IEncoder_ENotify
RDLive_DefaultDir

3.7 fnEncoderNotifyCB

编码回调函数。编码器进行音视频编码、文件保存、直播等过程中，发生错误或状态改变时，向上层发起通知的回调函数。

语法

```
VOID WINAPI fnEncoderNotifyCB(
        IEncoder_ENotify eNotify,
        DWORD dwValue,
        DWORD_PTR ptrValue,
        LPVOID pCbParam
);
typedef VOID ( WINAPI * fnEncoderNotifyCB )(
        IEncoder_ENotify eNotify,
        DWORD dwValue,
```

```
        DWORD_PTR ptrValue,
        LPVOID pCbParam
);
```

参数

eNotify

枚举，通知类型值。

详细信息请参阅 IEncoder_ENotify 的说明。

dwValue

回调的值，和 eNotify 相关，通知类型不同的时候，值的含义不同。

详细信息请参阅 IEncoder_ENotify 的说明。

ptrValue

回调的值，和 eNotify 相关，通知类型不同的时候，值的含义不同。

详细信息请参阅 IEncoder_ENotify 的说明。

pCbParam

回调函数的回传参数，就是在调用 RDLive_Init 时传入的 pCbParam。

请参阅 RDLive_Init 的说明。

返回值

无

备注

回调函数被调用时，通常不在程序的界面线程中，因此最好不要在回调函数中直接更新界面。

不能直接在回调函数中调用 Encoder_Start()、Encoder_End()、RDLive_Init()、RDLive_Uninit()，因为可能会引起死锁。但可以把消息交给其它线程处理，并且不要在回调函数中等待完成。

如果使用 Win32 API 和 MFC 开发程序，应该在回调函数中使用 PostMessage() 把通知发送到界面线程中进行处理。

对于 Qt 应该使用自定义的 signals/slots 把通知交给界面线程处理。

对于其它类型的语言和开发环境，可以使用 PostMessage() 或其它的从非界面线程发送通知到界面线程的方式。

请参阅

RDLive_Init
RDLive_Uninit
IEncoder_ENotify
Encoder_Start
Encoder_End
fnRenderNotifyCB
IRender_ENotify
RDLive_DefaultDir

3.8 ERdLiveAccreditType

SDK 授权的的类型，可以使用 RDLive_GetAccreditDays 获得指定类型授权的剩余天数。

语法

```
enum ERdLiveAccreditType
{
        eAccreditLocalSave,
        eAccreditRtmpLive,
        eAccreditCloudLive
};
```

成员

eAccreditLocalSave
本地保存视频文件。

eAccreditRtmpLive
RTMP 直播视频流。

eAccreditCloudLive
使用锐动云服务器直播视频流。

备注

应该在调用 RDLive_ResetAccredit 并得到 eEncNotify_AcceditDone 通知后再调用 RDLive_GetAccreditDays 函数，否则返回值为 0。

请参阅

RDLive_ResetAccredit
RDLive_GetAccreditDays
RDLive_Init
RDLive_Uninit

3.9 ERecDefaultDirs

各种默认路径的文件夹类型。

语法

```
enum ERecDefaultDirs
{
        RDir_Application,
        RDir_Profile,
        RDir_Videos,
        RDir_Audios,
        RDir_Images
};
```

成员

RDir_Application
程序本身所在的文件夹。

RDir_Profile

存放配置文件的默认文件夹。
例如：C:\Users\UserName\AppData\Local\17rd\RDLive

RDir_Videos

存放视频文件的默认文件夹。
例如：C:\Users\UserName\Videos\RDLive

RDir_Audios

存放音频文件的默认文件夹。
例如：C:\Users\UserName\Music\RDLive

RDir_Images

存放图片文件的默认文件夹。
例如：C:\Users\UserName\Pictures\RDLive

备注

在调用 RDLive Init 之后，再用 RDLive DefaultDir 函数获取文件夹路径字符串。

请参阅

RDLive Init
RDLive DefaultDir
RDLive Uninit

3.10 IRender_ENotify

场景回调函数向上层发起通知的类型。

语法

```
enum IRender_ENotify
{
        eNotify_None,
        eNotify_SceneAdding,
        eNotify_SceneAdded,
        eNotify_SceneDeleting,
        eNotify_SceneDeleted,
        eNotify_SceneSwitched,
        eNotify_SceneNamed,
        eNotify_SceneScroll,
        eNotify_RBDownCScene,
        eNotify_RBDownBScene,
        eNotify_RBUpCScene,
        eNotify_RBUpBScene,
        eNotify_ChipAdding,
        eNotify_ChipAdded,
        eNotify_ChipDeleting,
        eNotify_ChipDeleted,
        eNotify_ChipPosing,
        eNotify_ChipPosed,
```

```
        eNotify_ChipRotating,
        eNotify_ChipRotated,
        eNotify_ChipSwitched,
        eNotify_ChipIndexChg,
        eNotify_ChipStatus,
        eNotify_ChipVisible,
        eNotify_ChipViewLock,
        eNotify_ChipSourceSize,
        eNotify_CameraDevChanged,
        eNotify_AudioDevChanged,
        eNotify_AudioImmVolume,
        eNotify_Count
};
```

成员

eNotify_None
无。回调函数不会发起这个类型的通知。

eNotify_SceneAdding
用户点击了预览画面中的“添加场景”按钮。得到此通知时，上层如果不调用 Render_CreateScene，就不会添加场景。
hScene：NULL 值。
iValue：已有场景的数量（添加前）。

eNotify_SceneAdded
一个场景添加完成。

hScene：当前添加的场景的句柄。
iValue：是否影响到了预览布局。0　表示无影响，其它值表示布局改变，请重新设置界面。

eNotify_SceneDeleting
用户点击了预览画面中的“删除场景”按钮。得到此通知时，上层如果不调用　Render DestroyScene，就不会删除场景。
hScene：将要被删除的场景的句柄。
iValue：当前场景的数量（删除前）。

eNotify_SceneDeleted
一个场景删除完成。
hScene：当前删除的场景的句柄。
iValue：是否影响到了预览布局。0　表示无影响，其它值表示布局改变，请重新设置界面。

eNotify_SceneSwitched
切换了场景，选中了一个场景作为当前场景。
hScene：当前被选中的场景的句柄，如果为　NULL　则表示没有任何场景被选中。
iValue：0　值。

eNotify_SceneNamed
场景的名字发生了改变。
hScene：名字发生改变的场景的句柄。
iValue：0　值。

eNotify_SceneScroll
拖动了后台场景预览区域的滚动条，或者滚动条的值范围发生了改变。
hScene：后台场景区域当前显示的第一个场景的句柄。
iValue：低 16 位（WORD）是滚动条的最大值，高 16 位（WORD）滚动条的当前值。

eNotify_RBDownCScene

用户在预览图面的“当前场景”显示范围中【按下】了鼠标右键。

hScene：当前场景的句柄。

iValue：鼠标所在的元件的索引号，-1　表示没有在任何元件上。

eNotify_RBDownBScene

用户在预览图面的“背景场景”显示范围中【按下】了鼠标右键。

hScene：鼠标按下的场景的句柄，NULL　表示鼠标在后台场景显示区域的空白处，没有在任何场景上。

iValue：0　值。

eNotify_RBUpCScene

用户在预览图面的“当前场景”显示范围中【放开】了鼠标右键。

hScene：当前场景的句柄。

iValue：鼠标所在的元件的索引号，-1　表示没有在任何元件上。

eNotify_RBUpBScene

用户在预览图面的“背景场景”显示范围中【放开】了鼠标右键。

hScene：鼠标放开的场景的句柄，NULL　表示鼠标在后台场景显示区域的空白处，没有在任何场景上。

iValue：0　值。

eNotify_ChipAdding

用户在预览画面了点击了“添加元件”的按钮，得到此通知时，上层如果不调用　Scene_CreateChip，就不会添加元件。

hScene：将要添加元件的场景的句柄。

iValue：将要添加的元件的索引号。

eNotify_ChipAdded

一个元件添加完成。

hScene：添加元件的场景的句柄。
iValue：添加的元件的索引号，即场景中的元件数量。

eNotify_ChipDeleting
用户在预览画面了点击了“删除元件”的按钮，得到此通知时，上层如果不调用 Chip Destroy，就不会删除元件。
hScene：将要删除元件的场景的句柄。
iValue：将要删除的元件的索引号。

eNotify_ChipDeleted
一个元件删除完成。
hScene：删除元件的场景的句柄。
iValue：删除的元件的索引号。

eNotify_ChipPosing
场景中的元件位置或大小正在被用户改变(鼠标操作)。
hScene：当前场景的句柄。
iValue：被操作的元件的索引号。

eNotify_ChipPosed
场景中的元件位置或大小被用户改变完成(用户放开鼠标按钮，结束了操作)。
hScene：当前场景的句柄。
iValue：元件的索引号。

eNotify_ChipRotating
场景中的元件正在被用户旋转(鼠标操作)。
hScene：当前场景的句柄。
iValue：元件的索引号。

eNotify_ChipRotated

场景中的元件被用户旋转完成(用户放开鼠标按钮，结束了操作)。

hScene：当前场景的句柄。

iValue：元件的索引号。

eNotify_ChipSwitched

切换了场景中当前选中的元件。无论是由于用户点击，还是调用 Chip_SetCurent 造成的当前元件切换，都会触发此通知。

另外，如果当前元件被删除，也会触发此通知，此时没有任何元件被选中。

hScene：元件所在的场景的句柄。

iValue：低 16 位（SHORT）当前选中的元件的索引号，如果索引号为 -1，表示没有任何元件被选中。

高 16 位（SHORT）上次选中的元件的索引号，如果索引号为 -1，表示之前没有任何元件被选中。

eNotify_ChipIndexChg

场景中的元件的顺序发生了改变。元件的顺序就是显示顺序，最先添加的元件最先绘制，也就是说最后添加的元件在画面上显示在最前面。

hScene：元件所在的场景的句柄。

iValue：低 16 位（SHORT）元件之前的索引号。

高 16 位（SHORT）当前被插入到的位置（索引号）。

eNotify_ChipStatus

元件的源的状态发生改变。请参阅 IPinInput_EChipStatus 的相关说明。

hScene：元件所在的场景的句柄。

iValue：元件的索引号。

eNotify_ChipVisible

元件的可见状态发生改变。即显示或隐藏。

hScene：元件所在的场景的句柄。
iValue：元件的索引号。

eNotify_ChipViewLock

元件的位置、大小、角度锁定状态发生改变。当元件的位置被锁定时，不能用鼠标拖动位置。其它锁定也类似，不能用鼠标在预览画面上操作，但可以调用相关的函数直接设置值。

hScene：元件所在的场景的句柄。
iValue：元件的索引号。

eNotify_ChipSourceSize

元件的源的分辨率发生改变。例如截取桌面上的窗口时，窗口的大小发生了改变。

hScene：元件所在的场景的句柄。
iValue：元件的索引号。

eNotify_CameraDevChanged

摄像头设备有变化，比如拔出或插入了 USB 摄像头。

hScene：NULL 值。
iValue：当前的摄像头数量。

eNotify_AudioDevChanged

音频设备有变化，比如拔出或插入了 USB 声卡。现在的很多台式机主板，如果扬声器插孔上没有插音箱，麦克风插孔上没有插话筒，那么对应的音频设备也是不可用的状态，当插入了音箱和话筒，也会导致音频设备变化。

hScene：NULL 值。
iValue：0 值。

eNotify_AudioImmVolume

当前录音的音频设备的实时音量，也就是正在播放的声音的大小，或者麦克风输入声音的大小。

hScene：NULL　值。
iValue：低 16 位（WORD）是左声道音量(0-65535)，高 16 位是右声道音量(0-65535)。

eNotify_Count
通知类型的数量。回调时不会使用这个类型，仅用来作为通知类型数量的计数器。

请参阅

Render_CreateScene
Render_DestroyScene
Scene_CreateChip
Chip_Destroy
Chip_SetCurent
IPinInput_EChipStatus
RDLive_Init
RDLive_Uninit
fnRenderNotifyCB
fnEncoderNotifyCB
IEncoder_ENotify

3.11 IEncoder_ENotify

编码回调函数向上层发起通知的类型。

语法

```
enum IEncoder_ENotify
{
        eEncNotifySuccess,
        eEncNotifyStarted,
        eEncNotifyStoped,
        eEncNotifyEncodeError,
        eEncNotifyEncodeFps,
        eEncNotifyWriteFileFail,
        eEncNotifyDisconnected,
        eEncNotifyReConnectStart,
        eEncNotifyReConnectDone,
        eEncNotifyReConnectFail,
        eEncNotifyUploadBitrate,
        eEncNotifyDiscardPacks,
        eEncNotifyDiscardFrame,
        eEncNotifyResetBitrate,
        eEncNotifyResetPreset,
        eEncNotify_AcceditDone,
        eEncNotify_GetCloudFail,
        eEncNotify_Count
};
```

成员

eEncNotifySuccess
操作成功完成。回调函数不发起这个类型的通知。

eEncNotifyStarted

已经开始进行编码。

dwValue： 0 值。

ptrValue：0 值。

eEncNotifyStoped

已经停止了编码，包括由于各种错误导致的被迫停止。

dwValue： 如果是用户停止，那么值为 0，否则是出错引起停止的错误代码值。

ptrValue：0 值。

eEncNotifyEncodeError

编码器错误，例如编码过程中编码器返回失败。

dwValue： 错误代码。

ptrValue：0 值。

eEncNotifyEncodeFps

编码的帧率等信息，大约每秒计算并通知一次。

dwValue： 0 值。

ptrValue：SEncStatusEncodeFps 结构的指针，上层需要把该值强制转换为指针类型 PSEncStatusEncodeFps。

eEncNotifyWriteFileFail

写入文件失败。这个严重错误，将导致编码停止。

dwValue： 出错的文件在输出列表中的索引号，可以使用 Encoder_GetSaveFileInfo 取得该输出项的信息。

ptrValue：错误代码。

eEncNotifyDisconnected

上传时连接被断开。这不是严重错误，SDK 中会自动重试连接。

dwValue： 上传目标在输出列表中的索引号，可以使用 Encoder_GetSaveFileInfo 取得该输出项的信息。
ptrValue：错误代码。

eEncNotifyReConnectStart
在网络连接被断开后，开始重新连接。
dwValue： 上传目标在输出列表中的索引号，可以使用 Encoder_GetSaveFileInfo 取得该输出项的信息。
ptrValue：当前的重连计数，从 1 开始。

eEncNotifyReConnectDone
重新连接成功。
dwValue： 上传目标在输出列表中的索引号，可以使用 Encoder_GetSaveFileInfo 取得该输出项的信息。
ptrValue：当前的重连计数，从 1 开始。回调函数返回后，计数器会重置为 0。

eEncNotifyReConnectFail
重新连接失败。
dwValue： 上传目标在输出列表中的索引号，可以使用 Encoder_GetSaveFileInfo 取得该输出项的信息。
ptrValue：SEncStatusConnectFail 结构的指针，上层需要把该值强制转换为指针类型 PSEncStatusConnectFail。

eEncNotifyUploadBitrate
当前的上传速率，大约每秒计算并通知一次。
dwValue： 上传目标在输出列表中的索引号，可以使用 Encoder_GetSaveFileInfo 取得该输出项的信息。
ptrValue：SEncStatusUploadBitrate 结构的指针，上层需要把该值强制转换为指针类型 PSEncStatusUploadBitrate。

eEncNotifyDiscardPacks
因为上传缓慢，丢弃了部分编码后的视频数据。
dwValue： 上传目标在输出列表中的索引号，可以使用 Encoder_GetSaveFileInfo 取得该输出项的信息。
ptrValue：SEncStatusDiscardPacks 结构的指针，上层需要把该值强制转换为指针类型 PSEncStatusDiscardPacks。

eEncNotifyDiscardFrame

因为编码缓慢，丢弃了输入的一帧画面。

dwValue：　丢弃的帧的编号。

ptrValue：0　值。

eEncNotifyResetBitrate

因为上传缓慢，自动重设了视频的输出码率。

dwValue：　重设后的码率（kbit/s）。

ptrValue：0　值。

eEncNotifyResetPreset

因为编码缓慢，自动重设了视频的编码质量(仅对于　x264　编码)。

dwValue：　重设后的　EVideoPreset_x264。

ptrValue：0　值。

eEncNotify_AcceditDone

授权验证完成。

dwValue：　是否成功，TRUE　表示成功，FALSE　表示失败。

ptrValue：失败的文字说明，需要强制转换为指针　LPCWSTR。

eEncNotify_GetCloudFail

取得　Mix_RdCloud　云直播地址失败。

dwValue：　上传目标在输出列表中的索引号，可以使用　Encoder_GetSaveFileInfo　取得该输出项的信息。

ptrValue：失败的文字说明，需要强制转换为指针　LPCWSTR。

eEncNotify_Count

通知类型的数量。回调时不会使用这个类型，仅用来作为通知类型数量的计数器。

请参阅

Encoder_GetSaveFileInfo
SEncStatusEncodeFps
SEncStatusUploadBitrate
SEncStatusConnectFail
SEncStatusDiscardPacks
EVideoPreset_x264
RDLive_Init
RDLive_Uninit
fnRenderNotifyCB
fnEncoderNotifyCB
IRender_ENotify

