网络 SDK 编程手册

VERSION 3.0.5.4 (Build 20120605)

2012-06-05

前　言

非常感谢您使用我们公司的设备，我们将为您提供最好的服务。

本手册可能包含技术上不准确的地方或印刷错误，欢迎指正。我们将会定期更新手册的内容。

修订记录

日期	修订内容
2009.02.12	增加局域网内搜索设备接口 H264_DVR_SearchDevice
2009.02.18	增加语音对讲相关接口 H264_DVR_StartVoiceCom_MR，H264_DVR_VoiceComSendData，H264_DVR_StopVoiceCom，H264_DVR_SetTalkMode
2009.09.26	增加以下接口： H264_DVR_StartDVRRecord，H264_DVR_StopDVRRecord，H264_DVR_SetSystemDateTime，H264_DVR_GetDVRWorkState，H264_DVR_ClickKey
2010．08．7	增加以下接口： H264_DVR_StorageManage，H264_DVR_SendNetAlarmMsg，H264_DVR_StartAlarmCenterListen， H264_DVR_StopAlarmCenterListen，H264_DVR_API bool H264_DVR_DelRealDataCallBack，H264_DVR_API long H264_DVR_PlayBackByTime，H264_DVR_API long H264_DVR_GetFileByTime，H264_DVR_API bool H264_DVR_PTZControlEx，H264_DVR_API long H264_DVR_GetDevConfig，H264_DVR_API long H264_DVR_SetDevCon，H264_DVR_API long H264_DVR_SetConfigOverNet
2011．06．27	增加以下接口： H264_DVR_SetConnectTime, H264_DVR_API long CALL_METHOD H264_DVR_SetConfigOverNet, H264_DVR_API long CALL_METHOD H264_DVR_PauseRealPlay, H264_DVR_SetRealDataCallBack_V2, H264_DVR_FindFileByTime, H264_DVR_PlayBackByName_V2, H264_DVR_PlayBackByTime, H264_DVR_PlayBackByTimeEx, CALL_METHOD H264_DVR_GetFileByTime, H264_DVR_CatchPic, H264_DVR_CatchPicInBuffer, H264_DVR_SerialWrite, H264_DVR_SerialRead, H264_DVR_GetDDNSInfo, H264_DVR_MakeKeyFrame
2011．09．03	增加以下接口： H264_DVR_OpenTransComChannel，H264_DVR_CloseTransComChannel，H264_

	DVR_GetDeviceState
2011．09．27	增加数据结构： SDK_CameraAbility，SDK_AllCameraParam
2011. 10. 10	增加以下接口： H264_DVR_CatchPicUI
2011.11.21	增加已下接口：（只用于网络与解码合并情况） H264_DVR_OpenSound H264_DVR_CloseSound H264_DVR_LocalCatchPic H264_DVR_StartLocalVoiceCom H264_DVR_StartLocalRecord H264_DVR_StopLocalRecord H264_DVR_StartLocalPlay H264_DVR_StopLocalPlay H264_DVR_GetPlayPos H264_DVR_SetPlayPos H264_DVR_LocalPlayCtrl H264_DVR_SetFileEndCallBack H264_DVR_SetInfoFrameCallBack H264_DVR_LocalGetColor H264_DVR_LocalSetColor 增加结构体：SDK_LoalPlayAction 改变结构体：H264_DVR_FILE_DATA, H264_DVR_FINDINFO, H264_DVR_FINDINFO (加 hWnd 变量)
2012.5.18	增加以下接口 H264_DVR_SetLocalBindAddress H264_DVR_StartUploadData

	H264_DVR_StopUploadData H264_DVR_StartActiveRigister H264_DVR_StopActiveRigister H264_DVR_SetSubDisconnectCallBack 增加登录类型枚举 SocketStyle 主动注册数据结构 H264_DVR_ACTIVEREG_INFO
2012.8.2	增加以下接口 H264_DVR_SetKeepLifeTim
2013.2.25	增加搜索设备协议枚举 SDK_TransferProtocol_V2 增加一下接口 H264_DVR_SearchDeviceEX

目　录

1. 简 介

1.1 概述

欢迎使用我公司网络SDK编程手册，网络SDK是软件开发商在开发我司网络硬盘录像机监控联网应用时的开发套件。本文档详细描述了开发包中各个函数实现的功能、接口及其函数之间的调用关系和示例实现。

开发包所包括的文件有:

网络库	NetSDK	头文件
	NetSDK.lib	Lib 文件
	NetSDK.dll	接口库
辅助库	DllDeinterlace.dll	解码辅助库
	H264Play.dll	解码辅助库
	hi_h264dec_w.dll	解码辅助库

1.2 适用性

- 支持网络硬盘录像机的监视、回放、报警、远程配置、日志查询等功能。
- 支持 TCP 传输模式，设备端同时支持 10 个 TCP 连接。
- 可通过 SDK 回调接口开发流媒体转发、回放、报警等服务器程序。
- 客户端可以采用多种分辨率进行图像预览，支持的分辨率包括：QCIF、CIF 、2CIF、、HalfD1、D1，VGA（640×480）等
- SDK 在录像回放/下载时,同一登陆 ID 对于同一通道在同一时间回放和下载不可同时进行操作。
- SDK 性能与设备的运行情况和运行客户端的计算机 CPU 能力密切相关,理论上能同时支持 2000 个用户注册; 同时支持 2000 路网络预览和网络回放;同时支持 2000 路报警上传;在图象显示方面同时支持 300 路。

设计原则

1.3 编程说明

- 初始化和清除

1、 使用网络客户端软件包首先调用 **H264_DVR_Init()**对系统进行初始化，应用程序退出时调用 **H264_DVR_Cleanup()**释放所有占用的资源。

2、 大多数函数调用均应该在 **H264_DVR_Init()**之后，**H264_DVR_Cleanup()**之前，而 H264_DVR_GetLastError 可以在任何时候调用等等。

- 用户登录和注销

用户在访问前端设备之前必须通过调用 **H264_DVR_Login ()**登录到前端设备上，如果登陆的软件是特殊的(不是 web)可以调用 **H264_DVR_LoginEx()**指定登陆的软件类型,登录成功后返回一个全局唯一的句柄。此句柄就像一个会话通道，之后该用户可通过此句柄访问前端设备。退出该会话时则通过 H264_DVR_Logout()函数在前端设备上注销此句柄以终止该会话通道的使用。建立连接与登录是同步的。

- 心跳功能

在本开发包中提供自动心跳功能（20 秒一次心跳）当设备断开能及时回调给客户端。

- 同步与异步

异步通过设置回调函数的方式实现，网络数据通过回调函数传达到应用程序,有些异步在设置后返回请求句柄，结束请求时将请求句柄提供给 SDK 以注销相关资源。

- 回调函数

一般都有 *dwUser* 参数, 由用户自定义需要的数据, 一般用来传入类对象指针, 方便回调处理在类中实现,回调应用都可以采取这种方式。

1.4 典型调用顺序

A. 初始化

SDK 初始化	*H264_DVR_Init ()*

B. SDK 功能信息获取

设置报警消息回调	H264_DVR_SetDVRMessCallBack ()

C. 登录连接设备

登入设备	H264_DVR_Login () H264_DVR_LoginEx ()
报警消息订阅	H264_DVR_SetupAlarmChan ()

D. 设备功能操作与信息获取

系统参数配置	H264_DVR_GetDevConfig () H264_DVR_SetDevConfig ()
查询日志	H264_DVR_FindDVRLog ()
云台控制	H264_DVR_PTZControl () H264_DVR_PTZControlEx ()
透明串口控制	H264_DVR_OpenTransComChannel ( ) H264_DVR_CloseTransComChannel ( )

E. 实时监视通道

打开监视通道	H264_DVR_RealPlay () H264_DVR_StopRealPlay ()
监视数据回调保存	H264_DVR_SetRealDataCallBack ()

F. 回放/下载通道

查询录像	H264_DVR_FindFile () H264_DVR_FindFileByTime()
回放及控制	H264_DVR_PlayBackByName H264_DVR_PlayBackByName_V2()

	H264_DVR_PlayBackByTime()
	H264_DVR_PlayBackByTimeEx()
	H264_DVR_PlayBackControl()
	H264_DVR_StopPlayBack()
下载	H264_DVR_GetFileByName ()
	H264_DVR_GetFileByTime()
	H264_DVR_GetDownloadPos()
	H264_DVR_StopGetFile ()

G. 远程控制

远程升级	H264_DVR_Upgrade()
	H264_DVR_GetUpgradeState()
	H264_DVR_CloseUpgradeHandle()
重启/清除日志	H264_DVR_ControlDVR ()

H. 注销断开设备

停止报警消息订阅	H264_DVR_CloseAlarmChan ()
断开连接	H264_DVR_Logout ()

I. 释放 SDK 资源

SDK 退出	H264_DVR_Cleanup ()

2 数据结构定义

2.1 客户端数据结构

```
//云台操作类型
typedef enum PTZ_ControlType
{
    TILT_UP = 0,          //上
    TILT_DOWN,            //下
    PAN_LEFT,             //左
    PAN_RIGHT,            //右
    PAN_LEFTTOP,          //左上
    PAN_LEFTDOWN,         //左下
    PAN_RIGTHTOP,         //右上
    PAN_RIGTHDOWN,        //右下
    ZOOM_IN,              //变倍大
    ZOOM_OUT,             //变倍小
    FOCUS_FAR,            //焦点后调
    FOCUS_NEAR,           //焦点前调
    IRIS_OPEN,            //光圈扩大
    IRIS_CLOSE,           //光圈缩小

    EXTPTZ_OPERATION_ALARM,         ///< 报警功能
    EXTPTZ_LAMP_ON,                 ///< 灯光开
    EXTPTZ_LAMP_OFF,                //灯光关
    EXTPTZ_POINT_SET_CONTROL,       //设置预置点
    EXTPTZ_POINT_DEL_CONTROL,       //清除预置点
    EXTPTZ_POINT_MOVE_CONTROL,      //转预置点
    EXTPTZ_STARTPANCRUISE,          //开始水平旋转
    EXTPTZ_STOPPANCRUISE,           //停止水平旋转
    EXTPTZ_SETLEFTBORDER,           //设置左边界
    EXTPTZ_SETRIGHTBORDER,          //设置右边界
    EXTPTZ_STARTLINESCAN,           //自动扫描开始
    EXTPTZ_CLOSELINESCAN,           //自动扫描开停止
    EXTPTZ_ADDTOLOOP,               //加入预置点到巡航  p1巡航线路  p2预置点值
    EXTPTZ_DELFROMLOOP,             //删除巡航中预置点  p1巡航线路  p2预置点值
    EXTPTZ_POINT_LOOP_CONTROL,      //开始巡航
    EXTPTZ_POINT_STOP_LOOP_CONTROL,//停止巡航
    EXTPTZ_CLOSELOOP,               //清除巡航  p1巡航线路
    EXTPTZ_FASTGOTO,                //快速定位
    EXTPTZ_AUXIOPEN,              //辅助开关，关闭在子命令中
```

```
    EXTPTZ_OPERATION_MENU,              //球机菜单操作，其中包括开，关，确定等等
    EXTPTZ_REVERSECOMM,             //镜头翻转
    EXTPTZ_OPERATION_RESET,         ///< 云台复位

    EXTPTZ_TOTAL,
};
```

错误类型代号，用于 *GetLastError* 函数的返回

```
typedef enum SDK_RET_CODE
{
    H264_DVR_NOERROR                = 0,                //没有错误
    H264_DVR_SUCCESS                = 1,                //返回成功
    H264_DVR_SDK_NOTVALID           = -10000,           //非法请求
    H264_DVR_NO_INIT                = -10001,           //SDK未经初始化
    H264_DVR_ILLEGAL_PARAM          = -10002,       //用户参数不合法
    H264_DVR_INVALID_HANDLE         = -10003,       //句柄无效
    H264_DVR_SDK_UNINIT_ERROR       = -10004,       //SDK清理出错
    H264_DVR_SDK_TIMEOUT                = -10005,   //等待超时
    H264_DVR_SDK_MEMORY_ERROR       = -10006,       //内存错误，创建内存失败
    H264_DVR_SDK_NET_ERROR          = -10007,       //网络错误
    H264_DVR_SDK_OPEN_FILE_ERROR        = -10008,   //打开文件失败
    H264_DVR_SDK_UNKNOWNERROR       = -10009,       //未知错误
    H264_DVR_DEV_VER_NOMATCH            = -11000,   //收到数据不正确，可能版本
不匹配
    H264_DVR_SDK_NOTSUPPORT         = -11001,       //版本不支持

    H264_DVR_OPEN_CHANNEL_ERROR         = -11200,       //打开通道失败
    H264_DVR_CLOSE_CHANNEL_ERROR        = -11201,       //关闭通道失败
    H264_DVR_SUB_CONNECT_ERROR      = -11202,       //建立媒体子连接失败
    H264_DVR_SUB_CONNECT_SEND_ERROR     = -11203,       //媒体子连接通讯失败

    /// 用户管理部分错误码
    H264_DVR_NOPOWER                = -11300,       //无权限
    H264_DVR_PASSWORD_NOT_VALID         = -11301,   // 账号密码不对
    H264_DVR_LOGIN_USER_NOEXIST         = -11302,   //用户不存在
    H264_DVR_USER_LOCKED                = -11303,   // 该用户被锁定
    H264_DVR_USER_IN_BLACKLIST      = -11304,       // 该用户不允许访问(在黑名
单中)
    H264_DVR_USER_HAS_USED          = -11305,       // 该用户以登陆
    H264_DVR_USER_NOT_LOGIN         = -11306,       // 该用户没有登陆
    H264_DVR_CONNECT_DEVICE_ERROR       = -11307,   //可能设备不存在
    H264_DVR_ACCOUNT_INPUT_NOT_VALID = -11308,      //用户管理输入不合法
    H264_DVR_ACCOUNT_OVERLAP            = -11309,   //索引重复
```

```
    H264_DVR_ACCOUNT_OBJECT_NONE      = -11310,     //不存在对象，用于查询时
    H264_DVR_ACCOUNT_OBJECT_NOT_VALID= -11311,     //不存在对象
    H264_DVR_ACCOUNT_OBJECT_IN_USE    = -11312,     //对象正在使用
    H264_DVR_ACCOUNT_SUBSET_OVERLAP   = -11313,     //子集超范围(如组的权限超
过权限表，用户权限超出组的权限范围等等)
    H264_DVR_ACCOUNT_PWD_NOT_VALID    = -11314,     //密码不正确
    H264_DVR_ACCOUNT_PWD_NOT_MATCH    = -11315,     //密码不匹配
    H264_DVR_ACCOUNT_RESERVED     = -11316,         //保留帐号

/// 配置管理相关错误码
    H264_DVR_OPT_RESTART              = -11400,    // 保存配置后需要重启应用程
序
    H264_DVR_OPT_REBOOT           = -11401,         // 需要重启系统
    H264_DVR_OPT_FILE_ERROR       = -11402,        // 写文件出错
    H264_DVR_OPT_CAPS_ERROR       = -11403,        // 配置特性不支持
    H264_DVR_OPT_VALIDATE_ERROR       = -11404,        // 配置校验失败
    H264_DVR_OPT_CONFIG_NOT_EXIST = -11405,        // 请求或者设置的配置不存
在

    ///
    H264_DVR_CTRL_PAUSE_ERROR     = -11500,        //暂停失败
    H264_DVR_SDK_NOTFOUND         = -11501,        //查找失败，没有找到对应文
件
    H264_DVR_CFG_NOT_ENABLE         = -11502,            //配置未启用
    H264_DVR_DECORD_FAIL            = -11503,            //配置未启用

    //DNS协议解析返回错误码
    H264_DVR_SOCKET_ERROR             = -11600,          //创建套节字失败
    H264_DVR_SOCKET_CONNECT           = -11601,          //连接套节字失败
    H264_DVR_SOCKET_DOMAIN            = -11602,          //域名解析失败
    H264_DVR_SOCKET_SEND              = -11603,          //发送数据失败

};
```