4. 画面和预览

画面和预览

Render_SetSize	设置画面的分辨率，单位为像素。不是预览分辨率，而是生成视频的分辨率。
Render_GetSize	获取画面的分辨率。不是预览分辨率，而是生成视频的分辨率。
Render_SetFps	设置画面的帧率。
Render_GetFps	取得画面的帧率。
Render_SetPreviewLayout	设置预览画面的布局和显示预览画面的方式。

Render_GetPreviewLayout	取得当前的预览布局设置。
Render_SendPreviewMouseMessage	当使用回调预览图像数据自行绘制预览画面时，鼠标在预览画面区域移动、按下时，通知 SDK 进行相应的响应。
Render_SetBSceneScroll	设置预览区域后台场景滚动条的位置。
Render_GetBSceneScroll	获得预览区域后台场景滚动条的位置。
fnPreviewImageCB	回调函数，输出预览画面的图像数据。
IGlRender_SPreviewLayout	设置预览画面的布局和显示预览画面方式的结构。

4.1 Render_SetSize

设置画面的分辨率，单位为像素。不是预览分辨率，而是生成视频的分辨率。

语法

```
BOOL WINAPI Render_SetSize(
        IN              INT iWidth,
        IN              INT iHeight
);
```

参数

iWidth

宽度，横向的像素数量。值范围为 8 到 4096。

iHeight
高度，纵向的像素数量。值范围为 8 到 4096。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_GetSize
Render_SetFps
Render_GetFps

4.2 Render_GetSize

获取画面的分辨率。不是预览分辨率，而是生成视频的分辨率。

语法

```
SIZE WINAPI Render_GetSize();
```

参数

无

返回值

分辨率，SIZE　结构。

请参阅

Render_SetSize
Render_SetFps
Render_GetFps

4.3 Render_SetFps

设置画面的帧率。

语法

```
BOOL WINAPI Render_SetFps(
        IN              FLOAT fps
);
```

参数

fps
帧率，大于等于　1　并小于等于　120　的浮点数。

返回值

成功返回　TRUE，失败返回　FALSE。
如果失败，可以使用　GetLastError()　取得错误代码。

请参阅

Render_SetSize
Render_GetSize
Render_GetFps

4.4 Render_GetFps

取得画面的帧率。

语法

```
FLOAT WINAPI Render_GetFps();
```

参数

无

返回值

当前的帧率设置。

请参阅

Render_SetSize
Render_GetSize
Render_SetFps

4.5 Render_SetPreviewLayout

设置预览画面的布局和显示预览画面的方式。

语法

```
BOOL WINAPI Render_SetPreviewLayout(
        IN          IGlRender_SPreviewLayout* pLayout
);
```

参数

pLayout

预览布局结构的指针，可以为 NULL。
这个结构中有部分成员变量在计算过程中会发生改变，作为返回的数据。
当为 NULL 时，将恢复为 SDK 中内置的默认值。
详细信息请参阅 IGlRender_SPreviewLayout 结构的说明。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

IGlRender_SPreviewLayout
Render_GetPreviewLayout

Render_SendPreviewMouseMessage
Render_SetBSceneScroll
Render_GetBSceneScroll
fnPreviewImageCB

4.6 Render_GetPreviewLayout

取得当前的预览布局设置。

语法

```
const IGlRender_SPreviewLayout* WINAPI Render_GetPreviewLayout();
```

参数

无

返回值

成功返回只读的预览布局结构的指针，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

IGlRender_SPreviewLayout
Render_SetPreviewLayout
Render_SendPreviewMouseMessage

Render_SetBSceneScroll
Render_GetBSceneScroll
fnPreviewImageCB

4.7 Render_SendPreviewMouseMessage

当使用回调预览图像数据自行绘制预览画面时，鼠标在预览画面区域移动、按下时，通知 SDK 进行相应的响应。

语法

```
HCURSOR WINAPI Render_SendPreviewMouseMessage(
        IN            UINT uMsg,
        IN            WPARAM wParam,
        IN            int iX,
        IN            int iY
);
```

参数

uMsg

当前的鼠标消息。可以是以下值中的一个：

WM_MOUSEMOVE
WM_LBUTTONDOWN
WM_LBUTTONUP
WM_RBUTTONDOWN
WM_RBUTTONUP

WM_MBUTTONDOWN
WM_MBUTTONUP
WM_XBUTTONDOWN
WM_XBUTTONUP
WM_MOUSEWHEEL
WM_MOUSEHWHEEL

wParam

收到鼠标消息时的 wParam 参数。

iX

从鼠标消息的 lParam 参数解析出 x 坐标后，再计算出以绘制的预览画面左上角为原点的坐标。

iY

从鼠标消息的 lParam 参数解析出 y 坐标后，再计算出以绘制的预览画面左上角为原点的坐标。

返回值

返回鼠标光标句柄，可能是 0。

当鼠标在预览画面的各个区域移动时，鼠标光标可能需要改变，这里返回当前的鼠标光标句柄，上层应该在合适的时机设置鼠标光标。

如果返回 0，上层应该取消鼠标光标的自定义设置。

备注

只有当上层不使用本 SDK 内置的 OpenGL 绘图窗口来显示预览画面，而是通过回调函数获得预览画面的图像数据并自行绘制的时候，才需要使用本函数。如果使用了内置的 OpenGL 绘图窗口，就不需要调用本函数。

设置鼠标光标的说明（假设 hCursor 是本函数的返回值）：

MFC 重写 WM_SETCURSOR 消息的处理函数，在 CWnd 的 OnSetCursor() 的调用 SetCursor(hCursor) 函数。

Win32 API 直接在收到 WM_SETCURSOR 消息时，调用 SetCursor(hCursor) 函数。

Qt 在本函数返回时，就可以直接调用 setCursor(QCursor(hCursor))。

请参阅

Render_SetPreviewLayout
Render_GetPreviewLayout
Render_SetBSceneScroll
Render_GetBSceneScroll
IGlRender_SPreviewLayout
fnPreviewImageCB

4.8 Render_SetBSceneScroll

设置预览区域后台场景滚动条的位置。

语法

```
BOOL WINAPI Render_SetBSceneScroll(
        IN              INT iValue
);
```

参数

iValue
滚动条的值。最小值为 0，最大值为场景数减去显示的后台场景数。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_SetPreviewLayout
Render_GetPreviewLayout
Render_SendPreviewMouseMessage
Render_GetBSceneScroll
IGlRender_SPreviewLayout
fnPreviewImageCB

4.9 Render_GetBSceneScroll

获得预览区域后台场景滚动条的位置。

语法

```
INT WINAPI Render_GetBSceneScroll();
```

参数

无

返回值

当前滚动条的值。

请参阅

Render_SetPreviewLayout
Render_GetPreviewLayout
Render_SendPreviewMouseMessage
Render_SetBSceneScroll
IGlRender_SPreviewLayout
fnPreviewImageCB

4.10 fnPreviewImageCB

回调函数，输出预览画面的图像数据。

语法

```
VOID WINAPI fnPreviewImageCB(
        INT iWidth,
        INT iHeight,
        LPBYTE pImageBuffer,
```

```
        LPVOID pCbParam
);
typedef VOID ( WINAPI * fnPreviewImageCB )(
        INT iWidth,
        INT iHeight,
        LPBYTE pImageBuffer,
        LPVOID pCbParam
);
```

参数

iWidth
图像的宽度，单位为像素。

iHeight
图像的宽度，单位为像素。

pImageBuffer
图像的数据，是与 GDI 兼容的 RGB32 位图像数据。

pCbParam
回调参数，IGlRender SPreviewLayout 结构中 pCbParam 设置的值。

返回值

无

备注

只有当调用 Render_SetPreviewLayout 时设置的 IGlRender_SPreviewLayout 结构中 pCbFunction 有值且 hPreviewWnd 为 NULL 时才会调用此回调。

请参阅

Render_SetPreviewLayout
IGlRender_SPreviewLayout
Render_GetPreviewLayout
Render_SendPreviewMouseMessage

4.11 IGlRender_SPreviewLayout

设置预览画面的布局和显示预览画面方式的结构。

语法

```
struct IGlRender_SPreviewLayout
{
        IN          INT iVersion;
        IN OUT    FLOAT fScale;
        IN          HWND hMainWnd;
        IN          HWND hPreviewWnd;
        IN OUT    RECT rtPreview;
        IN          SIZE siMinimumLimit;
```

```
IN          fnPreviewImageCB pCbFunction;
IN          LPVOID pCbParam;
              enum  ELocBScene
              {
                       eUp,
                       eDown,
                       eLeft,
                       eRight,
              } ;
              enum  EReposWnd
              {
                       eNotChangePos,
                       eDesktopCenter,
                       eScreenCenter,
                       eDoNotExceedDesktop,
                       eDoNotExceedScreen,
              } ;
              union  UColor
              {
                      struct
                      {
                              BYTE r;
                              BYTE g;
                              BYTE b;
                              BYTE a;
                      } ;
```

```
                        BYTE co[4];
                        DWORD dwColor;
                    } ;
                    struct  SBorder
                    {
                        INT iSize;
                        INT iSpacing;
                        UColor sFoColor;
                        UColor sFoColor2;
                    } ;
IN              EReposWnd eReposWindow;
IN              BOOL bOnlyWorkArea;
IN OUT      RECT rtMainWindow;
IN OUT      RECT rtMainClient;
IN              SBorder boPreview;
IN              SBorder boCurScene;
IN              SBorder boCurChip;
IN              INT iBSceneCount;
IN              BOOL bAddSceneBut;
IN              BOOL bSceneName;
IN              BOOL bCurLikeBScene;
IN              FLOAT fEnterScale;
IN              BOOL bNoBigCurScene;
IN              ELocBScene eLocBScene;
IN              INT iBSceneSpacing;
IN              INT iCBSpacing;
```

```
        IN              SBorder boBScene;
        IN              INT iScrollBarSize;
        IN              INT iSBSpacing;
        IN              SBorder boScrollBar;
        IN              SBorder boScrollSlider;
        IN              BOOL bBSceneHolder;
        IN              BOOL bScrollHolder
};
```

成员

iVersion

此结构的版本，今后可能修改结构，为了保持兼容性。当前值为 0。

fScale

当前场景预览预览显示时，与原始画面的比例，值为 0 到 1 的浮点数。

当前场景的预览显示分辨率是视频分辨率与它的乘积。

如果小于等于 0，则根据设置的 rtPreview 计算预览区域的大小。

如果大于 0，则表示 fScale 是期待设置的预览比例，根据 rtPreview 和当前程序主窗口的大小计算出在预览比例改变后，需要把程序主窗口和预览显示区域设置为什么分辨率。函数返回时 fScale 被修改为实际可用的分辨率。同时被修改的，还有 rtPreview、rtMainWindow、rtMainClient。

hMainWnd

程序主界面的窗口句柄，不能为 NULL，预览画面位于该窗口中。

hPreviewWnd

绘制预览画面的子窗口句柄。可以设置为 NULL 或者与 hMainWnd 相同。

当有值时，将会把 SDK 内置的 OpenGL 的绘图窗口设置为它的子窗口，并忽略 pCbFunction 参数。

当为 NULL 时，将使用 pCbFunction 回调预览画面的图像数据，而不使用内置的 OpenGL 绘图窗口。

rtPreview

预览画面在窗口客户区的显示区域。

如果 hPreviewWnd 有值，则该区域是以 hPreviewWnd 客户区左上角为原点的区域，否则是以 hMainWnd 客户区左上角为原点的区域。

当 hPreviewWnd 和 hMainWnd 相同时，上面的表述仍然有效。

函数返回时，rtPreview 被修改为新的预览画面显示区域，应用程序可能需要参考该值重设 hPreviewWnd 窗口的大小，并按 rtMainWindow 重设主窗口的大小和位置。

siMinimumLimit

计算出的预览画面分辨率不得小于这个最小值。可以设置为 (0,0) 表示不限制最小分辨率。

pCbFunction

如果有值，且 hPreviewWnd 为 NULL，则通过该回调函数传出预览画面的数据，而不使用内置的 OpenGL 绘制窗口。

如果为 NULL，且 hPreviewWnd 也为 NULL，则不显示预览画面。

pCbParam

该参数是当调用 pCbFunction 时回传的参数。

ELocBScene

后台场景区域与当前场景的相对位置，上、下、左、右。

EReposWnd

枚举，当根据指定的预览比例计算主窗口新的大小和位置时，以哪种方式作为限制条件。

EReposWnd::eNotChangePos
计算时不要改变主窗口的坐标。

EReposWnd::eDesktopCenter
计算时，主窗口将位于桌面中心（当多显示器环境时，桌面是指所有的扩展显示器的总和）。

EReposWnd::eScreenCenter
计算时，主窗口将位于当前屏幕的中心。

EReposWnd::eDoNotExceedDesktop
计算时，只要大小改变后不超过桌面，就不要改变主窗口坐标。否则就移动位置使之在桌面范围内。

EReposWnd::eDoNotExceedScreen
计算时，只要大小改变后不超过当前屏幕，就不要改变主窗口坐标。否则就移动位置使之在当前屏幕范围内。

UColor
颜色

SBorder
设置预览画面各个区域的边框样式。

SBorder::iSize
边框线的宽度(粗细)，单位为像素。值为 0 到 32 之间的整数。

SBorder::iSpacing
显示内容与边框线之间的空白区域，单位为像素。值为 0 到 32 之间的整数。

SBorder::sFoColor

边框线的颜色。

SBorder::sFoColor2

颜色 2，当用于不同的区域时有不同的含义。
整个预览区域：背景色。
场景中的元件：鼠标进入的元件的边框色。
后台场景区域：后台场景区域中显示的当前场景的边框色。
滚动条区域：流动条的滑块的颜色。

eReposWindow

计算主窗口位置时的限制方式。

bOnlyWorkArea

计算主窗口位置和大小时，是否以桌面的工作区域（排除任务栏之后的区域）作为计算条件。
TRUE　表示主窗口大小不能超出工作区域，如果要改变窗口位置，也以工作区域为边界。
FALSE　表示不考虑工作区域，只以桌面(屏幕)大小为界限。

rtMainWindow

计算后得到的主窗口在屏幕上的区域，包含整个主窗口(标题栏、边框)。
应用程序应该根据该成员的值重新设置主窗口的大小和位置。

rtMainClient

计算后得到的主窗口的客户区域在屏幕上的区域，是使用的屏幕坐标，而不是窗口内部的坐标。

boPreview

整个预览画面的边框属性。绘制预览画面时，会按这个结构的设置来绘制边框。

boCurScene

当前场景的边框属性。绘制预览画面时，会按这个结构的设置来绘制当前场景的边框。

boCurChip

当前选中和鼠标进入的 Chip 的边框属性。

iBSceneCount

显示后台场景的数量，值为 0 到 32 的整数。例如 4，那么每个后台场景的长和宽都约为主预览画面的 1/4。

如果为 0 则表示不绘制后台场景，其它与后台场景相关的设置都会被忽略。

bAddSceneBut

在后台场景的显示区域，显示添加场景的按钮。这个按钮显示在所有后台场景的最后面。

bSceneName

后台场景上显示场景名。TRUE 表示显示，FALSE 表示不显示。

bCurLikeBScene

当前场景像后台场景那样绘制，也就是后台场景列表中也包括当前场景。

fEnterScale

当鼠标移入后台场景，后台场景缩略显示的放大比例，值为 1.0 到 iBSceneCount - 1 之间的的浮点数。

当鼠标移入的后台场景放大时，其它后台场景相应变小。鼠标移开就恢复正常。

bNoBigCurScene

不要绘制大幅的当前场景。如果设置为 TRUE，那么 bCurLikeBScene 也必须是 TRUE，并且 iBSceneCount 不能为 0。

eLocBScene

后台场景显示在当前场景的上下左右哪个位置。

iBSceneSpacing

后台场景之间的间距，单位为像素。值为 0 到 32 之间的整数。

iCBSpacing

预览时后台场景与当前场景之间的间距，单位为像素。值为 0 到 32 之间的整数。

boBScene

后台场景的边框属性。绘制预览画面时，会按这个结构的设置来绘制背景场景的边框。

iScrollBarSize

场景缩略图滚动条的大小，单位为像素。值为 0 到 32 之间的整数。

当添加的场景过多时，后台场景显示区域不能全部显示时，可能需要使用滚动条。

如果值为 0，则不使用内置的滚动条，其它与滚动条相关的参数设置会被忽略。应用程序仍可以调用相关接口切换当前显示的后台场景。

iSBSpacing

滚动条与后台场景显示区域的间距，单位为像素。值为 0 到 32 之间的整数。

boScrollBar

滚动条的边框属性。

boScrollSlider

滚动条滑块的边框属性。

bBSceneHolder

是否必须为后台场景保留显示的位置，即使当前只有一个场景。

如果为 TRUE 则始终保留后台场景区域的位置。

如果为 FALSE 则仅在需要时才给后台场景区域留下位置，当添加、删除场景时会对场景布局产生影响，可以在 fnRenderNotifyCB 回调时的 eNotify_SceneAdded, eNotify_SceneDeleted 通知中去修改主窗口的大小。

bScrollHolder

是否必须为后台场景滚动条保留显示的位置，即使当前不需要滚动条。

如果为 TRUE 则始终保留滚动条位置。

如果为 FALSE 则仅在需要时才给滚动条留下位置，当添加、删除场景时会对场景布局产生影响，可以在 fnRenderNotifyCB 回调时的 eNotify_SceneAdded, eNotify_SceneDeleted 通知中去修改主窗口的大小。

请参阅

fnRenderNotifyCB
fnPreviewImageCB
Render_SetPreviewLayout
Render_GetPreviewLayout
Render_SendPreviewMouseMessage
Render_SetBSceneScroll
Render_GetBSceneScroll

5. 场景管理

场景管理

Render_CreateScene	创建一个场景。
Render_DestroyScene	关闭并删除指定的场景。
Render_GetSceneCount	取得场景的数量。
Render_GetScene	取得指定索引值的场景句柄。
Render_SetCurScene	把指定的场景设置为当前场景。
Render_GetCurScene	取得当前场景的句柄。
Render_GetSceneIndex	取得指定场景的索引号。
Scene_SetName	设置指定场景的名字。
Scene_GetName	取得指定场景的名字。
Scene_GetChipCount	取得指定场景中的元件数量。
Scene_GetChip	使用元件索引值取得指定场景中的元件句柄。
Sceen_GetCurChip	取得指定场景中处于选中状态的元件的句柄。
Scene_CreateChip	在指定的场景中创建一个元件。

5.1 Render_CreateScene

创建一个场景。

语法

```
HSCENE WINAPI Render_CreateScene();
```

参数

无

返回值

成功返回场景句柄，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

新创建的场景中没有元件，也没有名字。可以使用 Scene_CreateChip 创建元件，和使用 Scene_SetName 设置场景的名字。

请参阅

Scene_SetName
Scene_CreateChip
Render_DestroyScene
Render_GetSceneCount
Render_GetScene
Scene_GetName
IRender_ENotify
Render_SetSize
Render_GetSize

5.2 Render_DestroyScene

关闭并删除指定的场景。

语法

```
BOOL WINAPI Render_DestroyScene(
        IN              HSCENE hScene
);
```

参数

hScene
场景的句柄，不能为 NULL。句柄是 Render_CreateScene 或 Render_GetScene 的返回值。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

关闭场景时会关闭并删除场景中的所有元件。
如果关闭的是当前场景，那么 SDK 会把另一场景设置为当前场景，并发送 eNotify_SceneSwitched 的通知。

请参阅

Render_CreateScene
Render_GetScene
Render_GetSceneCount
IRender_ENotify

5.3 Render_GetSceneCount

取得场景的数量。

语法

```
INT WINAPI Render_GetSceneCount();
```

参数

无

返回值

成功返回场景数量，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_CreateScene
Render_DestroyScene
Render_GetScene

Scene_SetName
Scene_GetName

5.4 Render_GetScene

取得指定索引值的场景句柄。

语法

```
HSCENE WINAPI Render_GetScene(
        IN              INT iIndex
);
```

参数

iIndex
场景索引值。大于等于 0，小于 Render_GetSceneCount 的返回值。

返回值

成功返回场景句柄，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_GetSceneCount
Render_CreateScene

Render_DestroyScene
Scene_SetName
Scene_GetName

5.5 Render_SetCurScene

把指定的场景设置为当前场景。

语法

```
BOOL WINAPI Render_SetCurScene(
        IN          HSCENE hScene
);
```

参数

hScene
场景的句柄，不能为 NULL。句柄是 Render_CreateScene 或 Render_GetScene 的返回值。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

当前场景是在预览画面上最大显示的场景，在进行视频编码时，只会编码当前场景的画面内容。

请参阅

Render_CreateScene
Render_GetScene
Render_GetSceneCount
IRender_ENotify

5.6 Render_GetCurScene

取得当前场景的句柄。

语法

```
HSCENE WINAPI Render_GetCurScene();
```

参数

无

返回值

成功返回场景句柄，失败返回 NULL。
如果没有任何场景存在，也返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

只要场景数量不为 0，那么总是存在当前场景。

请参阅

Render_CreateScene
Render_GetSceneCount
Render_GetScene

5.7 Render_GetSceneIndex

取得指定场景的索引号。

语法

```
INT WINAPI Render_GetSceneIndex(
        IN              HSCENE hScene
);
```

参数

hScene
场景的句柄，不能为 NULL。句柄是 Render_CreateScene 或 Render_GetScene 的返回值。

返回值

成功返回索引号，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_CreateScene
Render_GetScene
Render_GetSceneCount

5.8 Scene_SetName

设置指定场景的名字。

语法

```
BOOL WINAPI Scene_SetName(
        IN          HSCENE hScene,
        IN          LPCWSTR szName
);
```

参数

hScene
场景的句柄，不能为 NULL。句柄是 Render_CreateScene 或 Render_GetScene 的返回值。

szName
场景的名称。可以是空字符串或 NULL，表示清除场景的名称。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

场景的名字不是必须的，可以不给场景设置名字。

请参阅

Render_CreateScene
Render_GetScene
Render_GetSceneCount

5.9 Scene_GetName

取得指定场景的名字。

语法

```
LPCWSTR WINAPI Scene_GetName(
        IN          HSCENE hScene
);
```

参数

hScene
场景的句柄，不能为 NULL。句柄是 Render_CreateScene 或 Render_GetScene 的返回值。

返回值

成功返回场景的名称，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_CreateScene
Render_GetScene
Render_GetSceneCount

5.10 Scene_GetChipCount

取得指定场景中的元件数量。

语法

```
INT WINAPI Scene_GetChipCount(
        IN              HSCENE hScene
);
```

参数

hScene
场景的句柄，不能为 NULL。句柄是 Render_CreateScene 或 Render_GetScene 的返回值。

返回值

成功返回元件的数量，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_CreateScene
Render_GetScene
Scene_GetChip
Scene_CreateChip
Chip_Destroy

5.11 Scene_GetChip

使用元件索引值取得指定场景中的元件句柄。

语法

```
HCHIP WINAPI Scene_GetChip(
        IN          HSCENE hScene,
        IN          INT iIndex
);
```

参数

hScene
场景的句柄，不能为 NULL。句柄是 Render_CreateScene 或 Render_GetScene 的返回值。

iIndex

元件索引值。大于等于 0，小于 Scene_GetChipCount 的返回值。

返回值

成功返回元件句柄，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_CreateScene
Render_GetScene
Scene_GetChipCount
Scene_CreateChip
Chip_Destroy

5.12 Sceen_GetCurChip

取得指定场景中处于选中状态的元件的句柄。

语法

```
HCHIP WINAPI Sceen_GetCurChip(
        IN              HSCENE hScene
);
```

参数

hScene

场景的句柄，不能为 NULL。句柄是 Render_CreateScene 或 Render_GetScene 的返回值。

返回值

成功返回元件句柄，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_CreateScene
Render_GetScene
Chip_SetCurent
Scene_GetChipCount
Scene_GetChip
Scene_CreateChip
Chip_Destroy

5.13 Scene_CreateChip

在指定的场景中创建一个元件。

语法

```
HCHIP WINAPI Scene_CreateChip(
        IN            HSCENE hScene
);
```

参数

hScene
场景的句柄，不能为 NULL。句柄是 Render_CreateScene 或 Render_GetScene 的返回值。

返回值

返回值说明： 成功返回元件句柄，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Render_CreateScene
Render_GetScene
Scene_GetChipCount
Scene_GetChip
Chip_Destroy

6. 元件设置

元件设置

Chip_Open	为元件打开数据来源，例如屏幕截图、媒体文件等。
Chip_Close	关闭元件中打开的数据来源，但不删除元件。
Chip_Destroy	关闭并删除指定的元件。
Chip_SetIndex	设置元件的索引值（绘制顺序）。
Chip_GetIndex	取得指定元件所在的场景，以及在场景中的索引值。
Chip_SetCurent	把一个元件设置为选中状态。
Chip_SetRect	使用像素单位设置元件在画面上的位置和大小。
Chip_GetRect	取得以像素为单位的元件在画面上的位置和大小。
Chip_SetRectPercent	使用分数设置元件在画面上的位置和大小，把画布的宽高视为 1。
Chip_GetRectPercent	取得元件在画面上的位置和大小的分数值，把画布的宽高视为 1。
Chip_SetClipPercent	剪裁元件图像，被剪裁的部分不显示。
Chip_GetClipPercent	取得元件图像剪裁的设置。
Chip_SetVisible	设置元件的图像是否显示。
Chip_IsVisible	取得元件是否设置了隐藏。
Chip_SetViewLock	设置元件图像的大小、位置、角度等的锁定状态，被锁定能不能用鼠标操作。
Chip_GetViewLock	取得元件图像的大小、位置、角度等的锁定状态。
Chip_SetVolume	设置元件中播放的音视频的音量。
Chip_GetVolume	取得元件中播放音视频的音量设置。
Chip_GetClassName	取得元件中打开的数据来源所属的类别。
Chip_GetSourceName	取得元件中打开的数据来源在打开时使用的参数字符串。

Chip_GetStatus	取得元件中的数据来源（文件、流等）当前的状态。
Chip_GetStatusInfo	取得元件中打开的数据来源的详细状态信息。
Chip_GetCharacteristics	取得元件中打开的数据来源的固有特性描述结构。
IPinInput_SStatusInfo	数据来源的当前详细状态。
IPinInput_SCharacteristics	数据来源的特性，固有特性，由程序内部设置。
IPinInput_EChipStatus	加载的数据来源（文件、音视频流等）当前的状态。
IChip_EBorderFitMode	对图像大小进行缩放时，保持宽高比例的方式。
IChip_ELockType	元件图像的大小、位置、角度等的锁定状态，被锁定能不能用鼠标操作。

6.1 Chip_Open

为元件打开数据来源，例如屏幕截图、媒体文件等。

语法

```
BOOL WINAPI Chip_Open(
        IN              HCHIP hChip,
        IN              LPCWSTR szTypeName,
        IN              LPCWSTR szResource,
        IN              BOOL bCannotReuse = FALSE,
        IN              DWORD_PTR ptrParam = NULL
);
```

参数

hChip

元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

szTypeName

字符串，指定打开数据源的类型。例如：

"Camera": 摄像头。

"Screen": 屏幕区域和窗口。

"Picture": 图像文件。

"Game": 游戏进程。

还有其它类型的数据来源，以及它们的详细说明请参阅音像数据源和类型。

szResource

字符串，指定数据来源。例如：

当 szTypeName 是 "Picture" 时，szResource 就是文件路径，如 "C:\Picture\abc.gif"。

当 szTypeName 是 "Camera" 时，szResource 就是调用 Camera_GetDisplayName 获得的摄像头的 DisplayName。

bCannotReuse

是否允许源被复用。TRUE 表示不允许复用，FALSE 表示允许复用。

允许复用的情况（FALSE，默认）：例如多个元件打开同一个媒体文件，那么这个文件只会被加载一次，减少内存和 CPU 资源的消耗。

不允许复用的情况(TRUE)：例如同一个场景中的多个元件打开同一个视频文件，希望每个元件使用不同的播放进度，那么这个视频文件就只能被打开多次，不能重复利用。

ptrParam

这个参数通常使用不到，只是某些类型的源的一些特殊使用方式，可能需要用到它。例如 "Picture" 从内存指针加载位图数据，而不使用文件。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

元件打开的过程会发出 eNotify_ChipStatus 通知，元件的状态会变更为 ePin_Opened 或 ePin_Error，可能还会变更为 ePin_Played 等。

请参阅

Scene_GetChip
Scene_CreateChip
Camera_GetDisplayName
音像数据源和类型
Chip_Close
Chip_Destroy
Screen_AssembleSource
IRender_ENotify

6.2 Chip_Close

关闭元件中打开的数据来源，但不删除元件。

语法

```
BOOL WINAPI Chip_Close(
        IN              HCHIP hChip
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

关闭的过程会发出 eNotify_ChipStatus 通知，元件的状态最终会变更为 ePin_Closed。

请参阅

Scene_GetChip
Scene_CreateChip
Chip_Open

Chip_Destroy
Scene_GetChipCount
IRender_ENotify

6.3 Chip_Destroy

关闭并删除指定的元件。

语法

```
BOOL WINAPI Chip_Destroy(
        IN              HCHIP hChip
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

删除元件成功会发出 eNotify_ChipDeleted 通知。

如果删除的元件是当前选中的元件，那么还会在删除前发出 eNotify_ChipSwitched 通知，并且场景中没有任何元件被选中。

删除元件时，如果元件已经打开了数据来源，关闭的过程会发出 eNotify_ChipStatus 通知，元件的状态最终会变更为 ePin_Closed。

请参阅

Scene_GetChip
Scene_CreateChip
Chip_Open
Chip_Close
Scene_GetChipCount
IRender_ENotify

6.4 Chip_SetIndex

设置元件的索引值（绘制顺序）。

语法

```
BOOL WINAPI Chip_SetIndex(
        IN              HCHIP hChip,
        IN              INT iIndex
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

iIndex
元件索引值。大于等于 0，小于 Scene_GetChipCount 的返回值。
元件将从元件列表中移出，再插入到指定的位置。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

元件的索引值等同于元件绘制的顺序，首先绘制索引为 0 的元件，然后在它上面绘制索引为 1 的元件，以此类推直到绘制完所有元件。

设置索引值就是修改元件的绘制顺序，成功会发出 eNotify_ChipIndexChg 通知。

请参阅

Scene_GetChipCount
Scene_GetChip
Scene_CreateChip
Chip_Open

Chip_Close
Chip_Destroy
IRender_ENotify

6.5 Chip_GetIndex

取得指定元件所在的场景，以及在场景中的索引值。

语法

```
INT WINAPI Chip_GetIndex(
        IN              HCHIP hChip,
        OUT             HSCENE* pScene
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

pScene
场景句柄的指针，可以为 NULL。函数成功返回时，会把该指针指向的句柄修改为元件所在的场景句柄。
如果参数为 NULL，则忽略该参数。

返回值

成功返回元件在场景中的索引号，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

元件的索引值等同于元件绘制的顺序，首先绘制索引为 0 的元件，然后在它上面绘制索引为 1 的元件，以此类推直到绘制完所有元件。

请参阅

Scene GetChip
Scene CreateChip
Chip Open
Chip Close
Chip Destroy
Scene GetChipCount
IRender ENotify