报警事件类型

```
enum SDK_EventCodeTypes

{

    SDK_EVENT_CODE_INIT = 0,

    SDK_EVENT_CODE_LOCAL_ALARM = 1,   //本地报警
```

```
    SDK_EVENT_CODE_NET_ALARM,      //网络报警
    SDK_EVENT_CODE_MANUAL_ALARM,   //手动报警
    SDK_EVENT_CODE_VIDEO_MOTION,   //动态检测
    SDK_EVENT_CODE_VIDEO_LOSS,     //视频丢失
    SDK_EVENT_CODE_VIDEO_BLIND,        //视频遮挡
    SDK_EVENT_CODE_VIDEO_TITLE,
    SDK_EVENT_CODE_VIDEO_SPLIT,
    SDK_EVENT_CODE_VIDEO_TOUR,
    SDK_EVENT_CODE_STORAGE_NOT_EXIST,
    SDK_EVENT_CODE_STORAGE_FAILURE,
    SDK_EVENT_CODE_LOW_SPACE,
    SDK_EVENT_CODE_NET_ABORT,
    SDK_EVENT_CODE_COMM,
    SDK_EVENT_CODE_STORAGE_READ_ERROR,
    SDK_EVENT_CODE_STORAGE_WRITE_ERROR,
    SDK_EVENT_CODE_NET_IPCONFLICT,
    SDK_EVENT_CODE_ALARM_EMERGENCY,
    SDK_EVENT_CODE_DEC_CONNECT,
    SDK_EVENT_CODE_VideoAnalyze=25,
    SDK_EVENT_CODE_NR,
};
//报警信息
typedef struct SDK_ALARM_INFO
{
    int nChannel;
    int iEvent;
    int iStatus;
    SDK_SYSTEM_TIME SysTime;
}SDK_AlarmInfo;
```

2.1.1 常量定义

```
#define PAN_AUTO        29  /* 云台以SS的速度左右自动扫描*/
```

```
#define EXTPTZ_FASTGOTO30   //三维定位


#define NET_MAX_CHANNUM              32          //最大通道个数
#define NET_DECORDR_CH          16      //最大解码通道个数
#define NET_MAX_USER_NUM             128         //最多用户数
#define NET_MAX_RIGTH_NUM            128         //最多权限数
#define NET_MAX_GROUP_NUM            50          //最多组数
#define NET_MAX_USER_LENGTH              32          //用户名密码最大长度
#define NET_MAX_COMBINE_NUM              2           //最大组合编码通道数
#define NET_MAX_DECORDR_CH       16      //最大解码通道个数

#define NET_MAX_DDNS_TYPE            5           //支持的DDNS种类
#define NET_MAX_ARSP_TYPE            5
#define NET_MAX_ALARMSERVER_TYPE    5       //支持报警中心种类
#define NET_MAX_SYSFUNC                  20          //最多系统功能个数
#define NET_MAX_PTZ_PROTOCOL_LENGTH 32               //云台协议名称最大长度
#define NET_N_WEEKS                  7           //星期数
#define  NET_N_TSECT                 6           //时间段数
#define NET_MD_REGION_ROW            32          //动态检测区域行数
#define NET_COVERNUM                 8            //覆盖区域数
#define NET_MAX_FILTERIP_NUM         64          //IP地址最大过滤数
#define NET_NAME_PASSWORD_LEN        64          //用户名密码最大长度
#define NET_MAX_PATH_LENGTH          260         //路径长度
#define NET_N_MIN_TSECT              2
#define NET_MAX_RETURNED_LOGLIST        128         //最多日志条数
#define NET_MAX_MAC_LEN          32      //MAC地址字符最大长度
#define NET_IW_ENCODING_TOKEN_MAX   128
#define NET_MAX_AP_NUMBER          10
#define NET_MAX_INFO_LEN             128
#define NET_MAX_USERNAME_LENGTH          128
#define NET_MAX_SERIALNO_LENGTH          128      //最大解码通道个数

#define NET_CAPTURE_SIZE_NUM 9

//DDNS参数
#define DDNS_MAX_DDNS_NAMELEN 64  //主机名长度
#define DDNS_MAX_DDNS_PWDLEN 32   //密码长度
#define DDNS_MAX_DDNS_IPSIZE 64   //IP地址长度

//摄象机参数
#define CAMERAPARA_MAXNUM  16    //曝光能力中目前最大长度
```

```
//DDNS服务器设备的信息
typedef struct _DDNS_INFO
{
	std::string ID;	//设备标识
	char IP[DDNS_MAX_DDNS_IPSIZE];	//内网IP
	int WebPort; //Web端口,默认为80
	int MediaPort; //媒体端口,默认为34567
	int MobilePort;  //手机监控端口，默认为34599
	int UPNPWebPort;  //UPNP启动下Web端口,UPNP不开启为0
	int UPNPMediaPort; //UPNP启动下媒体端口,UPNP不开启为0
	int UPNPMobilePort; //UPNP启动下手机监控端口,UPNP不开启为0
	char Username[DDNS_MAX_DDNS_NAMELEN]; //用户名
	char Password[DDNS_MAX_DDNS_PWDLEN]; //密码
}DDNS_INFO,*pDDNS_INFO;


//实时预览扩展接口增加的参数：预览类型
typedef enum _H264_DVR_RealPlayType
{
	NET_RType_Realplay = 0,	//实时预览
	NET_RType_Main_Realplay2,	//实时监视-主码流，等同于NET_RType_Realplay
	NET_RType_Main_Realplay3,
	NET_RType_Sub_Realplay_1,	//实时监视-从码流
	NET_RType_Sub_Realplay_2,	//实时监视-从码流
	NET_RType_Sub_Realplay_3,	//实时监视-从码流

	NET_RType_Multiplay_1,	//多画面预览－画面
	NET_RType_Multiplay_4,	//多画面预览－画面
	NET_RType_Multiplay_6,	//多画面预览－画面
	NET_RType_Multiplay_8,	//多画面预览－画面
	NET_RType_Multiplay_9,	//多画面预览－画面
	NET_RType_Multiplay_12,	//多画面预览－画面
	NET_RType_Multiplay_16,	//多画面预览－画面
} NET_RealPlayType;

/// 组合编码模式
enum NetCombinType
{
	NET_COMBIN_NONE,
	NET_COMBIN_1,
	NET_COMBIN_2,
	NET_COMBIN_3,
	NET_COMBIN_4,
```

```
    NET_COMBIN_5,
    NET_COMBIN_6,
    NET_COMBIN_7,
    NET_COMBIN_8,
    NET_COMBIN_9,
    NET_COMBIN_10,
    NET_COMBIN_11,
    NET_COMBIN_12,
    NET_COMBIN_13,
    NET_COMBIN_14,
    NET_COMBIN_15,
    NET_COMBIN_16,
    NET_COMBIN_1_4,
    NET_COMBIN_5_8,
    NET_COMBIN_9_12,
    NET_COMBIN_13_16,
    NET_COMBIN_1_8,
    NET_COMBIN_9_16,
    NET_COMBIN_1_9,
    NET_COMBIN_8_16,
    NET_COMBIN_1_16
};

//这些结构体和枚举是提供给外部使用，所有可能会和设备那边定义了次,所以都在前面加了SDK_
enum SDK_CAPTURE_SIZE_t {
    SDK_CAPTURE_SIZE_D1,        ///< 720*576(PAL) 720*480(NTSC)
    SDK_CAPTURE_SIZE_HD1,       ///< 352*576(PAL) 352*480(NTSC)
    SDK_CAPTURE_SIZE_BCIF,      ///< 720*288(PAL) 720*240(NTSC)
    SDK_CAPTURE_SIZE_CIF,       ///< 352*288(PAL) 352*240(NTSC)
    SDK_CAPTURE_SIZE_QCIF,      ///< 176*144(PAL) 176*120(NTSC)
    SDK_CAPTURE_SIZE_VGA,       ///< 640*480(PAL) 640*480(NTSC)
    SDK_CAPTURE_SIZE_QVGA,      ///< 320*240(PAL) 320*240(NTSC)
    SDK_CAPTURE_SIZE_SVCD,      ///< 480*480(PAL) 480*480(NTSC)
    SDK_CAPTURE_SIZE_QQVGA,         ///< 160*128(PAL) 160*128(NTSC)
    SDK_CAPTURE_SIZE_NR             ///< 枚举的图形大小种类的数目。
};

/// 捕获码流控制模式类型
enum SDK_capture_brc_t {
    SDK_CAPTURE_BRC_CBR,            ///< 固定码流。
    SDK_CAPTURE_BRC_VBR,            ///< 可变码流。
    SDK_CAPTURE_BRC_MBR,            ///< 混合码流。
    SDK_CAPTURE_BRC_NR                  ///< 枚举的码流控制模式数目。
```

```
};

/// 捕获压缩格式类型
enum SDK_CAPTURE_COMP_t {
    SDK_CAPTURE_COMP_DIVX_MPEG4,  ///< DIVX MPEG4。
    SDK_CAPTURE_COMP_MS_MPEG4,    ///< MS MPEG4。
    SDK_CAPTURE_COMP_MPEG2,       ///< MPEG2。
    SDK_CAPTURE_COMP_MPEG1,       ///< MPEG1。
    SDK_CAPTURE_COMP_H263,        ///< H.263
    SDK_CAPTURE_COMP_MJPG,        ///< MJPG
    SDK_CAPTURE_COMP_FCC_MPEG4,   ///< FCC MPEG4
    SDK_CAPTURE_COMP_H264,        ///< H.264
    SDK_CAPTURE_COMP_NR           ///< 枚举的压缩标准数目。
};

/// 捕获通道类型
enum SDK_CAPTURE_CHANNEL_t {
    SDK_CHL_MAIN_T = 0,     ///< 主通道    -   主码流
    SDK_CHL_2END_T = 1,     ///< 辅通道    -   出辅码流
    SDK_CHL_3IRD_T = 2,     ///< 辅通道    -   出辅码流
    SDK_CHL_4RTH_T = 3,     ///< 辅通道    -   出辅码流
    SDK_CHL_JPEG_T = 4,     ///< 辅通道    -   出JPEG抓图
    SDK_CHL_FUNCTION_NUM
};
/// 音频编码类型
enum SDK_AudioEncodeTypes
{
    SDK_AUDIO_ENCODE_NONE = 0,
    SDK_AUDIO_ENCODE_G729_8KBIT,
    SDK_AUDIO_ENCODE_G726_16KBIT,
    SDK_AUDIO_ENCODE_G726_24KBIT,
    SDK_AUDIO_ENCODE_G726_32KBIT,
    SDK_AUDIO_ENCODE_G726_40KBIT,
    SDK_AUDIO_ENCODE_PCM_8TO16BIT,
    SDK_AUDIO_ENCODE_PCM_ALAW,
    SDK_AUDIO_ENCODE_PCM_ULAW,
    SDK_AUDIO_ENCODE_ADPCM8K16BIT,
    SDK_AUDIO_ENCODE_ADPCM16K16BIT,
    SDK_AUDIO_ENCODE_G711_ALAW,
    SDK_AUDIO_ENCODE_MPEG2_LAYER1,
    SDK_AUDIO_ENCODE_AMR8K16BIT,
    SDK_AUDIO_ENCODE_G711_ULAW,
    SDK_AUDIO_ENCODE_IMA_ADPCM_8K16BIT,
```

```
    SDK_AUDIO_ENCODE_TYPES_NR,
};

/// 报警事件码
enum SDK_EventCodeTypes
{
    SDK_EVENT_CODE_INIT = 0,
    SDK_EVENT_CODE_LOCAL_ALARM = 1,    //本地报警
    SDK_EVENT_CODE_NET_ALARM,       //网络报警
    SDK_EVENT_CODE_MANUAL_ALARM, //手动报警
    SDK_EVENT_CODE_VIDEO_MOTION,  //动态检测
    SDK_EVENT_CODE_VIDEO_LOSS,      //视频丢失
    SDK_EVENT_CODE_VIDEO_BLIND,          //视频遮挡
    SDK_EVENT_CODE_VIDEO_TITLE,
    SDK_EVENT_CODE_VIDEO_SPLIT,
    SDK_EVENT_CODE_VIDEO_TOUR,
    SDK_EVENT_CODE_STORAGE_NOT_EXIST,
    SDK_EVENT_CODE_STORAGE_FAILURE,
    SDK_EVENT_CODE_LOW_SPACE,
    SDK_EVENT_CODE_NET_ABORT,
    SDK_EVENT_CODE_COMM,
    SDK_EVENT_CODE_STORAGE_READ_ERROR,
    SDK_EVENT_CODE_STORAGE_WRITE_ERROR,
    SDK_EVENT_CODE_NET_IPCONFLICT,
    SDK_EVENT_CODE_ALARM_EMERGENCY,
    SDK_EVENT_CODE_DEC_CONNECT,
    SDK_EVENT_CODE_NR,
};
//! 编码配置的类型
enum SDK_ENCODE_TYPE_BY_RECORD
{
    SDK_ENCODE_TYPE_TIM = 0,
    SDK_ENCODE_TYPE_MTD = 1,
    SDK_ENCODE_TYPE_ALM = 2,
    SDK_ENCODE_TYPE_NUM = 3,
    SDK_ENCODE_TYPE_SNAP_TIMER = 0,
    SDK_ENCODE_TYPE_SNAP_TRIGGER = 1,
};
#define SDK_EXTRATYPES 3 //辅码流类型

//网络传输策略
enum SDK_TransferPolicy
```

```
{
    SDK_TRANSFER_POLICY_AUTO,       ///< 自适应
    SDK_TRANSFER_POLICY_QUALITY,    ///< 质量优先
    SDK_TRANSFER_POLICY_FLUENCY,    ///< 流量优先
    SDK_TRANSFER_POLICY_NR,
};
/// 编码功能
enum SDK_EncodeFunctionTypes
{
    SDK_ENCODE_FUNCTION_TYPE_DOUBLE_STREAM,       ///< 双码流功能
    SDK_ENCODE_FUNCTION_TYPE_COMBINE_STREAM,///< 组合编码功能
    SDK_ENCODE_FUNCTION_TYPE_SNAP_STREAM,         ///< 抓图功能
    SDK_ENCODE_FUNCTION_TYPE_NR,
};
/// 报警功能
enum SDK_AlarmFucntionTypes
{
    SDK_ALARM_FUNCTION_TYPE_MOTION_DETECT,    ///< 动态检测
    SDK_ALARM_FUNCTION_TYPE_BLIND_DETECT, ///< 视屏遮挡
    SDK_ALARM_FUNCTION_TYPE_LOSS_DETECT,  ///< 视屏丢失
    SDK_ALARM_FUNCTION_TYPE_LOCAL_ALARM,  ///< 本地报警
    SDK_ALARM_FUNCTION_TYPE_NET_ALARM,        ///< 网络报警
    SDK_ALARM_FUNCTION_TYPE_IP_CONFLICT,    ///< IP地址冲突
    SDK_ALARM_FUNCTION_TYPE_NET_ABORT,        ///< 网络异常
    SDK_ALARM_FUNCTION_TYPE_STORAGE_NOTEXIST,     ///< 存储设备不存在
    SDK_ALARM_FUNCTION_TYPE_STORAGE_LOWSPACE,     ///< 存储设备容量不足
    SDK_ALARM_FUNCTION_TYPE_STORAGE_FAILURE,  ///< 存储设备访问失败
    SDK_ALARM_FUNCTION_TYPE_NR
};
/// 网络服务功能
enum SDK_NetServerTypes
{
    SDK_NET_SERVER_TYPES_IPFILTER,        ///< 白黑名单
    SDK_NET_SERVER_TYPES_DHCP,            ///< DHCP功能
    SDK_NET_SERVER_TYPES_DDNS,            ///< DDNS功能
    SDK_NET_SERVER_TYPES_EMAIL,               ///< Email功能
    SDK_NET_SERVER_TYPES_MULTICAST,           ///< 多播功能
    SDK_NET_SERVER_TYPES_NTP,             ///< NTP功能
    SDK_NET_SERVER_TYPES_PPPOE,
    SDK_NET_SERVER_TYPES_DNS,
    SDK_NET_SERVER_TYPES_ARSP,            ///< 主动注册服务
    SDK_NET_SERVER_TYPES_3G,          ///< 3G拨号
    SDK_NET_SERVER_TYPES_MOBILE,      ///< 手机监控
```

```
    SDK_NET_SERVER_TYPES_UPNP,                  ///< UPNP
    SDK_NET_SERVER_TYPES_FTP,                   ///< FTP
    SDK_NET_SERVER_TYPES_WIFI,          ///<WIFI
    SDK_NET_SERVER_TYPES_ALARM_CENTER,  ///< 告警中心
    SDK_NET_SERVER_TYPES_NETPLAT_MEGA,  ///< 互信互通
    SDK_NET_SERVER_TYPES_NETPLAT_XINWANG,  ///< 星望
    SDK_NET_SERVER_TYPES_NETPLAT_SHISOU,  ///< 视搜
    SDK_NET_SERVER_TYPES_NETPLAT_VVEYE,  ///< 威威眼
    SDK_NET_SERVER_TYPES_RTSP,     //RTSP
    SDK_NET_SERVER_TYPES_NR,
};
/// 预览功能
enum SDK_PreviewTypes
{
    SDK_PREVIEW_TYPES_TOUR,             ///< 轮巡
    SDK_PREVIEW_TYPES_TALK,       ///< GUI界面配置
    SDK_PREVIEW_TYPES_NR
};
/// 存储设备控制类型
enum SDK_StorageDeviceControlTypes
{
    SDK_STORAGE_DEVICE_CONTROL_SETTYPE,         ///< 设置类型
    SDK_STORAGE_DEVICE_CONTROL_RECOVER,         ///< 恢复错误
    SDK_STORAGE_DEVICE_CONTROL_PARTITIONS,      ///< 分区操作
    SDK_STORAGE_DEVICE_CONTROL_CLEAR,           ///< 清除操作
    SDK_STORAGE_DEVICE_CONTROL_NR,
};

/// 清除磁盘数据类型
enum SDK_StorageDeviceClearTypes
{
    SDK_STORAGE_DEVICE_CLEAR_DATA,              ///< 清除录像数据
    SDK_STORAGE_DEVICE_CLEAR_PARTITIONS,    ///< 清除分区
    SDK_STORAGE_DEVICE_CLEAR_NR,
};

/// 驱动器类型
enum SDK_FileSystemDriverTypes
{
    SDK_DRIVER_READ_WRITE  = 0, ///< 读写驱动器
    SDK_DRIVER_READ_ONLY   = 1, ///< 只读驱动器
    SDK_DRIVER_EVENTS      = 2, ///< 事件驱动器
    SDK_DRIVER_REDUNDANT   = 3, ///< 冗余驱动器
```

```
    SDK_DRIVER_SNAPSHOT          = 4, ///< 快照驱动器
    SDK_DRIVER_TYPE_NR       = 5, ///< 驱动器类型个数
    SDK_DRIVER_UNUSED        = 0xff,    ///< 没有使用的驱动器结构
};
/// 默认配置种类
enum SDK_DefaultConfigKinds
{
    SDK_DEFAULT_CFG_GENERAL,              // 普通配置
    SDK_DEFAULT_CFG_ENCODE,                   // 编码配置
    SDK_DEFAULT_CFG_RECORD,                   // 录像配置
    SDK_DEFAULT_CFG_NET_SERVICE,          // 网络服务
    SDK_DEFAULT_CFG_NET_COMMON,               // 通用网络
    SDK_DEFAULT_CFG_ALARM,                // 报警
    SDK_DEFAULT_CFG_PTZCOMM,              // 云台，串口
    SDK_DEFAULT_CFG_USERMANAGER,          // 用户管理
    SDK_DEFAULT_CFG_PREVIEW,              // 预览配置
    SDK_DEFAULT_CFG_END,
};
enum SDK_File_Type
{
    SDK_RECORD_ALL = 0,
    SDK_RECORD_ALARM = 1, //外部报警录像
    SDK_RECORD_DETECT,  //视频侦测录像
    SDK_RECORD_REGULAR,      //普通录像
    SDK_RECORD_MANUAL,       //手动录像
    SDK_PIC_ALL = 10,
    SDK_PIC_ALARM,           //外部报警录像
    SDK_PIC_DETECT,          //视频侦测录像
    SDK_PIC_REGULAR,     //普通录像
    SDK_PIC_MANUAL,      //手动录像
    SDK_TYPE_NUM
};
```