6.6 Chip_SetCurent

把一个元件设置为选中状态。

语法

```
BOOL WINAPI Chip_SetCurent(
        IN              HCHIP hChip
);
```

参数

hChip

元件的句柄，可以为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。
当句柄为 NULL 时，取消场景中的元件选中状态，也就是没有任何元件被选中。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

一个场景只能有一个元件是选中状态，设置成功发出 eNotify_ChipSwitched 通知。
当用鼠标在预览区域点击元件时，也会造成当前元件切换。

请参阅

Scene_GetChip
Scene_CreateChip
Sceen_GetCurChip
Chip_Open
Chip_Close

Chip_Destroy
Scene_GetChipCount
IRender_ENotify

6.7 Chip_SetRect

使用像素单位设置元件在画面上的位置和大小。

语法

```
BOOL WINAPI Chip_SetRect(
        IN              HCHIP hChip,
        IN              INT iX,
        IN              INT iY,
        IN              INT iWidth,
        IN              INT iHeight,
        IN              IChip_EBorderFitMode eFitMode
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

iX
元件左上角的 X 坐标，单位为像素。

iY

元件左上角的 Y 坐标，单位为像素。

iWidth

元件的宽度，单位为像素。

iHeight

元件的高度，单位为像素。

eFitMode

当设置的大小和元件的源据来源图像宽高比例不同时，对大小进行更改的方式。

为以下值之一：

eIgnoreAspectRatio 忽略比例，不进行任何修改。

eKeepAspectRatio 保持比例，按源图像的比例缩放到设置的大小，新的大小在设置的大小的范围之内。

eKeepAspectRatioByExpanding 保持比例，按源图像的比例缩放到设置的大小，但只保证高和宽之一与设置的大小相同，新的大小会超出设置的大小的范围。

返回值

成功返回 TRUE，失败返回 FALSE。

如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Scene_GetChip

Scene_CreateChip

IChip_EBorderFitMode
Render_GetSize
Chip_GetRect
Chip_SetRectPercent
Chip_GetRectPercent
Chip_SetViewLock
IRender_ENotify

6.8 Chip_GetRect

取得以像素为单位的元件在画面上的位置和大小。

语法

```
BOOL WINAPI Chip_GetRect(
        IN            HCHIP hChip,
        OUT          PINT pX,
        OUT          PINT pY,
        OUT          PINT pWidth,
        OUT          PINT pHeight,
        IN            BOOL bRealDisplay
);
```

参数

hChip

元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

pX

返回元件左上角的 X 坐标，单位为像素。参数为 NULL 时忽略该参数。

pY

返回元件左上角的 Y 坐标，单位为像素。参数为 NULL 时忽略该参数。

pWidth

返回元件的宽度元件的宽度，单位为像素。参数为 NULL 时忽略该参数。

pHeight

返回元件的高度，单位为像素。参数为 NULL 时忽略该参数。

bRealDisplay

是否要取得真正显示的大小。TRUE 表示是，FALSE 表示否。

当设置了元件图像按比例缩放时，设置的元件大小可能并不是图像直正显示的大小，元件的显示区域的上下或左右两边会有无图像的区域，使用 TRUE 取得真正显示的大小。

返回值

成功返回 TRUE，失败返回 FALSE。

如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Scene_GetChip
Scene_CreateChip
Render_GetSize
Chip_SetRect
Chip_SetRectPercent
Chip_GetRectPercent
IRender_ENotify

6.9 Chip_SetRectPercent

使用分数设置元件在画面上的位置和大小，把画布的宽高视为 1。

语法

```
BOOL WINAPI Chip_SetRectPercent(
        IN              HCHIP hChip,
        IN              FLOAT fX,
        IN              FLOAT fY,
        IN              FLOAT fWidth,
        IN              FLOAT fHeight,
        IN              IChip_EBorderFitMode eFitMode
);
```

参数

hChip

元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

fX

元件左上角的 X 坐标，0 为画布的左边，1 为画布的右边，0.5 为中间。

fY

元件左上角的 Y 坐标，0 为画布的顶边，1 为画布的底边，0.5 为中间。

fWidth

元件的宽度，1 为画布的宽度，0.5 为画布宽度的一半。

fHeight

元件的高度，1 为画布的高度，0.5 为画布高度的一半。

eFitMode

当设置的大小和元件的源据来源图像宽高比例不同时，对大小进行更改的方式。

为以下值之一：

eIgnoreAspectRatio 忽略比例，不进行任何修改。

eKeepAspectRatio 保持比例，按源图像的比例缩放到设置的大小，新的大小在设置的大小的范围之内。

eKeepAspectRatioByExpanding 保持比例，按源图像的比例缩放到设置的大小，但只保证高和宽之一与设置的大小相同，新的大小会超出设置的大小的范围。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

把 Render_SetSize 设置的宽高进行归一计算，从逻辑上来说，画布左上角坐标为 (0,0)，右下角坐标为 (1,1)，设置的元件图像坐标和宽高，就使用小数。

请参阅

Render_SetSize
Scene_GetChip
Scene_CreateChip
IChip_EBorderFitMode
Render_GetSize
Chip_SetRect
Chip_GetRect
Chip_GetRectPercent
Chip_SetViewLock
IRender_ENotify

6.10 Chip_GetRectPercent

取得元件在画面上的位置和大小的分数值，把画布的宽高视为 1。

语法

```
BOOL WINAPI Chip_GetRectPercent(
        IN            HCHIP hChip,
        OUT          PFLOAT pX,
        OUT          PFLOAT pY,
        OUT          PFLOAT pWidth,
        OUT          PFLOAT pHeight,
        IN            BOOL bRealDisplay
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

pX
返回元件左上角的 X 坐标。参数为 NULL 时忽略该参数。

pY
返回元件左上角的 Y 坐标。参数为 NULL 时忽略该参数。

pWidth
返回元件的宽度元件的宽度。参数为 NULL 时忽略该参数。

pHeight
返回元件的高度。参数为 NULL 时忽略该参数。

bRealDisplay

是否要取得真正显示的大小。TRUE 表示是，FALSE 表示否。

当设置了元件图像按比例缩放时，设置的元件大小可能并不是图像直正显示的大小，元件的显示区域的上下或左右两边会有无图像的区域，使用 TRUE 取得真正显示的大小。

返回值

成功返回 TRUE，失败返回 FALSE。

如果失败，可以使用 GetLastError() 取得错误代码。

备注

把 Render_SetSize 设置的宽高进行归一计算，从逻辑上来说，画布左上角坐标为 (0,0)，右下角坐标为 (1,1)，元件图像坐标和宽高，就是小数。

请参阅

Render_SetSize
Scene_GetChip
Scene_CreateChip
Render_GetSize
Chip_SetRect
Chip_GetRect
Chip_SetRectPercent
IRender_ENotify

6.11 Chip_SetClipPercent

剪裁元件图像，被剪裁的部分不显示。

语法

```
BOOL WINAPI Chip_SetClipPercent(
        IN              HCHIP hChip,
        IN              FLOAT fLeft,
        IN              FLOAT fTop,
        IN              FLOAT fRight,
        IN              FLOAT fBottom
);
```

参数

hChip

元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

fLeft

左边剪掉的部分，图像宽度视为 1，如左边剪掉 20% 那么值就是 0.2。

fTop

顶边剪掉的部分，图像高度视为 1，如顶边剪掉 20% 那么值就是 0.2。

fRight
右边剪掉的部分，图像宽度视为 1，如右边剪掉 20% 那么值就是 0.2。

fBottom
底边剪掉的部分，图像高度视为 1，如底边剪掉 20% 那么值就是 0.2。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

把元件图像的大小进行归一计算，从逻辑上来说，元件图像左上角坐标为 (0,0)，右下角坐标为 (1,1)。

请参阅

Scene GetChip
Scene CreateChip
Chip GetClipPercent
Chip GetRect
Chip GetRectPercent
Chip SetViewLock
IRender ENotify

6.12 Chip_GetClipPercent

取得元件图像剪裁的设置。

语法

```
BOOL WINAPI Chip_GetClipPercent(
        IN            HCHIP hChip,
        OUT          PFLOAT pLeft,
        OUT          PFLOAT pTop,
        OUT          PFLOAT pRight,
        OUT          PFLOAT pBottom
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene CreateChip 或 Scene GetChip 的返回值。

pLeft
取得左边剪掉的设置值。参数为 NULL 则忽略该参数。

pTop
取得顶边剪掉的设置值。参数为 NULL 则忽略该参数。

pRight
取得右边剪掉的设置值。参数为 NULL 则忽略该参数。

pBottom
取得底边剪掉的设置值。参数为 NULL 则忽略该参数。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

把元件图像的大小进行归一计算，从逻辑上来说，元件图像左上角坐标为 (0,0)，右下角坐标为 (1,1)。

请参阅

Scene_GetChip
Scene_CreateChip
Chip_SetClipPercent
Chip_GetRect
Chip_GetRectPercent
IRender_ENotify

6.13 Chip_SetVisible

设置元件的图像是否显示。

语法

```
BOOL WINAPI Chip_SetVisible(
        IN              HCHIP hChip,
        IN              BOOL bVisible
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

bVisible
TRUE 表示显示，FALSE 表示隐藏。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Scene_GetChip
Scene_CreateChip
Chip_IsVisible
IRender_ENotify

6.14 Chip_IsVisible

取得元件是否设置了隐藏。

语法

```
BOOL WINAPI Chip_IsVisible(
        IN              HCHIP hChip
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

返回值

TRUE 表示显示，FALSE 表示隐藏。

请参阅

Scene_GetChip
Scene_CreateChip
Chip_SetVisible
IRender_ENotify

6.15 Chip_SetViewLock

设置元件图像的大小、位置、角度等的锁定状态，被锁定能不能用鼠标操作。

语法

```
BOOL WINAPI Chip_SetViewLock(
        IN              HCHIP hChip,
        IN              IChip_ELockType eLockType,
        IN              BOOL bLock
);
```

参数

hChip

元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

eLockType

锁定的类型，为以下值之一：

eLock_AspectRatio	宽高比例，锁定后缩放时保持锁定。
eLock_Position	坐标，锁定后不能用鼠标移动。
eLock_Size	大小，锁定后不能用鼠标缩放
eLock_Angle	旋转角度，锁定后不能用鼠标旋转。

bLock

是否锁定，TRUE 表示设置为锁定，FALSE 表示不锁定。

返回值

成功返回 TRUE，失败返回 FALSE。

如果失败，可以使用 GetLastError() 取得错误代码。

备注

设置锁定状态，会发出 eNotify_ChipViewLock 通知。
元件的锁定状态不影响调用函数进行大小和位置的修改，只有锁定宽高比时即使调用函数设置大小仍然会保持比例。

请参阅

Scene_GetChip
Scene_CreateChip
IChip_ELockType
Chip_GetViewLock
IChip_EBorderFitMode
Chip_GetRect
Chip_GetRectPercent
IRender_ENotify

6.16 Chip_GetViewLock

取得元件图像的大小、位置、角度等的锁定状态。

语法

```
BOOL WINAPI Chip_GetViewLock(
        IN              HCHIP hChip,
        IN              IChip_ELockType eLockType
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

eLockType
锁定的类型，为以下值之一：

eLock_AspectRatio	宽高比例，锁定后缩放时保持锁定。
eLock_Position	坐标，锁定后不能用鼠标移动。
eLock_Size	大小，锁定后不能用鼠标缩放
eLock_Angle	旋转角度，锁定后不能用鼠标旋转。

返回值

锁定返回 TRUE，未锁定返回 FALSE。

请参阅

Scene_GetChip
Scene_CreateChip
IChip_ELockType
Chip_SetViewLock
IChip_EBorderFitMode
Chip_GetRect
Chip_GetRectPercent
IRender_ENotify

6.17 Chip_SetVolume

设置元件中播放的音视频的音量。

语法

```
BOOL WINAPI Chip_SetVolume(
        IN              HCHIP hChip,
        IN              FLOAT fVolume,
        IN              BOOL bMute
);
```

参数

hChip

元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

fVolume

音量，有效值为 0 到 1 之间的浮点数。

bMute

是否设置为静音，TRUE 表示静音，FALSE 取消静音。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

只有在播放音视频文件或在线流时，音量设置才有效。

请参阅

Scene_GetChip
Scene_CreateChip
Chip_GetVolume
Chip_Open
Chip_Close
Chip_Destroy

6.18 Chip_GetVolume

取得元件中播放音视频的音量设置。

语法

```
FLOAT WINAPI Chip_GetVolume(
        IN              HCHIP hChip,
        OUT             PBOOL pMute
);
```

参数

hChip

元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

pMute

返回是否是静音状态，TRUE 表示静音，FALSE 表示没有静音。
如果参数为 NULL，则忽略该参数。

返回值

反正音量设置，值为 0 到 1 之间的浮点数。

请参阅

Scene_GetChip
Scene_CreateChip
Chip_SetVolume
Chip_Open
Chip_Close
Chip_Destroy

6.19 Chip_GetClassName

取得元件中打开的数据来源所属的类别。

语法

```
LPCWSTR WINAPI Chip_GetClassName(
        IN            HCHIP hChip
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

返回值

字符串，如 "Picture"、"Camera" 等类型名。

请参阅

Scene_GetChip
Scene_CreateChip
音像数据源和类型
Chip_Open
Render_GetClassCount
Render_GetClassInfo
IGlRender_SClassInfo

6.20 Chip_GetSourceName

取得元件中打开的数据来源在打开时使用的参数字符串。

语法

```
LPCWSTR WINAPI Chip_GetSourceName(
        IN            HCHIP hChip
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

返回值

字符串，返回在调用 Chip_Open 时传入的 szResource 参数。

请参阅

Scene_GetChip
Scene_CreateChip
Chip_Open
音像数据源和类型
Render_GetClassInfo
IGlRender_SClassInfo
Screen_AnalysisSource

6.21 Chip_GetStatus

取得元件中的数据来源（文件、流等）当前的状态。

语法

```
IPinInput_EChipStatus WINAPI Chip_GetStatus(
        IN              HCHIP hChip
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

返回值

状态，请参阅 IPinInput_EChipStatus 的说明。

请参阅

Scene_GetChip
Scene_CreateChip
IPinInput_EChipStatus
Chip_GetStatusInfo
Chip_GetCharacteristics
IPinInput_SStatusInfo
Chip_Open
Chip_Close

6.22 Chip_GetStatusInfo

取得元件中打开的数据来源的详细状态信息。

语法

```
BOOL WINAPI Chip_GetStatusInfo(
        IN              HCHIP hChip,
        OUT             IPinInput_SStatusInfo* pStatus
);
```

参数

hChip

元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

pStatus

返回时填充详细的状态信息到指针指向的结构。参数不能为 NULL。
请参阅 IPinInput_SStatusInfo 的详细说明。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Scene_GetChip
Scene_CreateChip
IPinInput_SStatusInfo
Chip_GetStatus
Chip_GetCharacteristics
Chip_Open
Chip_Close

6.23 Chip_GetCharacteristics

取得元件中打开的数据来源的固有特性描述结构。

语法

```
BOOL WINAPI Chip_GetCharacteristics(
        IN              HCHIP hChip,
        OUT             IPinInput_SCharacteristics* pCharacter
);
```

参数

hChip
元件的句柄，不能为 NULL。句柄是 Scene_CreateChip 或 Scene_GetChip 的返回值。

pCharacter

返回时填充信息到指针指向的结构。参数不能为 NULL。
请参阅 IPinInput_SCharacteristics 的详细说明。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

数据来源的固有特性，主要是来源是否有图像、是否有动画、是否有声音、是否可以控制播放进度等特性信息。

请参阅

Scene_GetChip
Scene_CreateChip
IPinInput_SCharacteristics
Chip_GetStatus
Chip_GetStatusInfo
IPinInput_SStatusInfo
Chip_Open
Chip_Close

6.24 IPinInput_SStatusInfo

数据来源的当前详细状态。

语法

```
struct IPinInput_SStatusInfo
{
        IPinInput_EChipStatus eStatus;
        bool bCannotReuse;
        bool bIsMute;
        bool bIsMixer;
        bool bIsInteract;
        bool bIsLoop;
        bool bIsRange;
        bool bRangIsFrame;
        INT64 iFrameCount;
        FLOAT fFrameRate;
        FLOAT fSecondCount;
        FLOAT fVolume;
        INT64 iFramePlay;
        FLOAT fSecondPlay;
        INT64 iFrameStart;
        INT64 iFrameEnd;
        FLOAT fSecondStart;
        FLOAT fSecondEnd
};
```

成员

eStatus
当前状态的枚举值。

bCannotReuse
由用户设置的是否允许复用，true 表示不允许，false 表示允许。

bIsMute
是否是静音状态。

bIsMixer
是否是软件混音。

bIsInteract
是否已经允许用户交互。

bIsLoop
是否是循环播放状态，即播放完后又从头开始。

bIsRange
是否是仅播放指定时间范围。

bRangIsFrame
指定的播放范围是否是按帧计算，false 表示按秒计算，true 表示按帧计算

iFrameCount
总的帧数。

fFrameRate
帧率。

fSecondCount
总的秒数。

fVolume
当前音量，值为 0 到 1。

iFramePlay
当前正在播放的帧。

fSecondPlay
当前正在播放的时间。

iFrameStart
设置的播放区域的开始帧。

iFrameEnd
设置的播放区域的结束帧。

fSecondStart
设置的播放区域的开始时间。

fSecondEnd

设置的播放区域的结束时间。

请参阅

IPinInput_EChipStatus
Chip_GetStatus
Chip_GetStatusInfo
Chip_GetCharacteristics
Chip_Open
Chip_Close

6.25 IPinInput_SCharacteristics

数据来源的特性，固有特性，由程序内部设置。

语法

```
struct IPinInput_SCharacteristics
{
        bool bExistImage;
        bool bExistMovie;
        bool bExistSound;
        bool bTunableVolume;
        bool bTunableMute;
        bool bTunableMixer;
```

```
        bool bTunablePause;
        bool bTunableStop;
        bool bTunableSeek;
        bool bTunableReuse;
        bool bExclusiveMode;
        bool bTunableInteract;
        bool bDefaultLockRatio;
        bool bTurnUpDown;
        bool bTurnLeftRight
};
```

成员

bExistImage
　　是否存在静止图像。

bExistMovie
　　是否存在运动图像。

bExistSound
　　是否存在声音。

bTunableVolume
　　是否允许控制音量。

bTunableMute
　　是否允许静音。

bTunableMixer
是否允许软件混音。

bTunablePause
是否允许暂停。

bTunableStop
是否允许停止。

bTunableSeek
是否允许重置播放位置。

bTunableReuse
是否允许复用，即多个输入使用同一个源（例如在线视频，只需一次解码）。

bExclusiveMode
源是否只能是独占模式，例如截取游戏画面，只允许打开一次，不能多次打开，但可以复用。

bTunableInteract
是否允许与用户交互，例如 Flash 动画。

bDefaultLockRatio
源是否默认锁定长宽比。

bTurnUpDown
源图像本身上下翻转，显示时需要上下翻转。

bTurnLeftRight

源本身左右翻转，显示时需要左右翻转。

请参阅

Chip_GetStatus
Chip_GetStatusInfo
Chip_GetCharacteristics
IPinInput_SStatusInfo
Chip_Open
Chip_Close

6.26 IPinInput_EChipStatus

加载的数据来源（文件、音视频流等）当前的状态。

语法

```
enum IPinInput_EChipStatus
{
        ePin_None,
        ePin_Error,
        ePin_Closed,
        ePin_Opened,
        ePin_Stoped,
        ePin_Paused,
```

```
        ePin_Played,
        ePin_Closing,
        ePin_Opening,
        ePin_Stoping,
        ePin_Pausing,
        ePin_Loading
};
```

成员

ePin_None
　　没有设置源，也就是还没有调用　Chip_Open　函数。

ePin_Error
　　源错误，打开失败或播放过程中出错。

ePin_Closed
　　已经关闭。

ePin_Opened
　　已经打开完成。

ePin_Stoped
　　已经停止播放。

ePin_Paused
　　已经暂停播放。

ePin_Played
正在播放。

ePin_Closing
正在关闭，关闭完成后状态会变更为 ePin_Closed。

ePin_Opening
正在打开，打开完成后状态会变更为 ePin_Opened。

ePin_Stoping
正在停止，停止完成后状态会变更为 ePin_Stoped。

ePin_Pausing
正在暂停，暂停完成后状态会变更为 ePin_Paused。

ePin_Loading
正在加载，或者正在重设播放进度。操作完成后，如果之前是 ePin_Played 状态，就继续播放，是 ePin_Paused 状态就继续暂停。

请参阅

Chip_Open
Chip_GetStatus
Chip_GetStatusInfo
Chip_GetCharacteristics
IPinInput_SStatusInfo
Chip_Close

6.27 IChip_EBorderFitMode

对图像大小进行缩放时，保持宽高比例的方式。

语法

```
enum IChip_EBorderFitMode
{
        eIgnoreAspectRatio,
        eKeepAspectRatio,
        eKeepAspectRatioByExpanding
};
```

成员

eIgnoreAspectRatio
忽略比例，直接缩放到指定的大小。

eKeepAspectRatio
保持比例，按源图像的比例缩放到设置的大小，新的大小在设置的大小的范围之内。

eKeepAspectRatioByExpanding
保持比例，按源图像的比例缩放到设置的大小，但只保证高和宽之一与设置的大小相同，新的大小会超出设置的大小的范围。

请参阅

6.28 IChip_ELockType

元件图像的大小、位置、角度等的锁定状态，被锁定能不能用鼠标操作。

语法

```
enum IChip_ELockType
{
        eLock_AspectRatio,
        eLock_Position,
        eLock_Size,
        eLock_Angle
};
```

成员

eLock_AspectRatio
宽高比例，锁定后缩放时保持锁定。

eLock_Position
坐标，锁定后不能用鼠标移动。

eLock_Size
大小，锁定后不能用鼠标缩放

eLock_Angle

旋转角度，锁定后不能用鼠标旋转。

备注

元件的锁定状态不影响调用函数进行大小和位置的修改，只有锁定宽高比时即使调用函数设置大小仍然会保持比例。

请参阅

7. 音像数据源和类型

使用本 SDK 可以把多种类型的图像整合在同一个画面中，这些图像可以来自于图像文件、视频文件、摄像头、实时桌面截图、实时游戏截图、在线视频流等。

如果按照常见的开发逻辑，可能打开不同类型的图像来源时要调用不同的接口，比如打开摄像头的接口，比如设置桌面截图的接口，比如打开在线视频的接口等，这样很麻烦，对于 SDK 的使用者很麻烦，对于 SDK 的封装也很麻烦。因此，为了简化对 SDK 的使用，也为了 SDK 接口封装的统一性和可扩展性，打开任何类型的图像来源，都使用同一个接口 Chip_Open 。

从 Chip_Open 函数的说明中可以看出，要确定一个图像源，需要两个参数。一是来源类型，二是对源进行描述的字符串。下面的表格对这两个参数进行详细的说明。

源类型	类型参数字符串	描述源的字符串格式

图像文件 支持 bmp,jpg,png,gif(含 gif 动画)	Picture	1. 文件的完整路径。例如 "C:\image\abc.png"。 例如：`Chip_Open( hChip, L"Picture", L"C:\image\abc.png", NULL );` 2. "Bitmap"，这表示将传入位图数据，而不是图像文件。此时 Chip_Open 的 ptrParam 参数有效，直接图像的数据指针作为 ptrParam 参数传入，只支持 32 位与 GDI 兼容的 ARGB 格式。 例如：`Chip_Open( hChip, L"Picture", L"Bitmap", (DWORD_PTR)pBitmapBuffer );`
摄像头 支持各种 USB 摄像头、和 WDM 标准的视频采集卡。	Camera	1. 摄像头的 DisplayName 字符串，使用 Camera_GetDisplayName 函数获得。使用默认分辨率打开摄像头。 例如：`Chip_Open( hChip, L"Camera", L"DisplayName", NULL );` 2. 指定打开摄像头的分辨率，字符串格式为 "DisplayName,Width,Height"。使用 "," 连接 DisplayName 和宽度高度，Width 和 Height 请使用大于 0 的正整数。 例如：`Chip_Open( hChip, L"Camera", L"DisplayName,640,480", NULL );`
屏幕和窗口录制 支持多显示器，各种分辨率和色深。	Screen	参数格式：ScreenIndex|WindowHWND|WindowTitle[,0|1[,Left,Top,Width,Height]]。 1. 指定屏幕编号，录制屏幕。0 到 255 指定屏幕编号，-1 表示所有屏幕(多显示器时录制所有显示器)。 例如：`Chip_Open( hChip, L"Screeen", L"0", NULL );` 2. 指定窗口句柄，录制指定的窗口。凡是值大于 255 的数，都认为是窗口句柄，不区分是 10 进制还是 16 进制。 例如：`Chip_Open( hChip, L"Screeen", L"0x0034ABCD", NULL );` 3. 指定窗口标题字符串，录制指定的窗口。字符串需要用 "[" 和 "]" 括起来。 例如：`Chip_Open( hChip, L"Screeen", L"[计算器]", NULL );` 4. 指定录制屏幕工作区、窗口的客户区。这就需要用 "," 连接第 2 个参数了，0 表示指定目标的所有区域，1 表示仅客户区（工作区）。 例如：`Chip_Open( hChip, L"Screeen", L"0,1", NULL );` 5. 录制指定目标的指定区域。例如只录制屏幕上从 (220, 300) 开始宽度为 800*600 的区域。 例如：`Chip_Open( hChip, L"Screeen", L"0,0,220,300,800,600", NULL );`

		指定录制区域的坐标，以目标的左上角为原点，例如目标是窗口，指定的坐标就是从窗口左上角开始的偏移。 第二个参数仍表示是否仅录制客户区（工作区），当为 1 时，就是以窗口的客户区或桌面的工作区左上角为原点， 屏幕和窗口录制的参数相对复杂一些，因此 SDK 中提供了两个函数： Screen_AssembleSource 可以把结构 IScreen_SCapParams 转换为 Chip_Open 使用的字符串； Screen_AnalysisSource 把字符串转换为 IScreen_SCapParams 结构。

与音像数据源相关函数、结构和枚举：

Render_GetClassCount	取得 SDK 支持的音像数据源（文件、摄像头等）类型数量。
Render_GetClassInfo	取得对音像数据源类型的说明。
Camera_GetCount	取得摄像头的数量。
Camera_GetFriendlyName	取得指定的摄像头的友好名称，也就是显示在界面上给用户看的名称。
Camera_GetDisplayName	取得指定的摄像头的内部名称，内部名称可以作为摄像头的唯一标识。
Camera_GetIndex	根据摄像头内部名称得到摄像的索引值。
Screen_GetCount	取得当前系统中显示器的数量。
Screen_GetInfo	取得指定的显示器的分辨率等相关信息。
Screen_AssembleSource	根据 IScreen_SCapParams 结构生成调用 Chip_Open 的参数。
Screen_AnalysisSource	把 Chip_GetSourceName 返回的截屏参数字符串分解到 IScreen_SCapParams 结构。
IScreen_SCapParams	屏幕、窗口截取的参数结构。
IGlRender_SClassInfo	SDK 所支持的音像数据来源的类型说明结构。

7.1 Render_GetClassCount

取得 SDK 支持的音像数据源（文件、摄像头等）类型数量。

语法

```
INT WINAPI Render_GetClassCount();
```

参数

无

返回值

成功返回数量，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

音像数据源和类型
Render_GetClassInfo
IGlRender_SClassInfo
Chip_Open

7.2 Render_GetClassInfo

取得对音像数据源类型的说明。

语法

```
IGlRender_SResourceInfo WINAPI Render_GetClassInfo(
        INT iIndex
);
```

参数

iIndex
类型的索引值，大于等于 0，小于 Render_GetClassCount 的返回值。

返回值

返回对源类型的描述结构。
详细信息请参阅 IGlRender_SClassInfo 和 音像数据源和类型。

请参阅

Render_GetClassCount
IGlRender_SClassInfo
音像数据源和类型
Chip_Open

7.3 Camera_GetCount

取得摄像头的数量。

语法

```
INT WINAPI Camera_GetCount();
```

参数

无

返回值

成功返回摄像头数量，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Camera_GetFriendlyName
Camera_GetDisplayName
Camera_GetIndex

7.4 Camera_GetDisplayName

取得指定的摄像头的内部名称，内部名称可以作为摄像头的唯一标识。

语法

```
LPCWSTR WINAPI Camera_GetDisplayName(
        IN              INT iIndex
);
```

参数

iIndex
大于等于 0，小于 Camera_GetCount 返回值的摄像头索引值。

返回值

成功返回内部名称字符串，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Camera_GetCount
Camera_GetFriendlyName
Camera_GetIndex
Chip_Open

7.5 Camera_GetFriendlyName

取得指定的摄像头的友好名称，也就是显示在界面上给用户看的名称。

语法

```
LPCWSTR WINAPI Camera_GetFriendlyName(
        IN          INT iIndex
);
```

参数

iIndex
大于等于 0，小于 Camera_GetCount 返回值的摄像头索引值。

返回值

成功返回友好名称字符串，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Camera_GetCount
Camera_GetDisplayName
Camera_GetIndex

7.6 Camera_GetIndex

根据摄像头内部名称得到摄像的索引值。

语法

```
INT WINAPI Camera_GetIndex(
        IN              LPCWSTR szDisplayName
);
```

参数

szDisplayName
使用 Camera_GetDisplayName 得到的内部名称字符串。

返回值

成功返回摄像头索引值，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Camera_GetDisplayName
Camera_GetCount
Camera_GetFriendlyName

7.7 Screen_GetCount

取得当前系统中显示器的数量。

语法

```
INT WINAPI Screen_GetCount();
```

参数

无

返回值

成功返回显示器数量，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

如果显示器是设置为复制显示，那么不会计算到数量中。

请参阅

Screen_GetInfo
IScreen_SCapParams

7.8 Screen_GetInfo

取得指定的显示器的分辨率等相关信息。

语法

```
BOOL WINAPI Screen_GetInfo(
        IN              INT iIndex,
```

```
        OUT           MONITORINFOEXW* pMonitorInfo
);
```

参数

iIndex
大于等于 0，小于 Screen_GetCount 返回值的显示器索引值。

pMonitorInfo

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Screen_GetCount
IScreen_SCapParams
Screen_AssembleSource

7.9 Screen_AssembleSource

根据 IScreen_SCapParams 结构生成调用 Chip_Open 的参数。

语法

```
LPCWSTR WINAPI Screen_AssembleSource(
        IN          IScreen_SCapParams* pParams
);
```

参数

pParams
IScreen_SCapParams 结构的指针，不能为 NULL。

返回值

成功返回参数字符串，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Chip_Open
IScreen_SCapParams
Screen_GetCount
Screen_GetInfo

7.10 Screen_AnalysisSource

把 Chip_GetSourceName 返回的截屏参数字符串分解到 IScreen_SCapParams 结构。

语法

```
BOOL WINAPI Screen_AnalysisSource(
        IN            LPCWSTR szSource,
        OUT           IScreen_SCapParams* pParams
);
```

参数

szSource
调用 Chip_Open 时传入的 szResource 参数，或 Chip_GetSourceName 的返回值。仅限于 "Screen" 类型的打开参数。

pParams
分解的参数会填充到指针指向的结构，参数不能为 NULL。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Chip_Open
Chip_GetSourceName
IScreen_SCapParams
Screen_GetCount
Screen_GetInfo
Screen_AssembleSource

7.11 IScreen_SCapParams

屏幕、窗口截取的参数结构。

语法