2.1.2 设备信息结构

- 设备结构定义如下

```
typedef struct _H264_DVR_DEVICEINFO

{
    char sSoftWareVersion[64]; ///< 软件版本信息
    char sHardWareVersion[64]; ///< 硬件版本信息
```

```
    char sEncryptVersion[64];  ///< 加密版本信息
    SDK_SYSTEM_TIME tmBuildTime;///< 软件创建时间
    char sSerialNumber[64];        ///< 设备序列号
    int byChanNum;              ///< 视频输入通道数
    int iVideoOutChannel;       ///< 视频输出通道数
    int byAlarmInPortNum;       ///< 报警输入通道数
    int byAlarmOutPortNum;      ///< 报警输出通道数
    int iTalkInChannel;         ///< 对讲输入通道数
    int iTalkOutChannel;        ///< 对讲输出通道数
    int iExtraChannel;        ///< 扩展通道数
    int iAudioInChannel;       ///< 音频输入通道数
    int iCombineSwitch;         ///< 组合编码通道分割模式是否支持切换
    int iDigChannel;     ///<数字通道数
    unsigned int uiDeviceRunTime; ///<系统运行时间
}H264_DVR_DEVICEINFO,*LPH264_DVR_DEVICEINFO;
```

2.1.3 时间信息

///系统时间结构

```
typedef struct SDK_SYSTEM_TIME{
    int  year;///< 年。
    int  month;///< 月，January = 1, February = 2, and so on.
    int  day;///< 日。
    int  wday;///< 星期，Sunday = 0, Monday = 1, and so on
    int  hour;///< 时。
    int  minute;///< 分。
    int  second;///< 秒。
    int  isdst;///< 夏令时标识。
  }SDK_SYSTEM_TIME;
//录像设置相关结构体
typedef struct tagSDK_TIMESECTION
{
    //!使能
    int enable;
    //!开始时间:小时
    int startHour;
    //!开始时间:分钟
    int   startMinute;
    //!开始时间:秒钟
    int   startSecond;
    //!结束时间:小时
    int   endHour;
    //!结束时间:分钟
```

```
    int   endMinute;
    //!结束时间:秒钟
    int   endSecond;
}SDK_TIMESECTION;

typedef struct{
    int dwYear;         //年
    int dwMonth;  //月
    int dwDay;          //日
    int dwHour;         //时
    int dwMinute; //分
    int dwSecond;       //秒
}H264_DVR_TIME,*LPH264_DVR_TIME;

//时间结构
typedef struct _NEW_NET_TIME
{
    unsigned int second         :6;             //    秒    1-60
    unsigned int minute         :6;             //    分    1-60
    unsigned int hour       :5;             //    时    1-24
    unsigned int day        :5;             //    日    1-31
    unsigned int month          :4;             //    月    1-12
    unsigned int year       :6;             //    年    2000-2063
}NEW_NET_TIME, *LPNET_TIME;
///< 夏令时结构
struct DSTPoint
{
    int   iYear;
    int   iMonth;
    int   iWeek;        ///<周:first  to2 3 4 -1:last one   0:表示使用按日计算的方法[-1,4]
    int   iWeekDay;     ///<weekday from sunday=0      [0, 6]
    int Hour;
    int Minute;
};
```

2.1.4 录像文件信息

//查询条件结构体

```
typedef struct
{
    int nChannelN0;          //通道号
    int nFileType;           //文件类型, 见SDK_File_Type
```

```
    H264_DVR_TIME startTime;    //开始时间
    H264_DVR_TIME endTime;  //结束时间
    char szCard[32];        //卡号
    void *hWnd;         //输出窗口句柄（为NULL时候，网络数据与解码播放分开处理）
}H264_DVR_FINDINFO;
```

//返回录像信息结构体

```
typedef struct
{
    int ch;                     //通道号
    int size;                //文件大小
    char sFileName[108];        ///< 文件名
    SDK_SYSTEM_TIME stBeginTime;   ///< 文件开始时间
    SDK_SYSTEM_TIME stEndTime;     ///< 文件结束时间
    void *hWnd;         //输出窗口句柄（为NULL时候，网络数据与解码播放分开处理）

}H264_DVR_FILE_DATA;


    //按时间段查询
    struct SDK_SearchByTime
    {
        int nHighChannel;           ///< 33~64 录像通道号掩码
        int nLowChannel;         ///< 1~32 录像通道号掩码
        int nFileType;               ///< 文件类型，见 SDK_File_Type
        SDK_SYSTEM_TIME stBeginTime;       ///< 查询开始时间
        SDK_SYSTEM_TIME stEndTime;     ///< 查询结束时间
        int    iSync;                ///< 是否需要同步
    };


    //每个通道的录像信息
    struct SDK_SearchByTimeInfo
    {
        int iChannel;            ///< 录像通道号
        ///< 录像记录用 720 个字节的 5760 位来表示一天中的 1440 分钟
```

```
        ///<  0000: 无 录 像    0001:F_COMMON  0002:F_ALERT  0003:F_DYNAMIC
0004:F_CARD 0005:F_HAND
        unsigned char cRecordBitMap[720];
    };


    struct SDK_SearchByTimeResult
    {
        int nInfoNum;                          ///< 通道的录像记录信息个数
        SDK_SearchByTimeInfo ByTimeInfo[NET_MAX_CHANNUM]; ///< 通道的录像记
录信息
    };
```

串口协议信息

```
struct SDK_COMMATTRI
{
    int iDataBits;    // 数据位取值为,6,7,8
    int iStopBits;    // 停止位
    int iParity;  // 校验位
    int iBaudRate;    // 实际波特率
};
// 串口协议
enum SDK_CommProtocol
{
    SDK_CONSOLE = 0,
    SDK_KEYBOARD,
    SDK_COM_TYPES,
};
/// 串口协议
struct SDK_COMMFUNC
{
    //每个协议最多由个字符组成
    int nProNum;
    char vCommProtocol[100][32];
};
// 串口配置
struct SDK_CONFIG_COMM_X
{
    char iProtocolName[32]; // 串口协议:“Console”
```

```
    int iPortNo;     // 端口号
    SDK_COMMATTRI aCommAttri;      // 串口属性
};
```

云台协议

```
struct SDK_STR_CONFIG_PTZ
{
    char sProtocolName[NET_MAX_PTZ_PROTOCOL_LENGTH];   // 协议名称
    int ideviceNo;              // 云台设备地址编号
    int iNumberInMatrixs;       // 在矩阵中的统一编号
    int iPortNo;            // 串口端口号  [1, 4]
    SDK_COMMATTRI dstComm;          // 串口属性
};
/// 云台协议
struct SDK_PTZPROTOCOLFUNC
{
    //每个协议最多由个字符组成
    int nProNum;
    char vPTZProtocol[100][NET_MAX_PTZ_PROTOCOL_LENGTH];
};

//所有通道云台协议
struct SDK_STR_PTZCONFIG_ALL
{
    SDK_STR_CONFIG_PTZ ptzAll[NET_MAX_CHANNUM];
};
//RS485
struct SDK_STR_RS485CONFIG_ALL
{
    SDK_STR_CONFIG_PTZ ptzAll[NET_MAX_CHANNUM];
};
```

用户管理功能数据结构

权限列表

```
typedef struct _OPR_RIGHT
{
        char name[NET_MAX_USER_LENGTH];
}OPR_RIGHT;

typedef struct _USER_INFO
```

```
{
    int         rigthNum;
    char        rights[NET_MAX_RIGTH_NUM][NET_MAX_USER_LENGTH];
    char        Groupname[NET_MAX_USER_LENGTH];
    char        memo[NET_MAX_USER_LENGTH];
    char        name[NET_MAX_USER_LENGTH];
    char        passWord[NET_MAX_USER_LENGTH];
    bool        reserved;       //是否保留
    bool        shareable;          //本用户是否允许复用1-复用，-不复用
}USER_INFO;

typedef struct _USER_GROUP_INFO
{
    int         rigthNum;
    char        memo[NET_MAX_USER_LENGTH];
    char        name[NET_MAX_USER_LENGTH];
    char        rights[NET_MAX_RIGTH_NUM][NET_MAX_USER_LENGTH];     //权限列表
}USER_GROUP_INFO;

//用户信息配置结构
typedef struct _USER_MANAGE_INFO
{
    int                 rigthNum;
    OPR_RIGHT           rightList[NET_MAX_RIGTH_NUM];
    int                 groupNum;
    USER_GROUP_INFO     groupList[NET_MAX_GROUP_NUM];
    int                 userNum;
    USER_INFO           userList[NET_MAX_USER_NUM];
}USER_MANAGE_INFO;

//修改用户
typedef struct _CONF_MODIFYUSER
{
    char UserName[NET_MAX_USER_LENGTH];
    USER_INFO User;
}CONF_MODIFYUSER;

//修改组
typedef struct _CONF_MODIFYGROUP
{
    char GroupName[NET_MAX_USER_LENGTH];
    USER_GROUP_INFO Group;
}CONF_MODIFYGROUP;
```

```
/// 修改用户密码请求
struct _CONF_MODIFY_PSW
{
    char sUserName[NET_MAX_USER_LENGTH];
    char Password[NET_MAX_USER_LENGTH];
    char NewPassword[NET_MAX_USER_LENGTH];
};
```

日志信息

```
#define NET_MAX_RETURNED_LOGLIST 1024       //最多日志条数
/// 日志查询条件
struct SDK_LogSearchCondition
{
    int nType;     ///< 日志类型
    int iLogPosition;           ///< 从上次查询的结束时的日志指针
    SDK_SYSTEM_TIME stBeginTime;   ///< 查询日志开始时间
    SDK_SYSTEM_TIME stEndTime;     ///< 查询日志结束时间
};
struct SDK_LogItem
{
    char sType[24];    ///< 日志类型
    char sUser[32];    ///< 日志用户
    char sData[68];    ///< 日志数据
    SDK_SYSTEM_TIME stLogTime;   ///< 日志时间
    int iLogPosition;              ///< 从上次查询的结束时的日志指针
};
//日志返回信息
struct SDK_LogList
{
    int iNumLog;
    SDK_LogItem Logs[NET_MAX_RETURNED_LOGLIST];
};;
```

- 查询硬盘信息的返回数据结构

```
enum
{
    SDK_MAX_DRIVER_PER_DISK = 2,    ///< 每个磁盘最多的分区数
    SDK_MAX_DISK_PER_MACHINE = 8,   ///< 最多支持块硬盘
};
struct SDK_STORAGEDISK
{
    int    iPhysicalNo;
    int    iPartNumber;     // 分区数
    SDK_DriverInformation diPartitions[SDK_MAX_DRIVER_PER_DISK];
};
struct SDK_StorageDeviceInformationAll
{
    int iDiskNumber;
    SDK_STORAGEDISK vStorageDeviceInfoAll[SDK_MAX_DISK_PER_MACHINE];
};
```

- 网络监视

```
typedef struct{
    int nChannel; //通道号
    int nStream;  //0表示主码流，为表示子码流
    int nMode;    //0：TCP方式,1：UDP方式,2：多播方式,3 - RTP方式，-音视频分开(TCP)
    int nComType; //只对组合编码通道有效， 组合编码通道的拼图模式
    void* hWnd;   //预览输出窗口句柄（为NULL时候，网络数据与解码播放分开处理）
}H264_DVR_CLIENTINFO,*LPH264_DVR_CLIENTINFO;
```

2.1.5 配置信息结构

H264_DVR_GetDevConfig、H264_DVR_SetDevConfig 的命令定义

```
typedef enum _SDK_CONFIG_TYPE
{
    E_SDK_CONFIG_NOTHING = 0,
    //用户管理部分
    E_SDK_CONFIG_USER,              //用户信息，包含了权限列表，用户列表和组列表
                                        USER_MANAGE_INFO
    E_SDK_CONFIG_ADD_USER,      //增加用户 USER_INFO
    E_SDK_CONFIG_MODIFY_USER,   //修改用户   CONF_MODIFYUSER
    E_SDK_CONFIG_DELETE_USER,   //删除用户   //同增加用户
```

```
E_SDK_CONFIG_ADD_GROUP,     //增加组      USER_GROUP_INFO
E_SDK_CONFIG_MODIFY_GROUP, //修改组      CONF_MODIFYGROUP
E_SDK_COFIG_DELETE_GROUP,   //删除组      //同增加组
E_SDK_CONFIG_MODIFY_PSW,    //修改密码    _CONF_MODIFY_PSW


//能力集部分
E_SDK_CONFIG_ABILITY_SYSFUNC = 9,//支持的网络功能      SDK_SystemFunctio
E_SDK_CONFIG_ABILTY_ENCODE,        //首先获得编码能力    CONFIG_EncodeAbility
E_SDK_CONFIG_ABILITY_PTZPRO,       //云台协议            SDK_PTZPROTOCOLFUNC
E_SDK_COMFIG_ABILITY_COMMPRO,      //串口协议            SDK_COMMFUNC
E_SDK_CONFIG_ABILITY_MOTION_FUNC, //动态检测块          SDK_MotionDetectFunction
E_SDK_CONFIG_ABILITY_BLIND_FUNC,  //视频遮挡块        SDK_BlindDetectFunction
E_SDK_CONFIG_ABILITY_DDNS_SERVER, //DDNS服务支持类型SDK_DDNSServiceFunction
E_SDK_CONFIG_ABILITY_TALK,         //对讲编码类型

//配置部分
E_SDK_CONFIG_SYSINFO = 17,      //系统信息          H264_DVR_DEVICEINFO
E_SDK_CONFIG_SYSNORMAL,         //普通配置          SDK_CONFIG_NORMAL
E_SDK_CONFIG_SYSENCODE,         //编码配置          SDK_EncodeConfigAll
E_SDK_CONFIG_SYSNET,            //网络设置          SDK_CONFIG_NET_COMMON
E_SDK_CONFIG_PTZ,               //云台页面        SDK_STR_PTZCONFIG_ALL
E_SDK_CONFIG_COMM,              //串口页面        SDK_CommConfigAll
E_SDK_CONFIG_RECORD,            //录像设置界面  SDK_RECORDCONFIG_ALL
E_SDK_CONFIG_MOTION,            //动态检测页面   SDK_MOTIONCONFIG
E_SDK_CONFIG_SHELTER,       //视频遮挡          SDK_BLINDDETECTCONFIG_ALL
E_SDK_CONFIG_VIDEO_LOSS,    //视频丢失,         SDK_VIDEOLOSSCONFIG_ALL
E_SDK_CONFIG_ALARM_IN,      //报警输入          SDK_ALARM_INPUTCONFIG_ALL
E_SDK_CONFIG_ALARM_OUT,     //报警输出
E_SDK_CONFIG_DISK_MANAGER  //硬盘管理界面
E_SDK_CONFIG_OUT_MODE,      //输出模式界面
E_SDK_CONFIG_AUTO,          //自动维护界面配置  SDK_AutoMaintainConfig
E_SDK_CONFIG_DEFAULT,       //恢复默认界面配置
E_SDK_CONFIG_DISK_INFO,     //硬盘信息          SDK_StorageDeviceInformationAll
E_SDK_CONFIG_LOG_INFO,      //查询日志          SDK_LogList
E_SDK_CONFIG_NET_IPFILTER, //网络部分：黑/白名单   SDK_NetIPFilterConfig
E_SDK_CONFIG_NET_DHCP,      //网络部分：DHCP
E_SDK_CONFIG_NET_DDNS,      //网络部分：DDNS        SDK_NetDDNSConfigALL
E_SDK_CONFIG_NET_EMAIL,     //网络部分：EMAIL       SDK_NetEmailConfig
E_SDK_CONFIG_NET_MULTICAST, //网络部分:组播         SDK_NetMultiCastConfig
E_SDK_CONFIG_NET_NTP,       //网络部分： NTP        SDK_NetNTPConfig
E_SDK_CONFIG_NET_PPPOE,     //网络部分:  PPPOE        SDK_NetPPPoEConfig
E_SDK_CONFIG_NET_DNS,       //网络部分:  DNS        SDK_NetDNSConfig
```

```
E_SDK_CONFIG_NET_FTPSERVER, //网络部分： FTP        SDK_FtpServerConfig
E_SDK_CONFIG_SYS_TIME, //系统时间
E_SDK_CONFIG_CLEAR_LOG,      //清除日志
E_SDK_REBOOT_DEV,            //重启启动设备

E_SDK_CONFIG_ABILITY_LANG,          //支持语言
E_SDK_CONFIG_VIDEO_FORMAT,          //视频制式
E_SDK_CONFIG_COMBINEENCODE,             //组合编码
E_SDK_CONFIG_EXPORT,                //配置导出
E_SDK_CONFIG_IMPORT,                //配置导入
E_SDK_LOG_EXPORT,                   //日志导出
E_SDK_CONFIG_COMBINEENCODEMODE,     //组合编码模式
E_SDK_WORK_STATE,                   //运行状态
E_SDK_ABILITY_LANGLIST, //实际支持的语言集
E_SDK_CONFIG_NET_ARSP,
E_SDK_CONFIG_SNAP_STORAGE,
E_SDK_CONFIG_NET_3G, //3G拨号
E_SDK_CONFIG_NET_MOBILE, //手机监控
E_SDK_CONFIG_UPGRADEINFO, //获取升级信息
E_SDK_CONFIG_NET_DECODER,
E_SDK_ABILITY_VSTD, //实际支持的视频制式
E_SDK_CONFIG_ABILITY_VSTD,   //支持视频制式
E_SDK_CONFIG_NET_UPNP, //UPUN设置
E_SDK_CONFIG_NET_WIFI,
E_SDK_CONFIG_NET_WIFI_AP_LIST,
E_SDK_CONFIG_SYSENCODE_SIMPLIIFY, //简化的编码配置
E_SDK_CONFIG_ALARM_CENTER,  //告警中心
E_SDK_CONFIG_NET_ALARM,
E_SDK_CONFIG_NET_MEGA,    //互信互通
E_SDK_CONFIG_NET_XINGWANG, //星望
E_SDK_CONFIG_NET_SHISOU,   //视搜
E_SDK_CONFIG_NET_VVEYE,    //VVEYE
E_SDK_VIDEO_PREVIEW,
E_SDK_CONFIG_NET_DECODER_V2,
E_SDK_CONFIG_NET_DECODER_V3,//数字通道
E_SDK_CONFIG_ABILITY_SERIALNO,   // 序列号
E_SDK_CONFIG_NET_RTSP,    //RTSP
E_SDK_GUISET,             //GUISET
E_SDK_CATCHPIC,               //抓图
E_SDK_VIDEOCOLOR,           //视频颜色设置
E_SDK_CONFIG_COMM485,
E_SDK_COMFIG_ABILITY_COMMPRO485, //串口485
E_SDK_CONFIG_SYS_TIME_NORTC,    //系统时间noRtc
```

E_SDK_CONFIG_REMOTECHANNEL, //远程通道
E_SDK_CONFIG_OPENTRANSCOMCHANNEL , //打开透明串口
E_SDK_CONFIG_CLOSETRANSCOMCHANNEL, //关闭透明串口
E_SDK_CONFIG_SERIALWIRTE, //写入透明串口信息
E_SDK_CONFIG_SERIALREAD, //读取透明串口信息
E_SDK_CONFIG_CHANNELTILE_DOT //点阵信息
E_SDK_CONFIG_CAMERA, //摄象机参数
E_SDK_CONFIG_ABILITY_CAMERA , //曝光能力级
E_SDK_CONFIG_BUGINFO, //命令调试
E_SDK_CONFIG_STORAGENOTEXIST,//硬盘不存在
E_SDK_CONFIG_STORAGELOWSPACE, //硬盘容量不足
E_SDK_CONFIG_STORAGEFAILURE, //硬盘出错
E_SDK_CFG_NETIPCONFLICT, //IP冲突
E_SDK_CFG_NETABORT, //网络异常
E_SDK_CONFIG_CHNSTATUS, //通道状态
E_SDK_CONFIG_CHNMODE, //通道模式