```
struct IScreen_SCapParams
{
        WCHAR szWindow[MAX_PATH];
        HWND hWindow;
        INT iScreen;
        BOOL bOnlyClient;
        BOOL bUseInitRect;
        RECT rtInit
};
```

成员

szWindow

窗口标题，可以设置为空字符串。

hWindow

窗口句柄，如果窗口标题不为空，则忽略该值。

iScreen

屏幕编号(-1 表示所有屏幕，其它从 0 开始)。如果窗口标题或窗口句柄有值，就忽略屏幕编号。

bOnlyClient
是否只截取窗口的客户区域，对于屏幕则是指工作区。

bUseInitRect
是否使用 rtInit 中指定的区域作为截图区域，不使用则是截取整个的窗口、屏幕等区域。

rtInit
指定截图区域，使用以窗口(或客户区)、屏幕（或工作区）左上角为原点的相对坐标。

备注

需要把结构用 Screen_AssembleSource 转换为 Chip_Open 可以使用的字符串参数，才能调用 Chip_Open 开启屏幕截取。

请参阅

Chip_Open
Screen_AssembleSource
Screen_AnalysisSource
Chip_GetSourceName

7.12 IGlRender_SClassInfo

SDK 所支持的音像数据来源的类型说明结构。

语法

```
struct IGlRender_SClassInfo
{
        LPCWSTR szClassName;
        LPCWSTR szDescription;
        LPCWSTR szRemark
};
```

成员

szClassName
类型名称，例如 "Camera"、"Screen"

szDescription
类型的基本说明，描述它是什么。

szRemark
类型的详细注释。

请参阅

音像数据源和类型
Render_GetClassInfo
Render_GetClassCount
Chip_Open

8. 录音控制

录音控制

Audio_GetDevCount	取得可以用于录音的指定类型的音频设备的数量。
Audio_GetDevName	取得指定的录音设备的名称，用于显示在界面上。
Audio_GetDevId	取得指定的录音设备的 ID 字符串，用于区分不同的设备。
Audio_SetCurrent	设置指定类型的当前选择的录音设备。
Audio_GetCurrent	取得指定类型的当前选择的录音设备 ID 字符串。
Audio_SetVolume	设置指定类型录音设备的录音音量。
Audio_GetVolume	取得指定类型录音设备的录音音量。
Audio_Enable	启用指定类型录音设备的录音。
Audio_IsEnabled	取得是否启用了指定类型录音设备的录音。
EAudioCaptureDev	可以用于录音的音频设备类型

8.1 Audio_GetDevCount

取得可以用于录音的指定类型的音频设备的数量。

语法

```
INT WINAPI Audio_GetDevCount(
        IN          EAudioCaptureDev eDev
);
```

参数

eDev

录音设备类型，为下列值之一：

eAudCap_Speaker　　扬声器，泛指音频输出设备。

eAudCap_Microphone　　麦克风，泛指音频输入设备。

返回值

成功返回设备数量，失败返回 -1。

如果失败，可以使用 GetLastError() 取得错误代码。

备注

系统中的可以用于录音的音频设备，在本 SDK 中抽象为“输入设备”和“输出设备”两类，无论是 Windows XP 还是在 Windows 7/8/10 上，应用程序都使用相同的接口进行录音的控制。

请参阅

EAudioCaptureDev
Audio_GetDevName
Audio_GetDevId
Audio_SetCurrent

8.2 Audio_GetDevName

取得指定的录音设备的名称，用于显示在界面上。

语法

```
LPCWSTR WINAPI Audio_GetDevName(
        IN          EAudioCaptureDev eDev,
        IN          UINT iIndex
);
```

参数

eDev

录音设备类型，为下列值之一：

eAudCap_Speaker　　扬声器，泛指音频输出设备。

eAudCap_Microphone　　麦克风，泛指音频输入设备。

iIndex

大于等于 0，小于 Audio_GetDevCount 返回值的索引值。

返回值

成功返回名称字符串，失败返回 NULL。

如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Audio_GetDevCount
EAudioCaptureDev
Audio_GetDevId
Audio_SetCurrent

8.3 Audio_GetDevId

取得指定的录音设备的 ID 字符串，用于区分不同的设备。

语法

```
LPCWSTR WINAPI Audio_GetDevId(
        IN          EAudioCaptureDev eDev,
        IN          UINT iIndex
);
```

参数

eDev

录音设备类型，为下列值之一：

eAudCap_Speaker　　扬声器，泛指音频输出设备。

eAudCap_Microphone　　麦克风，泛指音频输入设备。

iIndex

大于等于 0，小于 Audio_GetDevCount 返回值的索引值。

返回值

成功返回 ID 字符串，失败返回 NULL。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Audio_GetDevCount
EAudioCaptureDev
Audio_GetDevName
Audio_SetCurrent
Audio_GetCurrent

8.4 Audio_SetCurrent

设置指定类型的当前选择的录音设备。

语法

```
BOOL WINAPI Audio_SetCurrent(
        IN          EAudioCaptureDev eDev,
        IN          LPCWSTR szDevId
);
```

参数

eDev

录音设备类型，为下列值之一：

eAudCap_Speaker　　扬声器，泛指音频输出设备。

eAudCap_Microphone　　麦克风，泛指音频输入设备。

szDevId

Audio_GetDevId　返回的设备　ID　字符串。

返回值

成功返回　TRUE，失败返回　FALSE。

如果失败，可以使用　GetLastError()　取得错误代码。

请参阅

Audio_GetDevId

EAudioCaptureDev

Audio_GetDevCount

Audio_GetDevName

Audio_GetCurrent

8.5 Audio_GetCurrent

取得指定类型的当前选择的录音设备　ID　字符串。

语法

```
LPCWSTR WINAPI Audio_GetCurrent(
        IN          EAudioCaptureDev eDev
);
```

参数

eDev

录音设备类型，为下列值之一：

eAudCap_Speaker　　扬声器，泛指音频输出设备。

eAudCap_Microphone　　麦克风，泛指音频输入设备。

返回值

成功当前选择的录音设备 ID 字符串，失败返回 NULL。

如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

EAudioCaptureDev

Audio_GetDevCount

Audio_GetDevName

Audio_GetDevId

Audio_SetCurrent

8.6 Audio_SetVolume

设置指定类型录音设备的录音音量。

语法

```
BOOL WINAPI Audio_SetVolume(
        IN              EAudioCaptureDev eDev,
        IN              FLOAT fVolume
);
```

参数

eDev

录音设备类型，为下列值之一：

eAudCap_Speaker　　扬声器，泛指音频输出设备。

eAudCap_Microphone　　麦克风，泛指音频输入设备。

fVolume

录音音量设置，0　到　1　之间的浮点数。

返回值

成功返回　TRUE，失败返回　FALSE。

如果失败，可以使用　GetLastError()　取得错误代码。

备注

“输入”和“输出”两种类型的录音设备，可以分别设置不同的录音音量。SDK 内部会分别录音后再根据各自的音量设置进行软件混音。

请参阅

EAudioCaptureDev
Audio_GetVolume
Audio_Enable
Audio_GetDevName
Audio_GetDevId
Audio_SetCurrent
Audio_GetCurrent

8.7 Audio_GetVolume

取得指定类型录音设备的录音音量。

语法

```
FLOAT WINAPI Audio_GetVolume(
        IN          EAudioCaptureDev eDev
);
```

参数

eDev

录音设备类型，为下列值之一：

eAudCap_Speaker　　扬声器，泛指音频输出设备。

eAudCap_Microphone　　麦克风，泛指音频输入设备。

返回值

录音音量设置，0　到　1　之间的浮点数。

请参阅

EAudioCaptureDev
Audio_Enable
Audio_GetDevName
Audio_GetDevId
Audio_SetCurrent
Audio_GetCurrent

8.8 Audio_Enable

启用指定类型录音设备的录音。

语法

```
BOOL WINAPI Audio_Enable(
        IN          EAudioCaptureDev eDev,
```

```
        IN              BOOL bIsEnable
);
```

参数

eDev

录音设备类型，为下列值之一：

eAudCap_Speaker　　扬声器，泛指音频输出设备。

eAudCap_Microphone　　麦克风，泛指音频输入设备。

bIsEnable

TRUE　表示启动，FALSE　表示禁用。

返回值

成功返回　TRUE，失败返回　FALSE。

如果失败，可以使用　GetLastError()　取得错误代码。

请参阅

EAudioCaptureDev

Audio_GetDevCount

Audio_GetDevName

Audio_GetDevId

Audio_SetCurrent

Audio_GetCurrent

Audio_SetVolume

8.9 Audio_IsEnabled

取得是否启用了指定类型录音设备的录音。

语法

```
BOOL WINAPI Audio_IsEnabled(
        IN          EAudioCaptureDev eDev
);
```

参数

eDev

录音设备类型，为下列值之一：

eAudCap_Speaker　　扬声器，泛指音频输出设备。

eAudCap_Microphone　　麦克风，泛指音频输入设备。

返回值

启用了录音返回　TRUE，没有启用返回　FALSE。

请参阅

EAudioCaptureDev

Audio_GetDevCount

Audio_GetDevName

Audio_GetDevId

Audio_SetCurrent
Audio_GetCurrent

8.10 EAudioCaptureDev

可以用于录音的音频设备类型

语法

```
enum EAudioCaptureDev
{
        eAudCap_Speaker = 1,
        eAudCap_Microphone = 2
};
```

成员

eAudCap_Speaker
扬声器，泛指音频输出设备。

eAudCap_Microphone
麦克风，泛指音频输入设备。

备注

系统中的可以用于录音的音频设备，在本 SDK 中抽象为“输入设备”和“输出设备”两类，无论是 Windows XP 还是在 Windows 7/8/10 上，应用程序都使用相同的接口进行录音的控制。

请参阅

Audio_GetDevCount
Audio_GetDevName
Audio_GetDevId
Audio_SetCurrent
Audio_GetCurrent
Audio_Enable
Audio_SetVolume

9. 编码保存和直播

编码保存和直播

Encoder_IsUsable	检查视频编码器是否可以使用。
Encoder_SetCurrent	选择要使用的视频编码器。
Encoder_GetCurrent	取得当前选择的视频编码器。
Encoder_SetProfile	设置视频编码时使用的规格（profile）配置。
Encoder_GetProfile	取得视频编码时使用的规格（profile）配置。
Encoder_SetPreset	设置视频编码时使用的预设方案（perset）配置。
Encoder_GetPreset	取得视频编码时使用的预设方案（perset）配置。
Encoder_SetBitrate	设置视频编码时使用的码率和码率控制的方式。

Encoder_GetBitrate	取得视频编码时使用的码率和码率控制的方式。
Encoder_SetFrameParams	设置视频编码的关键帧、参考帧等参数。
Encoder_GetFrameParams	取得视频编码的关键帧、参考帧等参数。
Encoder_SetAudioParams	设置音频编码的采样率、码率等参数。
Encoder_GetAudioParam	取得音频编码的采样率、码率等参数。
Encoder_AddSaveFile	添加一个编码后输出（保存和上传）的目标。
Encoder_GetSaveFileCount	取得已经添加的编码后输出目标的的数量。
Encoder_GetSaveFileInfo	取得指定的编码后输出目标的信息。
Encoder_RemoveSaveFile	删除一个编码后的输出目标。
Encoder_Start	开始音视频编码、文件保存、上传。
Encoder_End	停止音视频编码、文件保存、上传。
SEncoderSaveFile	编码后输入（保存文件、上传）的设置。
SEncStatusEncodeFps	编码的帧率和总时长等，发出 eEncNotifyEncodeFps 通知时的相关信息。
SEncStatusUploadBitrate	当前的上传速率，发出 eEncNotifyUploadBitrate 通知时的相关信息。
SEncStatusConnectFail	重连失败，发出 eEncNotifyReConnectFail 通知时的相关信息。
SEncStatusDiscardPacks	因为上传缓慢，丢弃了部分编码后的视频数据，发出 eEncNotifyDiscardPacks 通知时的相关信息。
EVideoEncoder	视频编码器类型。
EVideoProfile	H.264 的编码规格。
EVideoPreset_x264	x264 视频编码器的预设方案。
EVideoPreset_Cuda	Nvidia CUDA 编码器的预设方案。

EVideoPreset_Nvenc	Nvidia NVENC 编码器的预设方案。
EVideoPreset_Intel	Intel 编码器的预设方案。
EVideoRateMode	视频编码时，对码率的控制方式。
EVideoCSP	视频编码时，视频流使用的色彩空间。
EAudioEncoder	音频编码器。
EAudioInSamples	音频录制和编码的采样率。
EFileContainer	音视频编码后数据流输出的方式。

9.1 Encoder_IsUsable

检查视频编码器是否可以使用。

语法

```
BOOL WINAPI Encoder_IsUsable(
        IN            EVideoEncoder eEncoder,
        OUT          LPDWORD pSupportProfileMask,
        OUT          LPDWORD pSupportPresetMask,
        OUT          LPDWORD pSupportColorSpaceMask
);
```

参数

eEncoder

编码器类型，为下列值之一：

VE_X264　　x264　软件编码器

VE_CUDA　　nvidia　CUDA　硬件加速编码器

VE_NVENC　　nvidia　NVENC　硬件编码器

VE_INTEL　　intel　核显加速编码器

pSupportProfileMask

指定的编码器支持的　Profile　类型，位掩码方式表示。位值为　0　表示不支持，值为　1　表示可用。

低　0　位为表示　VF_Auto　是否可用，

低　1　位表示　VF_BaseLine　是否可用，以此类推。

pSupportPresetMask

指定的编码器支持的　Preset　类型，位掩码方式表示。

pSupportColorSpaceMask

指定的编码器支持的输出色彩空间类型，位掩码方式表示。

返回值

成功返回　TRUE　表示编码器可以使用，失败返回　FALSE。

如果失败，可以使用　GetLastError()　取得错误代码。

备注

在没有安装 Nvidia 显卡的电脑上，CUDA 和 NVENC 不可用。
intel 编码器 SDK 暂时不支持。

如何通过返回的 Mask 检查特性是否可用？
例如当检查 VE_X264 编码器获得了 SupportPresetMask，要检查它支持哪些 perset，可以参考下面的代码：

```
for ( int i = VP_x264_UltraFast; i <= VP_x264_Placebo; ++i )
{
        if ( SupportPresetMask & ( 1 << i ) )
                printf( "%s OK.", video_preset_x264_names[i] );
}
```

其它的特性的检查方式和上面类似。

请参阅

EVideoEncoder
Encoder_SetCurrent
Encoder_SetProfile
Encoder_SetPreset

9.2 Encoder_SetCurrent

选择要使用的视频编码器。

语法

```
BOOL WINAPI Encoder_SetCurrent(
        IN          EVideoEncoder eEncoder
);
```

参数

eEncoder
编码器类型，详细内容请参阅 EVideoEncoder 的说明。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

EVideoEncoder
Encoder_IsUsable
Encoder_SetProfile
Encoder_SetPreset

9.3 Encoder_GetCurrent

取得当前选择的视频编码器。

语法

```
EVideoEncoder WINAPI Encoder_GetCurrent();
```

参数

无

返回值

返回当前选择的视频编码器类型。

请参阅

EVideoEncoder
Encoder_IsUsable
Encoder_SetCurrent
Encoder_SetProfile
Encoder_SetPreset

9.4 Encoder_SetProfile

设置视频编码时使用的规格（profile）配置。

语法

```
BOOL WINAPI Encoder_SetProfile(
        IN          EVideoEncoder eEncoder,
```

```
        IN              EVideoProfile eProfile
);
```

参数

eEncoder

编码器类型，详细内容请参阅 EVideoEncoder 的说明。

eProfile

编码规格，类型为 EVideoProfile 枚举，为下列值之一：

VF_Auto 不设置或者自动，默认值。

VF_BaseLine （最低 Profile）级别支持 I/P 帧，只支持无交错（Progressive）和 CAVLC，一般用于低阶或需要额外容错的应用，比如视频通话、手机视频等。

VF_Main （主要 Profile）级别提供 I/P/B 帧，支持无交错（Progressive）和交错（Interlaced），同样提供对于 CAVLC 和 CABAC 的支持，用于主流消费类电子产品规格如低解码（相对而言）的 mp4、便携的视频播放器、PSP 和 Ipod 等。

VF_High （高端 Profile，也叫 FREXt）级别在 Main 的基础上增加了 8x8 内部预测、自定义量化、无损视频编码和更多的 YUV 格式（如 4：4：4）用于广播及视频碟片存储（蓝光影片），高清电视的应用。

返回值

成功返回 TRUE，失败返回 FALSE。

如果失败，可以使用 GetLastError() 取得错误代码。

备注

这个函数建议不要调用，因为设置编码规格会覆盖其它的预设参数，除非解码环境只能特定的规格，例如某些特定的硬件播放器。

请参阅

EVideoEncoder
EVideoProfile
Encoder_GetProfile
Encoder_SetCurrent
Encoder_SetPreset
Encoder_SetBitrate
Encoder_SetFrameParams

9.5 Encoder_GetProfile

取得视频编码时使用的规格（profile）配置。

语法