E_SDK_CONFIG_NET_DAS, //主动注册
E_SDK_CONFIG_CAR_INPUT_EXCHANGE, //外部信息输入与车辆状态的对应关系
E_SDK_CONFIG_DELAY_TIME, //车载系统延时配置
E_SDK_CONFIG_NET_ORDER, //网络优先级
E_SDK_CONFIG_ABILITY_NETORDER , ////网络优先级设置能力
E_SDK_CONFIG_CARPLATE, //车牌号配置
E_SDK_CONFIG_LOCALSDK_NET_PLATFORM, ////网络平台信息设置
E_SDK_CONFIG_GPS_TIMING, //GPS校时相关配置
E_SDK_CONFIG_VIDEO_ANALYZE, //视频分析(智能DVR)
E_SDK_CONFIG_GODEYE_ALARM, //神眼接警
E_SDK_CONFIG_NAT_STATUS_INFO, //nat状态信息
E_SDK_CONFIG_BUGINFOSAVE, //命令调试(保存)
E_SDK_CONFIG_MEDIA_WATERMARK,//水印设置
E_SDK_CONFIG_ENCODE_STATICPARAM, //编码器静态参数
E_SDK_CONFIG_LOSS_SHOW_STR, //视频丢失显示字符串
E_SDK_CONFIG_DIGMANAGER_SHOW, //通道管理显示配置
E_SDK_CONFIG_ABILITY_ANALYZEABILITY, //智能分析能力
E_SDK_CONFIG_VIDEOOUT_PRIORITY, //显示HDMI VGA优先级别配置
E_SDK_CONFIG_NAT, //NAT功能，MTU值配置
E_SDK_CONFIG_CPCINFO, //智能CPC计数数据信息
E_SDK_CONFIG_STORAGE_POSITION, // 录像存储设备类型，
E_SDK_CONFIG_ABILITY_CARSTATUSNUM, //车辆状态数
E_SDK_CFG_VPN, //VPN
E_SDK_CFG_VIDEOOUT, ///VGA视频分辨率
E_SDK_CFG_ABILITY_VGARESOLUTION, //支持的VGA分辨率列表
E_SDK_CFG_NET_LOCALSEARCH, //搜索设备，设备端的局域网设备

```
        E_SDK_CFG_NETPLAT_KAINENG                    //温州凯能平台
}SDK_CONFIG_TYPE;

/// 支持的DDNS类型
struct SDK_DDNSServiceFunction
{
    int  nTypeNum;
    char vDDNSType[NET_MAX_DDNS_TYPE][64];
};
/// 区域遮挡能力集
struct SDK_BlindDetectFunction
{
    int iBlindConverNum;   ///< 区域遮挡块数
};

/// 动检区域能力集
struct SDK_MotionDetectFunction
{
    int iGridRow;
    int iGridColumn;
};
/// 串口协议
struct SDK_COMMFUNC
{
    int nProNum; /// 协议个数
    char vCommProtocol[SDK_COM_TYPES][32];
};
/// 云台协议
struct SDK_PTZPROTOCOLFUNC
{
    int nProNum; /// 协议个数
    char vPTZProtocol[100][NET_MAX_PTZ_PROTOCOL_LENGTH];
};
/// 编码信息
struct SDK_EncodeInfo
{
    bool bEnable;           ///< 使能项
    int iStreamType;        ///< 码流类型，capture_channel_t
    bool bHaveAudio;        ///< 是否支持音频
    unsigned int uiCompression;          ///< capture_comp_t的掩码
    unsigned int uiResolution;       ///< capture_size_t的掩码
};
/// 编码能力
```

```
struct CONFIG_EncodeAbility
{
    unsigned int iMaxEncodePower;       ///< 支持的最大编码能力
    int iChannelMaxSetSync;          ///< 每个通道分辨率是否需要同步0-不同步, 1 -同步
    unsigned int nMaxPowerPerChannel[NET_MAX_CHANNUM];       ///< 每个通道支持的最高编码能力
    unsigned int ImageSizePerChannel[NET_MAX_CHANNUM];       ///< 每个通道支持的图像分辨率
    unsigned int ExImageSizePerChannel[NET_MAX_CHANNUM];     ///< 每个通道支持的辅码流图像分
辨率
    SDK_EncodeInfo vEncodeInfo[SDK_CHL_FUNCTION_NUM];   ///< 编码信息,暂时最大就中码流
    SDK_EncodeInfo vCombEncInfo[SDK_CHL_FUNCTION_NUM]; ///< 组合编码信息,暂时最大就中码流};
    int   iMaxBps;                  ///< 最高码流 Kbps
    unsigned int ExImageSizePerChannelEx[NET_MAX_CHANNUM][NET_CAPTURE_SIZE_NUM];
}

///支持的系统功能
struct SDK_SystemFunction
{
    bool vEncodeFunction[SDK_ENCODE_FUNCTION_TYPE_NR];  ///< 编码功能EncodeFunctionTypes
    bool vAlarmFunction[SDK_ALARM_FUNCTION_TYPE_NR];    ///< 报警功能AlarmFucntionTypes
    bool vNetServerFunction[SDK_NET_SERVER_TYPES_NR];///< 网络服务功能NetServerTypes
    bool vPreviewFunction[SDK_PREVIEW_TYPES_NR];       ///< 预览功能PreviewTypes
};
///< 自动维护设置
struct SDK_AutoMaintainConfig
{
    int iAutoRebootDay;                 ///< 自动重启设置日期
    int iAutoRebootHour;             ///< 重启整点时间   [0, 23]
    int iAutoDeleteFilesDays;        ///< 自动删除文件时间[0, 30]
};
//硬盘信息
struct SDK_STORAGEDISK
{
    int         iPhysicalNo;
    int         iPartNumber;        // 分区数
    SDK_DriverInformation diPartitions[SDK_MAX_DRIVER_PER_DISK];
};
struct SDK_StorageDeviceInformationAll
{
    int iDiskNumber;
    SDK_STORAGEDISK vStorageDeviceInfoAll[SDK_MAX_DISK_PER_MACHINE];
};

// 云台联动类型
```

```
enum PtzLinkTypes
{
    PTZ_LINK_NONE,      // 不需要联动
    PTZ_LINK_PRESET,    // 转至预置点
    PTZ_LINK_TOUR,      // 巡航
    PTZ_LINK_PATTERN        // 轨迹
};

// 云台联动结构
struct SDK_PtzLinkConfig
{
  int iType;        // 联动的类型
  int iValue;       // 联动的类型对应的值
};

#define CHANNELNAME_MAX_LEN 64  //通道名称最大长度

// 联动操作
struct SDK_EventHandler
{
  unsigned int    dwRecord;                   // 录象掩码
  int         iRecordLatch;                   // 录像延时：10～300 sec
  unsigned int    dwTour;                     // 轮巡掩码
  unsigned int    dwSnapShot;                 // 抓图掩码
  unsigned int    dwAlarmOut;                 // 报警输出通道掩码
  unsigned int    dwMatrix;                   // 矩阵掩码
  int         iEventLatch;            // 联动开始延时时间，s 为单位
  int         iAOLatch;                   // 报警输出延时：10～300 sec
  SDK_PtzLinkConfig PtzLink[NET_MAX_CHANNUM];         // 云台联动项
  SDK_CONFIG_WORKSHEET schedule;          // 录像时间段
  bool    bRecordEn;                  // 录像使能
  bool    bTourEn;                // 轮巡使能
  bool    bSnapEn;                    // 抓图使能
  bool    bAlarmOutEn;            // 报警使能
  bool    bPtzEn;                     // 云台联动使能
  bool    bTip;                       // 屏幕提示使能
  bool    bMail;                      // 发送邮件
  bool    bMessage;                   // 发送消息到报警中心
  bool    bBeep;                      // 蜂鸣
  bool    bVoice;                     // 语音提示
  bool    bFTP;                       // 启动 FTP 传输
  bool    bMatrixEn;                  // 矩阵使能
  bool    bLog;                       // 日志使能，目前只有在 WTN 动态检测中使用
```

```
    bool    bMessagetoNet;                  // 消息上传给网络使能

    bool    bShowInfo;              // 是否在 GUI 上和编码里显示报警信息
    unsigned int    dwShowInfoMask;         // 要联动显示报警信息的通道掩码
    char    pAlarmInfo[CHANNELNAME_MAX_LEN];//要显示的报警信息
};

///< 遮挡检测配置
struct SDK_BLINDDETECTCONFIG
{
    bool bEnable;       ///< 遮挡检测开启
    int         iLevel;                 ///< 灵敏度：~
    SDK_EventHandler hEvent; ///< 遮挡检测联动
};

///< 报警输入配置
struct SDK_ALARM_INPUTCONFIG
{
    bool bEnable;       ///< 报警输入开关
    int         iSensorType;  ///< 传感器类型常开or 常闭
    SDK_EventHandler hEvent; ///< 报警联动
};
///< 网路报警
struct SDK_NETALARMCONFIG
{
    bool bEnable;               ///< 使能
    SDK_EventHandler hEvent; ///< 处理参数
};

/// 所有通道的网路报警结构
struct SDK_NETALARMCONFIG_ALL
{
    SDK_NETALARMCONFIG vNetAlarmConfig[NET_MAX_CHANNUM];
};
///< 本地报警输出配置
struct SDK_AlarmOutConfig
{
    int nAlarmOutType;              ///< 报警输出类型: 配置,手动,关闭
    int nAlarmOutStatus;    ///< 报警状态: 0:打开1;闭合
};

///< 所有通道的报警输出配置
struct SDK_AlarmOutConfigAll
```

```
{
    SDK_AlarmOutConfig vAlarmOutConfigAll[NET_MAX_CHANNUM];
};

/// 所有通道的解码器地址设置V2版本
struct SDK_AbilitySerialNo
{
    char serialNo[NET_MAX_SERIALNO_LENGTH];
    char productType[NET_MAX_SERIALNO_LENGTH];
};

///< 驱动器信息结构
struct SDK_DriverInformation
{
    int         iDriverType;        ///< 驱动器类型
    bool bIsCurrent;                ///< 是否为当前工作盘
    unsigned int   uiTotalSpace;        ///< 总容量，MB为单位
    unsigned int   uiRemainSpace;           ///< 剩余容量，MB为单位
    int         iStatus;            ///< 错误标志，文件系统初始化时被设置
    int         iLogicSerialNo;                 ///< 逻辑序号
    SDK_SYSTEM_TIME  tmStartTimeNew;            ///< 新录像时间段的开始时间
    SDK_SYSTEM_TIME      tmEndTimeNew;              ///< 新录像时间段的结束时间
    SDK_SYSTEM_TIME      tmStartTimeOld;        ///< 老录像时间段的开始时间
    SDK_SYSTEM_TIME      tmEndTimeOld;              ///< 老录像时间段的结束时间
};

///< 所有通道的报警输入配置
struct SDK_ALARM_INPUTCONFIG_ALL
{
    SDK_ALARM_INPUTCONFIG vAlarmConfigAll[NET_MAX_CHANNUM];
};

/// 全通道遮挡检测配置
struct SDK_BLINDDETECTCONFIG_ALL
{
    SDK_BLINDDETECTCONFIG vBlindDetectAll[NET_MAX_CHANNUM];
};
///< 动态检测设置
struct SDK_MOTIONCONFIG
{
    bool bEnable;                               // 动态检测开启
    int iLevel;                                     // 灵敏度
    unsigned int mRegion[NET_MD_REGION_ROW];            // 区域，每一行使用一个二进制串
```

```
    SDK_EventHandler hEvent;                    // 动态检测联动
};
/// 全通道动态检测配置
struct SDK_MOTIONCONFIG_ALL
{
    SDK_MOTIONCONFIG vMotionDetectAll[NET_MAX_CHANNUM];
};
///< 视频丢失
struct SDK_VIDEOLOSSCONFIG
{
    bool bEnable;           ///< 使能
    SDK_EventHandler hEvent; ///< 处理参数
};
/// 所有通道的视频丢失结构
struct SDK_VIDEOLOSSCONFIG_ALL
{
    SDK_VIDEOLOSSCONFIG vGenericEventConfig[NET_MAX_CHANNUM];
};

/// 录像模式种类
enum SDK_RecordModeTypes
{
    SDK_RECORD_MODE_CLOSED,     ///< 关闭录像
    SDK_RECORD_MODE_MANUAL,     ///< 手动录像
    SDK_RECORD_MODE_CONFIG,     ///< 按配置录像
    SDK_RECORD_MODE_NR,
};

///< 录像设置
struct SDK_RECORDCONFIG
{
    int iPreRecord;             ///< 预录时间，为零时表示关闭
    bool bRedundancy;           ///< 冗余开关
    bool bSnapShot;             ///< 快照开关
    int iPacketLength;      ///< 录像打包长度（分钟）[1, 255]
    int iRecordMode;        ///< 录像模式，见SDK_RecordModeTypes
    SDK_CONFIG_WORKSHEET wcWorkSheet;           ///< 录像时间段
    unsigned int typeMask[NET_N_WEEKS][NET_N_TSECT];    ///< 录像类型掩码
};

//录像设置结构体
struct SDK_RECORDCONFIG_ALL
{
```

```
    SDK_RECORDCONFIG vRecordConfigAll[NET_MAX_CHANNUM];
};
///< 图片设置
struct SDK_SnapshotConfig
{
    int iPreSnap;              ///< 预抓图片数
    bool bRedundancy;          ///< 冗余开关
    int iSnapMode;             ///< 录像模式，见RecordModeTypes
    SDK_CONFIG_WORKSHEET wcWorkSheet;                  ///< 录像时间段
    unsigned int typeMask[NET_N_WEEKS][NET_N_TSECT];  ///< 录像类型掩码，见enum RecordTypes
};
///<录像设置
struct SDK_SnapshotConfigAll
{
    SDK_SnapshotConfig vSnapshotConfigAll[NET_MAX_CHANNUM];
};
//普通配置页结构体
typedef struct _SDK_CONFIG_NORMAL
{
    SDK_SYSTEM_TIME sysTime;           ///< 系统时间
    int iLocalNo;                  ///< 本机编号:[0, 998]
    int iOverWrite;                ///< 硬盘满时处理 "OverWrite", "StopRecord"
    int iSnapInterval;             ///< 定时抓图的时间间隔，以秒为单位
    char sMachineName[64];     ///< 机器名
    int iVideoStartOutPut;  ///< 输出模式*/
    int iAutoLogout;               ///< 本地菜单自动注销(分钟)  [0, 120]
    int iVideoFormat;          ///< 视频制式:“PAL”, “NTSC”, “SECAM”
    int iLanguage;                 ///< 语言选择:“English”, “SimpChinese”, “TradChinese”, “Italian”,
“Spanish”, “Japanese”, “Russian”, “French”, “German”
    int iDateFormat;           ///< 日期格式:“YYMMDD”, “MMDDYY”, “DDMMYY”
    int iDateSeparator;            ///< 日期分割符:“.”, “-”, “/”
    int iTimeFormat;           ///< 时间格式:“12”, “24”
    int iDSTRule;              ///< 夏令时规则
    int iWorkDay;              ///< 工作日
    DSTPoint dDSTStart;
    DSTPoint dDSTEnd;
}SDK_CONFIG_NORMAL;

// 编码设置
struct SDK_CONFIG_ENCODE
{
    SDK_MEDIA_FORMAT dstMainFmt[SDK_ENCODE_TYPE_NUM];          // 主码流格式
    SDK_MEDIA_FORMAT dstExtraFmt[SDK_EXTRATYPES];   // 辅码流格式
```

```
    SDK_MEDIA_FORMAT dstSnapFmt[SDK_ENCODE_TYPE_NUM];          // 抓图格式
};
struct  SDK_EncodeConfigAll
{
    SDK_CONFIG_ENCODE vEncodeConfigAll[NET_MAX_CHANNUM];
};
// 简化版本编码配置
/// 媒体格式
struct SDK_MEDIA_FORMAT_SIMPLIIFY
{
    SDK_VIDEO_FORMAT vfFormat;              // 视频格式定义
    bool bVideoEnable;                  // 开启视频编码
    bool bAudioEnable;                  // 开启音频编码
};

/// 编码设置
struct SDK_CONFIG_ENCODE_SIMPLIIFY
{
    SDK_MEDIA_FORMAT dstMainFmt;            // 主码流格式
    SDK_MEDIA_FORMAT dstExtraFmt;    // 辅码流格式
};

/// 全通道编码配置
struct SDK_EncodeConfigAll_SIMPLIIFY
{
    SDK_CONFIG_ENCODE_SIMPLIIFY vEncodeConfigAll[NET_MAX_CHANNUM];
};


// 组合编码设置
struct  SDK_CombineEncodeConfigAll
{
    SDK_CONFIG_ENCODE vEncodeConfigAll[NET_MAX_COMBINE_NUM];
};
struct SDK_CombEncodeParam
{
    int iEncodeMode;   //见CombineEncodeMode
};
struct SDK_CombEncodeModeAll
{
    SDK_CombEncodeParam vEncodeParam[NET_MAX_COMBINE_NUM];
};
//!视频物件结构
```

```
struct  SDK_VIDEO_WIDGET
{
    unsigned int rgbaFrontground;        ///< 物件的前景MakeRGB，和透明度
    unsigned int rgbaBackground;         ///< 物件的后景MakeRGB，和透明度
    sdkRect  rcRelativePos;              ///< 物件边距与整长的比例*8191
    bool bPreviewBlend;                  ///< 预览叠加
    bool bEncodeBlend;                   ///< 编码叠加
};

//!视频物件设置
struct SDK_CONFIG_VIDEOWIDGET
{
    SDK_VIDEO_WIDGET    dstCovers[NET_COVERNUM];
    SDK_VIDEO_WIDGET    ChannelTitle;
    SDK_VIDEO_WIDGET    TimeTitle;
    struct
    {
        char strName[NET_NAME_PASSWORD_LEN];
        __int64 iSerialNo;
    }ChannelName;                        /// 通道名称
    int                 iCoverNum;       /*!< 当前该通道有几个叠加的区域*/
};

/// 视频物件(输出模式对话框)
struct SDK_VideoWidgetConfigAll
{
    SDK_CONFIG_VIDEOWIDGET vVideoWidegetConfigAll[NET_MAX_CHANNUM];
};
/// 所有通道名称标题
struct SDK_ChannelNameConfigAll
{
    char channelTitle[NET_MAX_CHANNUM][NET_NAME_PASSWORD_LEN];
};
//输出模式
struct SDK_GUISetConfig
{
    int iWindowAlpha;           ///< 窗口透明度      [128, 255]
    bool bTimeTitleEn;          ///< 时间标题显示使能
    bool bChannelTitleEn;       ///< 通道标题显示使能
    bool bAlarmStatus;          ///<  报警状态
    bool bRecordStatus;         ///<  录像状态显示使能
    bool bChanStateRecEn;       ///< 录像标志显示使能
    bool bChanStateVlsEn;       ///< 视频丢失标志显示使能
```

```
    bool bChanStateLckEn;       ///< 通道锁定标志显示使能
    bool bChanStateMtdEn;       ///< 动态检测标志显示使能
    bool bBitRateEn;            ///< 码流显示使能
    bool bRemoteEnable;         ///< 遥控器使能
    bool bDeflick;              ///< 抗抖动
};

////!普通网络设置
struct SDK_CONFIG_NET_COMMON
{
    char HostName[NET_NAME_PASSWORD_LEN]; //!主机名
    CONFIG_IPAddress HostIP;            //!主机IP
    CONFIG_IPAddress Submask;           //!子网掩码
    CONFIG_IPAddress Gateway;           //!网关IP
    int HttpPort;                       //!HTTP服务端口
    int TCPPort;                        //!TCP侦听端口
    int SSLPort;                        //!SSL侦听端口
    int UDPPort;                        //!UDP侦听端口
    int MaxConn;                        //!最大连接数
    int MonMode;                        //!监视协议{"TCP","UDP","MCAST",...}
    int MaxBps;                         //!限定码流值
    int TransferPlan;                   //!传输策略//char TransferPlan[NET_NAME_PASSWORD_LEN];
    bool bUseHSDownLoad;                //!是否启用高速录像下载
    char sMac[NET_MAX_MAC_LEN];         //!MAC地址
    unsigned int DeviceID; ///< 设备ID
};
//DHCP
struct SDK_NetDHCPConfig
{
    bool bEnable;
    char ifName[32];
};
/// 所有网卡的DHCP配置
struct SDK_NetDHCPConfigAll
{
    SDK_NetDHCPConfig vNetDHCPConfig[2];
};
///< 服务器结构定义
struct SDK_RemoteServerConfig
{
    char ServerName[NET_NAME_PASSWORD_LEN];     ///< 服务名
    CONFIG_IPAddress ip;                        ///< IP地址
    int Port;                           ///< 端口号
```

```
    char UserName[NET_NAME_PASSWORD_LEN];       ///< 用户名
    char Password[NET_NAME_PASSWORD_LEN];       ///< 密码
    bool Anonymity;                             ///< 是否匿名登录
};

// 云台设置
struct SDK_STR_CONFIG_PTZ
{
    char sProtocolName[NET_MAX_PTZ_PROTOCOL_LENGTH];    // 协议名称
    int   ideviceNo;                    // 云台设备地址编号
    int   iNumberInMatrixs;     // 在矩阵中的统一编号
    int iPortNo;                    // 串口端口号   [1, 4]
    SDK_COMMATTRI dstComm;                  // 串口属性
};
//所有通道云台协议
struct SDK_STR_PTZCONFIG_ALL
{
    SDK_STR_CONFIG_PTZ ptzAll[NET_MAX_CHANNUM];
};
struct SDK_CONFIG_WORKSHEET
{
    SDK_TIMESECTION     tsSchedule[NET_N_WEEKS][NET_N_TSECT];    /*!< 时间段*/
};
// 串口配置
struct SDK_CONFIG_COMM_X
{
    char iProtocolName[32];     // 串口协议:“Console”
    int iPortNo;        // 端口号
    SDK_COMMATTRI aCommAttri;       // 串口属性
};
struct SDK_CommConfigAll
{
    SDK_CONFIG_COMM_X vCommConfig[SDK_COM_TYPES];
};


///< IP权限设置
struct SDK_NetIPFilterConfig
{
    bool Enable;        ///< 是否开启
    CONFIG_IPAddress BannedList[NET_MAX_FILTERIP_NUM];          ///< 黑名单列表
    CONFIG_IPAddress TrustList[NET_MAX_FILTERIP_NUM];       ///< 白名单列表
};
```

```
///< 组播设置
struct SDK_NetMultiCastConfig
{
    bool Enable;        ///< 是否开启
    SDK_RemoteServerConfig Server;          ///< 组播服务器
};

///< pppoe设置
struct SDK_NetPPPoEConfig
{
    bool Enable;  ///< 是否开启
    SDK_RemoteServerConfig Server;          ///< PPPOE服务器
     CONFIG_IPAddress addr;          ///< 拨号后获得的IP地址
};

///< DDNS设置
struct SDK_NetDDNSConfig
{
    bool Enable;  ///< 是否开启
    bool Online;        ///< 是否在线
    char DDNSKey[NET_NAME_PASSWORD_LEN];  ///< DDNS类型名称
    char HostName[NET_NAME_PASSWORD_LEN]; ///< 主机名
    SDK_RemoteServerConfig Server;                  ///< DDNS服务器
};

///< DDNS设置
struct SDK_NetDDNSConfigALL
{
    SDK_NetDDNSConfig ddnsConfig[NET_MAX_DDNS_TYPE];
};

///< ftp设置
struct SDK_FtpServerConfig {
    bool bEnable;       ///< 服务器使能
    SDK_RemoteServerConfig Server;     ///< FTP服务器
    char cRemoteDir[NET_MAX_PATH_LENGTH];     ///< 远程目录
    int iMaxFileLen;     ///< 文件最大长度
};


///< NTP设置
```

```
struct SDK_NetNTPConfig
{
    ///< 是否开启
    bool Enable;
    ///< 服务器
    SDK_RemoteServerConfig Server;
    ///< 更新周期
    int UpdatePeriod;
    ///< 时区
    int TimeZone;
};
#define  NET_MAX_EMAIL_TITLE_LEN 64
#define  NET_MAX_EMAIL_RECIEVERS  5
#define  NET_EMAIL_ADDR_LEN  32

///< EMAIL设置
struct SDK_NetEmailConfig
{
    ///< 是否开启
    bool Enable;
    ///< smtp 服务器地址使用字符串形式填充
    ///< 可以填ip,也可以填域名
    SDK_RemoteServerConfig Server;
    bool bUseSSL;
    ///< 发送地址
    char SendAddr[NET_EMAIL_ADDR_LEN];
    ///< 接收人地址
    char Recievers[NET_MAX_EMAIL_RECIEVERS][NET_EMAIL_ADDR_LEN];
    ///< 邮件主题
    char Title[NET_MAX_EMAIL_TITLE_LEN];
    ///< email有效时间
    SDK_TIMESECTION Schedule[NET_N_MIN_TSECT];
};
///< ARSP(主动注册服务器)设置
struct SDK_NetARSPConfig
{
    bool bEnable; ///< 是否开启
    char sARSPKey[NET_NAME_PASSWORD_LEN];  ///< DNS类型名称
    int iInterval;   ///< 保活间隔时间
    char sURL[NET_NAME_PASSWORD_LEN];    ///< 本机域名
    SDK_RemoteServerConfig Server;            ///< DDNS服务器
     int nHttpPort;                ///< 服务器HTTP端口
};
```

```
struct SDK_NetARSPConfigAll
{
    SDK_NetARSPConfig vNetARSPConfigAll[NET_MAX_ARSP_TYPE];
};
///< 解码器地址设置
struct SDK_NetDecorderConfig
{
    bool Enable;                                ///< 是否开启
    char UserName[NET_NAME_PASSWORD_LEN]; ///< DDNS类型名称, 目前有: JUFENG
    char PassWord[NET_NAME_PASSWORD_LEN];   ///< 主机名
    char Address[NET_NAME_PASSWORD_LEN];
    int Protocol;
    int Port;                                   ///< 解码器连接端口
    int Channel;                                ///< 解码器连接通道号
     int Interval;                  ///< 轮巡的间隔时间(s)
};
/// 解码器地址设置
struct SDK_NetDecorderConfigAll
{
    SDK_NetDecorderConfig vNetDecorderConfig[NET_MAX_DECORDR_CH];
};

/// 解码器地址设置V2版本
struct SDK_NetDecoderConfig_V2
{
    int nTotalNum;          //有效的数组个数，最大为NET_MAX_CHANNUM
    SDK_NetDecoderConfig vNetDecoderConfig[NET_MAX_CHANNUM];
};

/// 所有通道的解码器地址设置V2版本
struct SDK_NetDecoderConfigAll_V2
{
    SDK_NetDecoderConfig_V2 vNetDecoderArray[NET_MAX_DECORDR_CH];
};

///< 解码器地址设置
struct SDK_NetDecorderConfigV3
{
    bool Enable;                                ///< 是否开启
    char UserName[NET_NAME_PASSWORD_LEN]; ///< DDNS类型名称, 目前有: JUFENG
    char PassWord[NET_NAME_PASSWORD_LEN];   ///< 主机名
    char Address[NET_NAME_PASSWORD_LEN];
    int Protocol;
```

```
    int Port;                                   ///< 解码器连接端口
    int Channel;                                ///< 解码器连接通道号
    int Interval;                   ///< 轮巡的间隔时间(s),0:表示永久
    char ConfName[NET_NAME_PASSWORD_LEN]; ///<配置名称
    int DevType;                                ///<设备类型
    int StreamType;                                 ///<连接的码流类型CaptureChannelTypes
};

/*解码器连接类型*/
enum SDK_DecorderConnType
{
    SDK_CONN_SINGLE = 0,    /*单连接*/
    SDK_CONN_MULTI = 1,     /*多连接轮巡*/
    SDK_CONN_TYPE_NR,
};

/*数字通道的配置*/
struct SDK_NetDigitChnConfig
{
    bool Enable;        /*数字通道是否开启*/
    int ConnType;       /*连接类型，取DecoderConnectType的值*/
    int TourIntv;       /*多连接时轮巡间隔*/
    unsigned int SingleConnId;  /*单连接时的连接配置ID, 从1开始，0表示无效*/
    bool EnCheckTime;       /*开启对时*/
    SDK_NetDecorderConfigV3 NetDecorderConf[32]; /*网络设备通道配置表*/
};

/*所有数字通道的配置*/
struct SDK_NetDecorderConfigAll_V3
{
    SDK_NetDigitChnConfig DigitChnConf[NET_MAX_DECORDR_CH];
};

///< 3G拨号设置
struct SDK_Net3GConfig
{
    bool bEnable;               ///< 无线模块使能标志
    int iNetType;               ///< 无线网络类型
    char sAPN[NET_NAME_PASSWORD_LEN];       ///< 接入点名称
    char sDialNum[NET_NAME_PASSWORD_LEN];   ///< 拨号号码
    char sUserName[NET_NAME_PASSWORD_LEN];  ///< 拨号用户名
    char sPassword[NET_NAME_PASSWORD_LEN];  ///< 拨号密码
    CONFIG_IPAddress addr;                  ///< 拨号后获得的IP地址
```

```
};
///< 手机监控设置包括
struct SDK_NetMoblieConfig
{
    bool bEnable; ///< 是否开启
    SDK_RemoteServerConfig Server;          ///< 服务器
};
///< UPNP设置
struct SDK_NetUPNPConfig
{
    bool bEnable;               ///< 使能标志
    bool bState;            ///< 状态, 1: OK 0: NOK
    int iHTTPPort;              ///< HTTP连接映射后的端口
    int iMediaPort;                 ///< 媒体连接映射后的端口
    int iMobliePort;            ///< 手机监控映射后的端口
};

///< WIFI设置
struct SDK_NetWifiConfig
{
    bool bEnable;
    char sSSID[36];         //SSID Number
    int nChannel;               //channel
    char sNetType[32];      //Infra, Adhoc
    char sEncrypType[32];     //NONE, WEP, TKIP, AES
    char sAuth[32];         //OPEN, SHARED, WEPAUTO, WPAPSK, WPA2PSK, WPANONE, WPA, WPA2
    int  nKeyType;              //0:Hex 1:ASCII
    char sKeys[NET_IW_ENCODING_TOKEN_MAX];
    CONFIG_IPAddress HostIP;        ///< host ip
    CONFIG_IPAddress Submask;           ///< netmask
    CONFIG_IPAddress Gateway;           ///< gateway
};
enum SDK_RSSI_SINGNAL
{
    SDK_RSSI_NO_SIGNAL,   //<= -90db
    SDK_RSSI_VERY_LOW,    //<= -81db
    SDK_RSSI_LOW,         //<= -71db
    SDK_RSSI_GOOD,        //<= -67db
    SDK_RSSI_VERY_GOOD,   //<= -57db
    SDK_RSSI_EXCELLENT    //>-57db
};
struct SDK_NetWifiDevice
{
```

```
    char sSSID[36];          //SSID Number
    int nRSSI;               //SEE SDK_RSSI_SINGNAL
    int nChannel;
    char sNetType[32];       //Infra, Adhoc
    char sEncrypType[32];    //NONE, WEP, TKIP, AES
    char sAuth[32];          //OPEN, SHARED, WEPAUTO, WPAPSK, WPA2PSK, WPANONE, WPA, WPA2
};
struct SDK_NetWifiDeviceAll
{
    int nDevNumber;
    SDK_NetWifiDevice vNetWifiDeviceAll[NET_MAX_AP_NUMBER];
};

///< DNS设置
struct SDK_NetDNSConfig
{
    CONFIG_IPAddress      PrimaryDNS;
    CONFIG_IPAddress      SecondaryDNS;
};


/// 音频输入格式，语音对讲用
struct SDK_AudioInFormatConfig
{
    int iBitRate;      ///< 码流大小，kbps为单位
    int iSampleRate;   ///< 采样率，Hz为单位
    int iSampleBit;    ///< 采样的位深
    int iEncodeType;   ///< 编码方式，参照AudioEncodeTypes定义
};

/// 告警状态
struct SDK_DVR_ALARMSTATE
{
    int iVideoMotion; ///< 移动侦测状态,用掩码表示通道号,bit0代表通道一,以此类推1: 有告警0: 无告警
    int iVideoBlind; ///< 视频遮挡状态,用掩码表示通道号,bit0代表通道一,以此类推1: 有告警0: 无告警
    int iVideoLoss;      ///< 视频丢失状态,用掩码表示通道号,bit0代表通道一,以此类推1: 有告警0: 无告警
    int iAlarmIn;   ///< 告警输入状态,用掩码表示通道号,bit0代表通道一,以此类推1: 有告警0: 无告警
    int iAlarmOut;///< 告警输出状态,用掩码表示通道号,bit0代表通道一,以此类推1: 有告警0: 无告警
};

// 通道状态
struct SDK_DVR_CHANNELSTATE
{
```

```
    bool bRecord; ///< 是否正在录像
    int iBitrate;    ///< 当前码率
};

// 设备工作状态信息
struct SDK_DVR_WORKSTATE
{
    SDK_DVR_CHANNELSTATE vChnState[NET_MAX_CHANNUM];
    SDK_DVR_ALARMSTATE vAlarmState;
};
/// 支持语言
struct SDK_MultiLangFunction
{
    //每个协议最多由个字符组成
    int nLangNum;
    char vLanguageName[128][64];
};
/// 支持的视频制式
struct SDK_MultiVstd
{
    //每个协议最多由个字符组成
    int nVstdNum;
    char vVstdName[3][64];
};
/// 恢复的默认配置种类
struct SDK_SetDefaultConfigTypes
{
    bool vSetDefaultKinds[SDK_DEFAULT_CFG_END];
};
/// 语音对讲格式
struct SDK_AudioInFormatConfigAll
{
    SDK_AudioInFormatConfig vAudioInFormatConfig[SDK_AUDIO_ENCODE_TYPES_NR];
};
// 升级信息获取
struct SDK_UpgradeInfo
{
    char szSerial[64];
    char szHardware[64];
    char szVendor[64];
    unsigned int uiLogoArea[2];
};
typedef struct {
```

```
    int left;
    int top;
    int right;
    int bottom;
}sdkRect;

/// 音频输入格式，语音对讲用
struct SDK_AudioInFormatConfig
{
    int iBitRate;      ///< 码流大小，kbps为单位，比如kbps，kbps
    int iSampleRate;   ///< 采样率，Hz为单位，比如Hz
    int iSampleBit;    ///< 采样的位深
    int iEncodeType;   ///< 编码方式，参照AudioEncodeTypes定义
};

//语音对讲格式
typedef enum __TALK_CODING_TYPE
{
    TALK_DEFAULT = 0,
    TALK_PCM = 1,                   //PCM
    TALK_G711a,                     //G711a
    TALK_AMR,                       //AMR
    TALK_G711u,                     //G711u
    TALK_G726,                      //G726
}TALK_CODING_TYPE;
//语音对讲
typedef struct
{
    TALK_CODING_TYPE        encodeType;     //编码类型
    int                     nAudioBit;      //用实际的值表示，如位则填值为
    unsigned int            dwSampleRate;   //采样率，如k 则填值为
    char                    reserved[64];
}H264_DVR_TALKDECODE_INFO;
struct SDK_VIDEO_FORMAT
{
    int     iCompression;               // 压缩模式
    int     iResolution;            // 分辨率参照枚举capture_size_t
    int     iBitRateControl;        // 码流控制参照枚举capture_brc_t
    int     iQuality;               // 码流的画质档次-6
    int     nFPS;                       // 帧率值，NTSC/PAL不区分,负数表示多秒一帧
    int     nBitRate;               // 0-4096k,该列表主要由客户端保存，设备只接收实际的码流值而不是
下标。
    int     iGOP;                       // 描述两个I帧之间的间隔时间，-12
```

```
};
struct  SDK_AUDIO_FORMAT
{
    int         nBitRate;               //  码流kbps
    int         nFrequency;                 //  采样频率
    int         nMaxVolume;                 //  最大音量阈值
};
// 媒体格式
struct SDK_MEDIA_FORMAT
{
    SDK_VIDEO_FORMAT vfFormat;              //  视频格式定义
    SDK_AUDIO_FORMAT afFormat;              //  音频格式定义
    bool bVideoEnable;                  //  开启视频编码
    bool bAudioEnable;                  //  开启音频编码
};

typedef union {    //IP addr
    unsigned char      c[4];
    unsigned short     s[2];
    unsigned int   l;
}CONFIG_IPAddress;

//短信配置
struct SDK_NetShortMsgCfg
{
    bool bEnable;      //发送手机短信的功能是否启用
    char pDesPhoneNum[MAX_RECIVE_MSG_PHONE_COUNT][16];
    int  sendTimes;    //需要向每个手机发送多少次短信
};
//手机彩信配置
struct SDK_NetMultimediaMsgCfg
{
    bool bEnable;                   // 发送手机彩信的功能是否启用
    char pDesPhoneNum[MAX_RECIVE_MSG_PHONE_COUNT][16]; //接收彩信的手机号，现支
持个手机号
    char pGateWayDomain[40];  // 网关地址，域名或IP
    int  gateWayPort;               // 网关端口
    char pMmscDomain[40];       // 彩信服务器地址，IP或域名
    int  mmscPort;                  // 彩信服务器端口号
};
```

2.1.6 网络键盘键值定义

```
/// 按键值
enum SDK_NetKeyBoardValue
{
    SDK_NET_KEY_0, SDK_NET_KEY_1, SDK_NET_KEY_2, SDK_NET_KEY_3, SDK_NET_KEY_4,
SDK_NET_KEY_5, SDK_NET_KEY_6, SDK_NET_KEY_7, SDK_NET_KEY_8, SDK_NET_KEY_9,
    SDK_NET_KEY_10, SDK_NET_KEY_11, SDK_NET_KEY_12, SDK_NET_KEY_13, SDK_NET_KEY_14,
SDK_NET_KEY_15, SDK_NET_KEY_16, SDK_NET_KEY_10PLUS,
    SDK_NET_KEY_UP = 20,    // 上或者云台向上
    SDK_NET_KEY_DOWN,       // 下或者云台向下
    SDK_NET_KEY_LEFT,       // 左或者云台向左
    SDK_NET_KEY_RIGHT,      // 右或者云台向右
    SDK_NET_KEY_SHIFT,
    SDK_NET_KEY_PGUP,       // 上一页
    SDK_NET_KEY_PGDN,       // 下一页
    SDK_NET_KEY_RET,        // 确认
    SDK_NET_KEY_ESC,        // 取消或退出
    SDK_NET_KEY_FUNC,       // 切换输入法
    SDK_NET_KEY_PLAY,       // 播放/暂停
    SDK_NET_KEY_BACK,       // 倒放
    SDK_NET_KEY_STOP,       // 停止
    SDK_NET_KEY_FAST,       // 快放
    SDK_NET_KEY_SLOW,       // 慢放
    SDK_NET_KEY_NEXT,       // 下一个文件
    SDK_NET_KEY_PREV,       // 上一个文件
    SDK_NET_KEY_REC = 40,   // 录像设置
    SDK_NET_KEY_SEARCH,     // 录像查询
    SDK_NET_KEY_INFO,       // 系统信息
    SDK_NET_KEY_ALARM,      // 告警输出
    SDK_NET_KEY_ADDR,       // 遥控器地址设置
    SDK_NET_KEY_BACKUP,     // 备份
    SDK_NET_KEY_SPLIT,      // 画面分割模式切换，每按一次切换到下一个风格模式
    SDK_NET_KEY_SPLIT1,     // 单画面
    SDK_NET_KEY_SPLIT4,     // 四画面
    SDK_NET_KEY_SPLIT8,     // 八画面
    SDK_NET_KEY_SPLIT9,     // 九画面
    SDK_NET_KEY_SPLIT16,    // 16画面
    SDK_NET_KEY_SHUT,       // 关机
    SDK_NET_KEY_MENU,       // 菜单
    SDK_NET_KEY_PTZ = 60,   // 进入云台控制模式
    SDK_NET_KEY_TELE,       // 变倍减
```

```
    SDK_NET_KEY_WIDE,       // 变倍加
    SDK_NET_KEY_IRIS_SMALL, // 光圈增
    SDK_NET_KEY_IRIS_LARGE, // 光圈减
    SDK_NET_KEY_FOCUS_NEAR, // 聚焦远
    SDK_NET_KEY_FOCUS_FAR,  // 聚焦近
    SDK_NET_KEY_BRUSH,      // 雨刷
    SDK_NET_KEY_LIGHT,      // 灯光
    SDK_NET_KEY_SPRESET,    // 设置预置点
    SDK_NET_KEY_GPRESET,    // 转至预置点
    SDK_NET_KEY_DPRESET,    // 清除预置点
    SDK_NET_KEY_PATTERN,    // 模式
    SDK_NET_KEY_SCANON,     // 自动扫描开始
    SDK_NET_KEY_SCANOFF,    // 自动扫描结束
    SDK_NET_KEY_AUTOTOUR,   // 自动巡航
    SDK_NET_KEY_AUTOPANON,  // 线扫开始
    SDK_NET_KEY_AUTOPANOFF, // 线扫结束
};

/// 按键状态
enum SDK_NetKeyBoardState
{
    SDK_NET_KEYBOARD_KEYDOWN,      // 按键按下
    SDK_NET_KEYBOARD_KEYUP,        // 按键松开
};

struct SDK_NetKeyBoardData
{
    int iValue;             // 见 SDK_NetKeyBoardValue
    int iState;             // 见 SDK_NetKeyBoardState
};
```