```
EVideoProfile WINAPI Encoder_GetProfile(
        IN          EVideoEncoder eEncoder
);
```

参数

eEncoder

编码器类型，详细内容请参阅 EVideoEncoder 的说明。

返回值

EVideoProfile 类型的枚举值。

请参阅

EVideoEncoder
EVideoProfile
Encoder_SetProfile
Encoder_SetCurrent
Encoder_SetPreset
Encoder_SetBitrate
Encoder_SetFrameParams

9.6 Encoder_SetPreset

设置视频编码时使用的预设方案（perset）配置。

语法

```
BOOL WINAPI Encoder_SetPreset(
        IN          EVideoEncoder eEncoder,
```

```
        IN              EVideoPreset ePreset
);
```

参数

eEncoder

编码器类型，详细内容请参阅 EVideoEncoder 的说明。

ePreset

对视频编码的各种算法参数的预设方案。

参数值根据 eEncoder 的设置不同，对应使用下列不同枚举类型的成员：

编码器	枚举类型
VE_X264	EVideoPreset_x264
VE_CUDA	EVideoPreset_Cuda
VE_NVENC	EVideoPreset_Nvenc
VE_INTEL	EVideoPreset_Intel

返回值

成功返回 TRUE，失败返回 FALSE。

如果失败，可以使用 GetLastError() 取得错误代码。

备注

不同的预设方案对视频编码的质量和速度影响极大，特别是对于 x264 软件编码来说，在相同的码率下，不同的预设方案有显示的画面质量差距，但编码的速度也有极大的差距。预设值从低到高，对画面质量的影响主要在于运动侦测、评估这方面精度的提升，以及向前向后参考帧的数量的增加。依靠算法复杂度的提高，来提高画面质量，同时由于计算量增加会造成编

码速度的下降。

不同的预设方案下，视频编码的速度也不是恒定不变的，当画面很少有运动物体时，编码速度较快，当画面运动物体很多，甚至像在游戏中奔跑时整个画面一直在变化时，编码速度就会迅速下降。特别对于 x264 软件编码方案，极易造成 CPU 占用率达到 90% 以下，而编码速度却达不到实时。根据经验，即使电脑使用的 i7 CPU，如果使用 x264 编码器编码 1920*1080*30fps 的视频，perset 设置也不应高于 VP_x264_Medium。否则当出现连续几秒的运动画面时，编码就无法达到实时，当直播时会造成严重的延迟。不过，如果使用 NVENC 的纯硬件方案，就几乎不会造成影响了，但它在相同码率下，画面质量比 x264 略低。

然而如果设置的视频码率较低的话，即使使用较高的预设方案，对画面质量的提升也不明显，反而会造成编码速度的下降。如何合理的设置码率，请参阅 Encoder_SetBitrate 中相关的说明。

请参阅

Encoder_SetBitrate
EVideoEncoder
EVideoPreset_x264
EVideoPreset_Cuda
EVideoPreset_Nvenc
EVideoPreset_Intel
Encoder_SetCurrent
Encoder_SetProfile
Encoder_GetPreset
Encoder_SetFrameParams

9.7 Encoder_GetPreset

取得视频编码时使用的预设方案（perset）配置。

语法

```
EVideoPreset WINAPI Encoder_GetPreset(
        IN          EVideoEncoder eEncoder
);
```

参数

eEncoder
编码器类型，详细内容请参阅 EVideoEncoder 的说明。

返回值

指定的编码器当前设置的预设方案枚举值。

请参阅

EVideoEncoder
Encoder_SetCurrent
Encoder_GetProfile
Encoder_SetPreset
Encoder_GetBitrate
Encoder_GetFrameParams

9.8 Encoder_SetBitrate

设置视频编码时使用的码率和码率控制的方式。

语法

```
BOOL WINAPI Encoder_SetBitrate(
        IN          EVideoRateMode eMode,
        IN          INT iBitrate,
        IN          INT iBitrateMax,
        IN          INT iVbvSize
);
```

参数

eMode

码率控制的方式，为下列值之一：

VR_AverageBitrate　　平均码率(ABR)，码率允许小幅度波动，适合画面运动部分较少的情况，例如战棋类、卡牌类游戏、坐着几乎不动的美女直播等。

VR_VariableBitrate　　可变码率(VBR)，码率在 iBitrateMax 设置的范围内波动，适合画面经常运动的情况，例如直播舞蹈、户外、3D 游戏。

VR_ConstantBitrate　　固定码率(CBR)，码率基本上不变化。网络带宽较小，不能承受突发的高码率数据时适用。

VR_ConstantQP　　恒定质量(CQP)，基本上只适合于录制视频仅保存在本机，不进行直播的时候。

iBitrate

基本的码率设置，值的有效范围是 8 到 102400， 单位为 kbps，和网络带宽术语意思一样。例如网络带宽是 2M，就是 2000kbps，如果要用这个网络直播，参数值就不能超过 2000。

iBitrateMax

最大码率，值的有效范围是 0 到 iBitrate * 10， 单位为 kbps。
如果设置为 0，那么不使用最大码率控制，否则应该设置为大于等于 iBitrate 的值。
最大码率的设置只在 eMode 为 VR_VariableBitrate 和 VR_ConstantQP 时有效。

iVbvSize

码率控制缓冲区的大小，值的有效范围是 0 到 iBitrate * 10， 单位为 kbps。
如果设置为 0，那么不使用最大码率控制，否则应该设置为大于等于 iBitrateMax 的值。
缓冲区的设置只在 eMode 为 VR_VariableBitrate 和 VR_ConstantQP 时有效。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

EVideoRateMode
Encoder_SetPreset
Encoder_GetBitrate
Encoder_SetFrameParams

9.9 Encoder_GetBitrate

取得视频编码时使用的码率和码率控制的方式。

语法

```
BOOL WINAPI Encoder_GetBitrate(
        OUT         EVideoRateMode* pMode,
        OUT         INT* pBitrate,
        OUT         INT* pBitrateMax,
        OUT         INT* pVbvSize
);
```

参数

pMode
返回码率控制方式，如果参数为 NULL，就忽略此参数。

pBitrate
返回基本码率设置，如果参数为 NULL，就忽略此参数。

pBitrateMax
返回最大码率设置，如果参数为 NULL，就忽略此参数。

pVbvSize
返回码率控制缓冲区设置，如果参数为 NULL，就忽略此参数。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

EVideoRateMode
Encoder_GetPreset
Encoder_SetBitrate
Encoder_GetFrameParams

9.10 Encoder_SetFrameParams

设置视频编码的关键帧、参考帧等参数。

语法

```
BOOL WINAPI Encoder_SetFrameParams(
        IN              INT iGop,
        IN              INT iGopMin,
        IN              INT iRefFrames,
        IN              INT iBFrames
);
```

参数

iGop

Group of Pictures，两个 IDR 帧之间间隔的帧数。相当于每多少帧产生一个关键帧。GOP 通常要大于帧率，否则关键帧太密集会导致码率难以控制，编码的图像质量也不好。

参数如果为 0，则使用默认值，否则应该设置为大于 0 的值。

iGopMin

限制 I 帧最小间隔，I 帧也是关键帧，因为编码器内部检查帧间图像变化超某个阀值时，就会把这帧编码为关键帧，为了避免图像剧烈变化时把每帧都编码为关键帧，就可以设置一个最小间隔限制。

参数如果为 0，则使用默认值，否则应该设置为大于 0 的值。

iRefFrames

参考帧的最大数量。在视频编码时当前帧与之前的帧进行比较，寻找差异，用于比较的帧就叫参考帧。

参数如果为 0，则使用默认值，否则应该设置为 1 到 16 之间的值。

iBFrames

限制最大的连续的 B 帧数量。B 帧会同时参考前面的帧和后面的来压缩本帧，仅记录本帧与前后帧的差异，可以提高压缩比，但对于视频直播会增加延迟时间。

参数如果为 0，则不使用 B 帧，否则应该设置为大于 0 小于 GOP 的值。

返回值

成功返回 TRUE，失败返回 FALSE。

如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Encoder_SetPreset
Encoder_SetBitrate
Encoder_GetFrameParams

9.11 Encoder_GetFrameParams

取得视频编码的关键帧、参考帧等参数。

语法

```
BOOL WINAPI Encoder_GetFrameParams(
        OUT          INT* pGop,
        OUT          INT* pGopMin,
        OUT          INT* pRefFrames,
        OUT          INT* pBFrames
);
```

参数

pGop
获取两个 IDR 帧之间间隔的帧数。如果参数为 NULL，则忽略该参数。

pGopMin
获取 I 帧最小间隔的限制值。如果参数为 NULL，则忽略该参数。

pRefFrames
获到参考帧的最大数量。如果参数为 NULL，则忽略该参数。

pBFrames
获取最大的连续的 B 帧数量的限制值。如果参数为 NULL，则忽略该参数。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Encoder_GetPreset
Encoder_GetBitrate
Encoder_SetFrameParams

9.12 Encoder_SetAudioParams

设置音频编码的采样率、码率等参数。

语法

```
BOOL WINAPI Encoder_SetAudioParams(
        IN          WORD nChannels,
        IN          EAudioInSamples eSample,
        IN          WORD wBits,
        IN          EAudioEncoder eEncoder,
        IN          INT iEncBitsRate
);
```

参数

nChannels

音频编码的声道数，只能是 1 或 2，表示单声道或立体声。

如果实际录制到的声音与设置的声道数不一样，SDK 内部会根据声音数据的声道布局合理地合并成为设置的声道数。

如果声道数设置为 0，则本次编码数据流中将不包含音频流数据。

eSample

音频的采样率。只能是下列的值之一：

Aud_Inp_Samp_11025	=	11025	11Khz
Aud_Inp_Samp_22050	=	22050	22Khz
Aud_Inp_Samp_44100	=	44100	44Khz
Aud_Inp_Samp_48000	=	48000	48Khz

采样率越高，那么对录音中的高频成份保留越好，声音也越细腻，但对编码的码率要求也越高。

wBits

音频的采样位数，可以是 16、24、32 表示指定位数的采样，并以整数方式记录采样的音频数据。如果是其它的任何值，就会使用 32 位浮点数的方式记录采样的音频数据。

eEncoder

音频编码器的选择，当前只支持 aac 编码，参数值只能是 Aud_Enc_AAC。

iEncBitsRate

音频编码的码率，单位是 kbps，值范围是 0 到 256。

如果设置为 0，在编码时会自动设置为视频码率的 1/10，且大于等于 32 小于等于 256。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

EAudioEncoder
EAudioInSamples
Encoder_GetAudioParam
Encoder_SetBitrate

9.13 Encoder_GetAudioParam

取得音频编码的采样率、码率等参数。

语法

```
BOOL WINAPI Encoder_GetAudioParam(
        OUT         WORD* pChannels,
        OUT         EAudioInSamples* pSample,
        OUT         WORD* pBits,
        OUT         EAudioEncoder* pEncoder,
        OUT         INT* pEncBitsRate
);
```

参数

pChannels
获取音频编码的声道数。如果参数为 NULL，则忽略该参数。

pSample
获取音频编码的采样率设置。如果参数为 NULL，则忽略该参数。

pBits
获取音频编码的采样位数设置。如果参数为 NULL，则忽略该参数。

pEncoder
获取音频编码的编码器设置。如果参数为 NULL，则忽略该参数。

pEncBitsRate
获取音频编码的编码设置。如果参数为 NULL，则忽略该参数。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

EAudioEncoder
EAudioInSamples
Encoder_SetBitrate

9.14 Encoder_AddSaveFile

添加一个编码后输出（保存和上传）的目标。

语法

```
INT WINAPI Encoder_AddSaveFile(
        IN            SEncoderSaveFile* pSaveFile
);
```

参数

pSaveFile

输出目标设置的结构指针，不能为 NULL。
详细信息请参阅 SEncoderSaveFile 的说明。

返回值

成功返回当前添加的输出目标的索引值，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

可以添加多种输出方式的多个输出目标。也就是可以同时保存为多种格式的多个文件，并直播到多个 RTMP 地址和锐动云直播地址。

请参阅

SEncoderSaveFile
Encoder_GetSaveFileCount
Encoder_GetSaveFileInfo
Encoder_RemoveSaveFile
Encoder_Start
Encoder_End

9.15 Encoder_GetSaveFileCount

取得已经添加的编码后输出目标的的数量。

语法

```
INT WINAPI Encoder_GetSaveFileCount();
```

参数

无

返回值

成功返回数量，失败返回 -1。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Encoder_AddSaveFile
Encoder_GetSaveFileInfo
Encoder_RemoveSaveFile
Encoder_Start
Encoder_End

9.16 Encoder_GetSaveFileInfo

取得指定的编码后输出目标的信息。

语法

```
BOOL WINAPI Encoder_GetSaveFileInfo(
        IN          INT iIndex,
        OUT         SEncoderSaveFile* pSaveFile
);
```

参数

iIndex
索引值，必须大于等于 0 小于 Encoder_GetSaveFileCount 的返回值。

pSaveFile
输出目标设置的结构指针，不能为 NULL。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Encoder_GetSaveFileCount
SEncoderSaveFile
Encoder_AddSaveFile
Encoder_RemoveSaveFile
Encoder_Start
Encoder_End

9.17 Encoder_RemoveSaveFile

删除一个编码后的输出目标。

语法

```
BOOL WINAPI Encoder_RemoveSaveFile(
        IN              INT iIndex
);
```

参数

iIndex
索引值，必须大于等于 0 小于 Encoder_GetSaveFileCount 的返回值。

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Encoder_GetSaveFileCount
Encoder_AddSaveFile
Encoder_GetSaveFileInfo
Encoder_Start
Encoder_End

9.18 Encoder_Start

开始音视频编码、文件保存、上传。

语法

```
BOOL WINAPI Encoder_Start();
```

参数

无

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

备注

函数调用成功后会立即返回，真正的文件打开、网络连接等操作是异步进行的。成功开始后会发出 eEncNotifyStarted 通知，开始的过程中如果发生错误，会发生 eEncNotifyStoped 通知。

请参阅

Encoder_AddSaveFile
Encoder_GetSaveFileCount
Encoder_GetSaveFileInfo
Encoder_RemoveSaveFile
Encoder_End
IEncoder_ENotify

9.19 Encoder_End

停止音视频编码、文件保存、上传。

语法

```
BOOL WINAPI Encoder_End();
```

参数

无

返回值

成功返回 TRUE，失败返回 FALSE。
如果失败，可以使用 GetLastError() 取得错误代码。

请参阅

Encoder_AddSaveFile
Encoder_GetSaveFileCount
Encoder_GetSaveFileInfo
Encoder_RemoveSaveFile
Encoder_Start
IEncoder_ENotify

9.20 SEncoderSaveFile

编码后输入（保存文件、上传）的设置。

语法

```
struct SEncoderSaveFile
{
        EFileContainer eContainer;
        LPCWSTR szSavePath;
```