2.1.7 网络报警信息

```
/// 网络报警
struct SDK_NetAlarmInfo
{
    int iEvent;  //目前未使用

    int iState;

}

///< 报警中心设置
```

```
struct SDK_NetAlarmCenterConfig
{
    bool bEnable;       ///< 是否开启
    char sAlarmServerKey[NET_NAME_PASSWORD_LEN];     ///< 报警中心协议类型名称,
    ///< 报警中心服务器
    SDK_RemoteServerConfig Server;
    bool bAlarm;
    bool bLog;

  };

struct SDK_NetAlarmServerConfigAll
{
    SDK_NetAlarmCenterConfig vAlarmServerConfigAll[NET_MAX_ALARMSERVER_TYPE];
};
// 报警中心消息类型
enum SDK_AlarmCenterMsgType
{
    SDK_ALARMCENTER_ALARM,
    SDK_ALARMCENTER_LOG,
};
// 报警中心消息类型
enum SDK_AlarmCenterStatus
{
    SDK_AC_START,
    SDK_AC_STOP,

  };

// 告警中心消息内容
struct SDK_NetAlarmCenterMsg
{
    CONFIG_IPAddress HostIP;          ///< 设备IP
    int nChannel;                     ///< 通道
    int nType;                       ///< 类型见AlarmCenterMsgType
    int nStatus;                      ///< 状态见AlarmCenterStatus
    SDK_SYSTEM_TIME Time;             ///< 发生时间
    char sEvent[NET_MAX_INFO_LEN];  ///< 事件
    char sSerialID[NET_MAX_MAC_LEN]; ///< 设备序列号
    char sDescrip[NET_MAX_INFO_LEN];  ///< 描述

  };
```