```
        LPCWSTR szTitle;
        LPCWSTR szContent;
        union
        {
                INT64 iSplitSize;
                struct
                {
                        DWORD dwDelaySecond;
                        BOOL bAdaptiveNetwork;
                } ;
        } ;
};
```

成员

eContainer

输入的类型，请参阅 EFileContainer 中的说明。

szSavePath

目标路径字符串。

如果是存储为文件，则是类似于 "C:\video\abc.flv" 这样的完整的文件路径。

如果是上传到 RTMP 地址，则是类似于 "rtmp://192.168.0.1/live/abc" 或 rtmp://www.17rd.com/live/abc 这样的 URL。

如果是上传到锐动云服务器，则是一个由数字和小写字母组成的小于 32 个字符的字符串，这个字符串由应用程序提供，用于唯标识用户。例如用户在某网站注册，在应用程序中登录后，应用程序使用用户名或用户名 md5 后的字符串，或其它的任何可以区分用户的字符作为输入参数。

szTitle

保存文件或直播时的标题。

当 eContainer 为 Mix_RdCloud 时，不用为空字符串或 NULL，其它输出类型则可以使用空字符串或 NULL。

对于保存文件，如果文件格式支持，那么将会尝试把标题写入到录制的文件中。

对于直播到锐动云服务器，则作为直播的标题。

对于直播到 RTMP 地址，这个参数没有意义。

szContent

保存文件或直播时的内容描述。可以为空字符串或 NULL。

对于保存文件，如果文件格式支持，那么将会尝试把内容描述写入到录制的文件中。

对于直播到锐动云服务器，则作为直播的内容描述。

对于直播到 RTMP 地址，这个参数没有意义。

unnamed-0::iSplitSize

保存到文件时，文件分卷的大小限制。可以为 0 表示自动。

unnamed-0::dwDelaySecond

当直播时，指定编码后的音视频流将在本地缓存多少秒后再上传，也就是音视频延迟一段时间后再开始直播。设置为 0 表示不缓存，立即上传。

unnamed-0::bAdaptiveNetwork

当直播时，如果网络条件较差，是否自动调整音视频编码的码率，使音视频能尽量流畅地直播。

请参阅

EFileContainer
Encoder_AddSaveFile

Encoder_GetSaveFileCount
Encoder_GetSaveFileInfo
Encoder_RemoveSaveFile
Encoder_Start
Encoder_End

9.21 SEncStatusEncodeFps

编码的帧率和总时长等，发出 eEncNotifyEncodeFps 通知时的相关信息。

语法

```
typedef struct SEncStatusEncodeFps
{
        INT64 iRecordTime;
        INT64 iInpFrameNum;
        INT64 iEncFrameNum;
        INT64 iEncFrameCount;
        float fAvgEncodeFps;
        float fCurEncodeFps;
        float fAvgInputFps;
        float fCurInputFps
};
```

成员

iRecordTime
从开始录制到当前经过的时长（毫秒）。

iInpFrameNum
当前正在输入帧的编号。

iEncFrameNum
当前正在编码帧的编号。

iEncFrameCount
实际编码完成的帧数，不包括跳过的帧数。

fAvgEncodeFps
平均的编码帧率。

fCurEncodeFps
当前编码的帧率。

fAvgInputFps
平均的输入帧率。

fCurInputFps
当前输入的帧率。

请参阅

IEncoder_ENotify
Encoder_Start
Encoder_End

9.22 SEncStatusUploadBitrate

当前的上传速率，发出 eEncNotifyUploadBitrate 通知时的相关信息。

语法

```
typedef struct SEncStatusUploadBitrate
{
        INT64 iConnectUploadMSeconds;
        INT64 iTotalUploadMSeconds;
        DWORD uCurrentUploadBitrate;
        DWORD uAverageUploadBitrate
};
```

成员

iConnectUploadMSeconds
当前连接已经上传的视频时长（毫秒）。

iTotalUploadMSeconds
累计上传的视频时长（毫秒）。

uCurrentUploadBitrate
当前上传比特率。

uAverageUploadBitrate
平均上传比特率（最近一次连接）。

请参阅

IEncoder_ENotify
Encoder_Start
Encoder_End

9.23 SEncStatusConnectFail

重连失败，发出 eEncNotifyReConnectFail 通知时的相关信息。

语法

```
typedef struct SEncStatusConnectFail
{
        DWORD dwErrorCode;
        DWORD dwTryLaterMS
};
```

成员

dwErrorCode
连接失败的错误代码。

dwTryLaterMS
下一次尝试重连是在多少毫秒之后。

请参阅

IEncoder_ENotify
Encoder_Start
Encoder_End

9.24 SEncStatusDiscardPacks

因为上传缓慢，丢弃了部分编码后的视频数据，发出 eEncNotifyDiscardPacks 通知时的相关信息。

语法

```
typedef struct SEncStatusDiscardPacks
{
        INT64 iDiscardStartMSeconds;
        INT64 iDiscardMSeconds;
        DWORD uDiscardFrames;
        DWORD uAverageUploadBitrate
};
```

成员

iDiscardStartMSeconds
丢弃帧的起始时间（毫秒）。

iDiscardMSeconds
丢弃的时长（毫秒）。

uDiscardFrames
丢弃的帧数。

uAverageUploadBitrate
平均上传比特率（最近一次连接）。

请参阅

IEncoder_ENotify
Encoder_Start
Encoder_End

9.25 EVideoEncoder

视频编码器类型。

语法

```
enum EVideoEncoder
{
        VE_X264,
        VE_CUDA,
        VE_NVENC,
        VE_INTEL
};
```

成员

VE_X264
x264 软件编码器。

VE_CUDA
nvidia CUDA 硬件加速编码器。

VE_NVENC
nvidia NVENC 硬件编码器。

VE_INTEL
intel 核显加速编码器。

备注

在没有安装 Nvidia 显卡的电脑上，CUDA 和 NVENC 不可用。
intel 编码器 SDK 暂时不支持。

在使用相同的码率设置，都使用中等或最高的预设方案时，
视频编码的画面质量从高到低依次是：
VE_X264 > VE_INTEL > VE_NVENC > VE_CUDA。
其中除了 VE_CUDA，其它 3 个的编码画面质量差距不明显。

编码速度从高到低依次是：
VE_NVENC > VE_INTEL > VE_CUDA > VE_X264。
其中 VE_INTEL 和 VE_CUDA 的速度要视不同的 CPU 和显卡而定，不一定谁快。

编码时对 CPU 和内存的消耗按从多到少的顺序排列依次是：
VE_X264 > VE_INTEL > VE_CUDA > VE_NVENC。

请参阅

Encoder_IsUsable
Encoder_SetCurrent
Encoder_GetCurrent

9.26 EVideoProfile

H.264 的编码规格。

语法

```
enum EVideoProfile
{
```

```
        VF_Auto,
        VF_BaseLine,
        VF_Main,
        VF_High
};
```

成员

VF_Auto
不设置或者自动，默认值。

VF_BaseLine
（最低 Profile）级别支持 I/P 帧，只支持无交错（Progressive）和 CAVLC，一般用于低阶或需要额外容错的应用，比如视频通话、手机视频等。

VF_Main
（主要 Profile）级别提供 I/P/B 帧，支持无交错（Progressive）和交错（Interlaced），同样提供对于 CAVLC 和 CABAC 的支持，用于主流消费类电子产品规格如低解码（相对而言）的 mp4、便携的视频播放器、PSP 和 Ipod 等。

VF_High
（高端 Profile，也叫 FREXt）级别在 Main 的基础上增加了 8x8 内部预测、自定义量化、无损视频编码和更多的 YUV 格式（如 4：4：4）用于广播及视频碟片存储（蓝光影片），高清电视的应用。

备注

建议不要调用 Encoder_SetProfile 设置编码规格，因为设置编码规格会覆盖其它的预设参数，除非解码环境只能特定的规格，例如某些特定的硬件播放器。

请参阅

Encoder_SetProfile
Encoder_IsUsable
Encoder_SetCurrent
Encoder_GetCurrent
Encoder_GetProfile

9.27 EVideoPreset_x264

x264 视频编码器的预设方案。

语法

```
enum EVideoPreset_x264
{
        VP_x264_UltraFast,
        VP_x264_SuperFast,
        VP_x264_VeryFast,
        VP_x264_Faster,
        VP_x264_Fast,
        VP_x264_Medium,
        VP_x264_Slow,
        VP_x264_Slower,
        VP_x264_VerySlow,
```

```
        VP_x264_Placebo
};
```

成员

VP_x264_UltraFast
编码速度最快，画面质量最差。
几乎当前的任何电脑都能完成每秒 30 帧的 1080p 视频编码。

VP_x264_SuperFast

VP_x264_VeryFast
编码速度很快，画面质量尚可。
大多数电脑，比如 i3、i5 CPU 的都可以完成每秒 30 帧的 1080p 视频编码。

VP_x264_Faster

VP_x264_Fast
编码速度快，画面质量也不错。
使用 i7 CPU 的电脑应该都可以完成每秒 30 帧的 1080p 视频编码。

VP_x264_Medium
编码速度和画面质量比较均衡。
使用 i7 CPU 的电脑基本上可以完成每秒 30 帧的 1080p 视频编码。

VP_x264_Slow

编码速度比较慢，画面质量较好。
使用 i7 高端 CPU 的电脑可以完成每秒 30 帧的 1080p 视频编码。

VP_x264_Slower

VP_x264_VerySlow

编码速度非常慢，画面质量很好。
使用 i7 高端 CPU 的电脑可能也无法完成每秒 30 帧的 1080p 视频编码。

VP_x264_Placebo

无损编码。

备注

对各个编码预设和 CPU 编码速度的描述，基于要编码的画面是有间歇性的运动场景，每帧都有差异的情况。例如类似 3D 游戏中偶尔在地图上到处奔跑，偶尔停下，偶尔战斗这样的动态画面。

请参阅

Encoder_SetPreset
Encoder_GetPreset
Encoder_IsUsable
Encoder_SetCurrent
Encoder_GetCurrent

9.28 EVideoPreset_Cuda

Nvidia CUDA 编码器的预设方案。

语法

```
enum EVideoPreset_Cuda
{
        VP_Cuda_PSP,
        VP_Cuda_iPod,
        VP_Cuda_AVCHD,
        VP_Cuda_BD,
        VP_Cuda_HDV1440,
        VP_Cuda_ZuneHD,
        VP_Cuda_FlipCam
};
```

成员

VP_Cuda_PSP

VP_Cuda_iPod

VP_Cuda_AVCHD

VP_Cuda_BD

VP_Cuda_HDV1440

VP_Cuda_ZuneHD

VP_Cuda_FlipCam

备注

CUDA 编码器的画面质量较差，特别是在低码率设置下。性能方面没有问题，基本上都可以完成每秒 30 帧的 1080p 视频编码。

请参阅

Encoder_SetPreset
Encoder_GetPreset
Encoder_IsUsable
Encoder_SetCurrent
Encoder_GetCurrent

9.29 EVideoPreset_Nvenc

Nvidia NVENC 编码器的预设方案。

语法

```
enum EVideoPreset_Nvenc
{
```

```
        VP_Nvenc_Default,
        VP_Nvenc_HighPerformance,
        VP_Nvenc_HighQuality,
        VP_Nvenc_BlurayDisk,
        VP_Nvenc_LowLatencyDefault,
        VP_Nvenc_LowLatencyHP,
        VP_Nvenc_LowLatencyHQ,
        VP_Nvenc_LosslessDefault,
        VP_Nvenc_LosslessHP
};
```

成员

VP_Nvenc_Default
默认方案。

VP_Nvenc_HighPerformance
高性能方案。

VP_Nvenc_HighQuality
高品质方案。

VP_Nvenc_BlurayDisk
蓝光光碟的高品质方案。

VP_Nvenc_LowLatencyDefault
低延迟的默认方案。

VP_Nvenc_LowLatencyHP
低延迟的高性能方案。

VP_Nvenc_LowLatencyHQ
低延迟的高品质方案。

VP_Nvenc_LosslessDefault
无损编码的默认方案。

VP_Nvenc_LosslessHP
无损编码的高性能方案。

备注

NVENC 的不同预设方案编码的画面质量其实差距不大，因为它是纯硬件实现的，实际的可配置的参数不多，因此各种预设方案的编码性能差异也不大。

请参阅

Encoder SetPreset
Encoder GetPreset
Encoder IsUsable
Encoder SetCurrent
Encoder GetCurrent

9.30 EVideoPreset_Intel

Intel 编码器的预设方案。

语法

```
enum EVideoPreset_Intel
{
        VP_Intel_Speed,
        VP_Intel_Balanced,
        VP_Intel_Quality
};
```

成员

VP_Intel_Speed

VP_Intel_Balanced

VP_Intel_Quality

备注

暂时不支持。

请参阅

Encoder_SetPreset
Encoder_GetPreset

Encoder_IsUsable
Encoder_SetCurrent
Encoder_GetCurrent

9.31 EVideoRateMode

视频编码时，对码率的控制方式。

语法

```
enum EVideoRateMode
{
        VR_AverageBitrate,
        VR_VariableBitrate,
        VR_ConstantBitrate,
        VR_ConstantQP
};
```

成员

VR_AverageBitrate
平均码率(ABR)，码率允许小幅度波动，适合画面运动部分较少的情况，例如战棋类、卡牌类游戏、坐着几乎不动的美女直播等。

VR_VariableBitrate
可变码率(VBR)，码率在 iBitrateMax 设置的范围内波动，适合画面经常运动的情况，例如直播舞蹈、户外、3D 游戏。

VR_ConstantBitrate
固定码率(CBR)，码率基本上不变化。网络带宽较小，不能承受突发的高码率数据时适用。

VR_ConstantQP
恒定质量(CQP)，基本上只适合于录制视频仅保存在本机，不进行直播的时候。

请参阅

Encoder_SetBitrate
Encoder_SetPreset
Encoder_GetFrameParams

9.32 EVideoCSP

视频编码时，视频流使用的色彩空间。

语法

```
enum EVideoCSP
{
        Vid_CSP_I420,
        Vid_CSP_YV12,
        Vid_CSP_NV12,
        Vid_CSP_NV21,
        Vid_CSP_I422,
        Vid_CSP_YV16,
```

```
        Vid_CSP_NV16,
        Vid_CSP_YUY2,
        Vid_CSP_UYVY,
        Vid_CSP_V210,
        Vid_CSP_I444,
        Vid_CSP_YV24,
        Vid_CSP_AYUV,
        Vid_CSP_RGB,
        Vid_CSP_BGR,
        Vid_CSP_ARGB,
        Vid_CSP_ARGB10
};
```

成员

Vid_CSP_I420

Vid_CSP_YV12

Vid_CSP_NV12

Vid_CSP_NV21

Vid_CSP_I422

Vid_CSP_YV16

Vid_CSP_NV16

Vid_CSP_YUY2

Vid_CSP_UYVY

Vid_CSP_V210

Vid_CSP_I444

Vid_CSP_YV24

Vid_CSP_AYUV

Vid_CSP_RGB

Vid_CSP_BGR

Vid_CSP_ARGB

Vid_CSP_ARGB10

请参阅

9.33 EAudioEncoder

音频编码器。

语法

```
enum EAudioEncoder
{
        Aud_Enc_AAC,
        Aud_Enc_PCM
};
```

成员

Aud_Enc_AAC

Aud_Enc_PCM

备注

当前只支持 AAC。

请参阅

Encoder_SetAudioParams
Encoder_GetAudioParam

9.34 EAudioInSamples

音频录制和编码的采样率。

语法

```
enum EAudioInSamples
{
        Aud_Inp_Samp_11025 = 11025,
        Aud_Inp_Samp_22050 = 22050,
        Aud_Inp_Samp_44100 = 44100,
        Aud_Inp_Samp_48000 = 48000
};
```

成员

Aud_Inp_Samp_11025

Aud_Inp_Samp_22050

Aud_Inp_Samp_44100

Aud_Inp_Samp_48000

请参阅

Encoder_SetAudioParams
Encoder_GetAudioParam

9.35 EFileContainer

音视频编码后数据流输出的方式。

语法

```
enum EFileContainer
{
        Mix_FLV,
        Mix_MP4,
        Mix_AVI,
        Mix_RTMP,
        Mix_RdCloud,
        Vid_264,
        Aud_ONLY,
        File_Container_TypeCount
};
```

成员

Mix_FLV
保存为 flv 格式的文件。

Mix_MP4
保存为 mp4 格式的文件。

Mix_AVI
保存为 avi 格式的文件。不建议，因为 avi 格式规范太老，与 h264 和 aac 编码的兼容性不太好。

Mix_RTMP
直播到 rtmp 地址。

Mix_RdCloud
直播到锐动云服务器。

Vid_264
保存 h.264 裸流数据到文件，不包含音频流。

Aud_ONLY
只保存音频流到文件，不包含视频流。

File_Container_TypeCount
文件容器类型的数量，这只是一个计数器。

请参阅

Encoder_AddSaveFile
Encoder_GetSaveFileCount
Encoder_GetSaveFileInfo
Encoder_RemoveSaveFile
Encoder_Start
Encoder_End