2.1.8 存储设备控制信息

```
/// 存储设备控制
struct SDK_StorageDeviceControl
{
    int iAction;    ///< 见enum SDK_StorageDeviceControlTypes
    int iSerialNo;  ///< 磁盘序列号
    int iPartNo;    ///< 分区号
    int iType;      ///< enum SDK_StorageDeviceClearTypes或者SDK_FileSystemDriverTypes
    int iPartSize[2/*MAX_DRIVER_PER_DISK*/];      ///< 各个分区的大小
};
```

2.1.9 RTSP 信息

```
//RTSP
struct SDK_NetRTSPConfig
{
    bool bServer;
    bool bClient;
    SDK_RemoteServerConfig Server;          ///< 服务器模式
    SDK_RemoteServerConfig Client;          ///< 客户端模式
};
```

2.1.10 互信互通

```
//互信互通
struct SDK_CONFIG_NET_MEGA
{
    bool bEnable;
    bool bNetManEnable;
    CONFIG_IPAddress ServerIP;
    int iServerPort;
    char sDeviceId[32];
    char sUserName[24];
    char sPasswd[32];
    int iMaxCon;
    int iVideoPort;
    int iAudioPort;
    int iMsgPort;
```

```
    int iUpdatePort;
};
```

2.1.11 新望平台

```
// 新望平台
struct SDK_CONFIG_NET_XINGWANG
{
    bool bEnable;
    bool bSyncTime;
    bool bSubStream;
    CONFIG_IPAddress ServerIP;
    int iServerPort;
    int iDownLoadPort;
    char sPasswd[32];
    char szSID[32];
};
```

2.1.12 视搜平台

```
// 视搜平台
struct SDK_CONFIG_NET_SHISOU
{
    bool bEnable;
    SDK_RemoteServerConfig Server;
    char szSID[NET_MAX_USERNAME_LENGTH];
};
```

2.1.13 VVEYE 平台

```
// VVEYE平台
struct SDK_CONFIG_NET_VVEYE
{
    bool bEnable;
    bool bCorpEnable;          //只有在使用企业服务器时才需要设置Server
    SDK_RemoteServerConfig Server;
    char szDeviceName[NET_MAX_USERNAME_LENGTH];
};
```

2.1.14 媒体包及包信息

```
enum SERIAL_TYPE
{
    RS232 = 0,
    RS485 = 1,
};

enum MEDIA_PACK_TYPE
{
    FILE_HEAD = 0,              // 文件头
    VIDEO_I_FRAME = 1,          // 视频I帧
    VIDEO_B_FRAME =     2,          // 视频B帧
    VIDEO_P_FRAME = 3,          // 视频P帧
    VIDEO_BP_FRAME = 4,         // 视频BP帧
    VIDEO_BBP_FRAME     = 5,    // 视频B帧B帧P帧
    VIDEO_J_FRAME = 6,          // 图片帧
    AUDIO_PACKET = 10,          // 音频包
};

typedef struct
{
    int         nPacketType;            // 包类型,见MEDIA_PACK_TYPE
    char*       pPacketBuffer;          // 缓存区地址
    unsigned int   dwPacketSize;            // 包的大小

    // 绝对时标
    int         nYear;                  // 时标:年
    int         nMonth;                 // 时标:月
    int         nDay;                   // 时标:日
    int         nHour;                  // 时标:时
    int         nMinute;                // 时标:分
    int         nSecond;                // 时标:秒
    unsigned int   dwTimeStamp;             // 相对时标低位，单位为毫秒
    unsigned int   dwTimeStampHigh;     //相对时标高位，单位为毫秒
    unsigned int    dwFrameNum;         //帧序号
    unsigned int    dwFrameRate;        //帧率
    unsigned short uWidth;          //图像宽度
    unsigned short uHeight;         //图像高度
    unsigned int    Reserved[6];        //保留
} PACKET_INFO_EX;
```

```
struct SDK_OEMInfo
{
	int nOEMID;			//OEM ID
	char sCompanyName[NET_MAX_USERNAME_LENGTH]; //公司名
	char sTel[NET_MAX_USERNAME_LENGTH];		//电话
	char sAddr[NET_MAX_USERNAME_LENGTH];		//地址
};
typedef struct __TransComChannel//透明窗口
{
	SERIAL_TYPE TransComType;//SERIAL_TYPE
	unsigned int baudrate;
	unsigned int databits;
	unsigned int stopbits;
	unsigned int parity;
} TransComChannel;
typedef enum SDK_State_Type
{
	DEV_STATE_DDNS=0,
};


//摄象机参数.....

//曝光配置
struct SDK_ExposureCfg
{
	int  level;	//曝光等级
	unsigned int leastTime;//自动曝光时间下限或手动曝光时间，单位微秒
	unsigned int mostTime; //自动曝光时间上限，单位微秒
};

//增益配置
struct SDK_GainCfg
{
	int gain;	//自动增益上限(自动增益启用)或固定增益值
	int autoGain;//自动增益是否启用，0:不开启  1:开启
};

//网络摄像机配置
struct SDK_CameraParam
{
	unsigned int whiteBalance;		//白平衡
```

```
    unsigned int dayNightColor;       //日夜模式，取值有彩色、自动切换和黑白
    int elecLevel;            //参考电平值
    unsigned int apertureMode;        //自动光圈模式
    unsigned int BLCMode;             //背光补偿模式
    SDK_ExposureCfg exposureConfig;//曝光配置
    SDK_GainCfg    gainConfig;    //增益配置
};

//所有摄象机配置
struct SDK_AllCameraParam
{
    SDK_CameraParam vCameraParamAll[NET_MAX_CHANNUM];   //所有的通道
};

//曝光能力级
struct SDK_CameraAbility
{
    int  count;     //支持曝光速度数量
    unsigned int speeds[CAMERAPARA_MAXNUM]; //曝光速度
    int  reserve[15];//保留
};
```

2.1.15 本地播放控制

```
//本地播放控制
enum SDK_LoalPlayAction
{
    SDK_Local_PLAY_PAUSE,       /*<! 暂停播放*/
    SDK_Local_PLAY_CONTINUE,        /*<! 继续正常播放*/
    SDK_Local_PLAY_FAST,          /*<! 加速播放*/
    SDK_Local_PLAY_SLOW,          /*<! 减速播放*/

};
```

2.1.16 主动服务

```
//主动服务回调数据
struct H264_DVR_ACTIVEREG_INFO
{
    char deviceSarialID[64];//设备序列号，如果大于64位则赋值0
    H264_DVR_DEVICEINFO deviceInfo;//设备信息
};
```

2.1.17 子连接类型

```
enum SubConnType
{
    conn_realTimePlay=1,
    conn_talk,
    conn_playback
};
```

2.1.18 连接类型

```
enum SocketStyle
{
    TCPSOCKET=0,
    UDPSOCKET,
    SOCKETNR
};
```

2.1.19 搜索协议类型

```
enum SDK_TransferProtocol_V2
{
    SDK_TRANSFER_PROTOCOL_NETIP,  //NETIP协议
    SDK_TRANSFER_PROTOCOL_ONVIF,  //ONVIF协议
    SDK_TRANSFER_PROTOCOL_NR_V2=4, //所有协议
};
```

3 接口定义

3.1 SDK 初始化

1. `H264_DVR_API long H264_DVR_GetLastError();`

- 函数说明：返回函数执行失败代码，当调用下面的接口失败时，可以用该函数获取失败的代码，具体错误代码参见错误类型代号说明
- 参数说明：无
- 返回值：返回错误类型码

```
typedef void (__stdcall *fDisConnect)(long lLoginID, char *pchDVRIP, long nDVRPort, unsigned long dwUser);
```

2. `H264_DVR_API long H264_DVR_Init(fDisConnect cbDisConnect, unsigned long dwUser);`

- 函数说明：初始化 SDK, 在所有的 SDK 函数之前调用
- 参数说明：

cbDisConnect

断线回调函数，回调出当前网络已经断开的设备，对调用 SDK 的 H264_DVR_Logout ()函数主动断开的设备不回调，设置为 0 时禁止回调

[in]dwUser

用户数据

- 回调函数参数说明：

lLoginID

H264_DVR_Login 的返回值

pchDVRIP

设备 IP

nDVRPort

端口

dwUser

用户数据，就是上面输入的用户数据

- 返回值：成功返回 TRUE，不成功返回 FALSE
- 相关函数：H264_DVR_Cleanup

3. `CLIENT_API void H264_DVR_Cleanup ();`

- 函数说明：清空 SDK, 释放占用的资源，在所有的 SDK 函数之后调用。
- 参数：无

- 返回值：无
- 相关函数：H264_DVR_Init
- 典型应用：在应用程序关闭时调用

3.2 报警状态获取

```
typedef bool (__stdcall *fMessCallBack)(long lLoginID, char
*pBuf,
                                        unsigned long dwBufLen, long
dwUser);
```

- H264_DVR_API bool H264_DVR_SetDVRMessCallBack(fMessCallBack cbAlarmcallback, unsigned long lUser);
- 函数说明：设置设备消息回调函数，用来得到设备当前状态信息，与调用顺序无关，SDK 默认不回调，此回调函数必须先调用报警消息订阅接口 H264_DVR_SetupAlarmChan 才有效，同时需要说明的是针对目前定义的报警，是每秒回调设备当前的报警信息
- 参数说明：

cbAlarmcallback

消息回调函数，可以回调设备的状态，如报警状态可以通过此回调获取；当设置为 0 时表示禁止回调

[in] lUser

用户数据

回调函数参数说明：

lLoginID

H264_DVR_Login 的返回值

pBuf

具体信心见 SDK_AlarmInfo

dwBufLen

pBuf 的长度，（单位字节）

dwUser

回调的用户数据，就是上面输入的用户数据

- 返回值：TRUE 回调函数执行正确，FALSE 执行错误
- 相关函数：H264_DVR_SetupAlarmChan、H264_DVR_CloseAlarmChan

4. H264_DVR_API long H264_DVR_SetupAlarmChan(long lLoginID);

- 函数说明：开始对某个设备订阅消息，用来设置是否需要对设备消息回调，得到的消息从 H264_DVR_SetDVRMessCallBack 的设置值回调出来。
- 参数说明：

[in]lLoginID

H264_DVR_Login 的返回值

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_SetDVRMessCallBack，H264_DVR_CloseAlarmChan

典型应用：在设备连接后调用本函数进行消息订阅.

5. H264_DVR_API bool H264_DVR_CloseAlarmChan(long lLoginID);

- 函数说明：停止对某个设备侦听消息
- 参数说明：

[in]lLoginID

H264_DVR_Login 返回值

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_SetupAlarmChan
- 典型应用：参见 demo 程序

3.3 设备注册

6. H264_DVR_API long H264_DVR_Login (char *sDVRIP, unsigned short wDVRPort, char *sUserName, char *sPassword, LPH264_DVR_DEVICEINFO lpDeviceInfo, int *error , SocketStyle socketTyle=TCPSOCKET);

- 函数说明：注册用户到设备，当设备端把用户设置为复用（设备默认的用户如 admin,不能设置为复用），则使用该帐号可以多次向设备注册。
- 参数说明：

[in] sDVRIP

设备 IP

[in] wDVRPort

设备端口

[in] sUserName

用户名

[in] sPassword

用户密码

[out] lpDeviceInfo

设备信息,属于输出参数

[out] error

(当函数返回成功时,该参数的值无意义)，返回登录错误码：

[in] socketTyle

登入类型 参见：SocketStyle

- 返回值：失败返回 0，成功返回设备 ID，登录成功之后对设备的操作都可以通过此值(设备句柄)对应到相应的设备。
- 相关函数：H264_DVR_Logout

- 典型应用：在初始化后就可以调用本接口注册到指定的设备，成功后将返回设备句柄，给相关的函数调用

7. H264_DVR_API long H264_DVR_LoginEx(char *sDVRIP, unsigned short wDVRPort, char *sUserName, char *sPassword, LPH264_DVR_DEVICEINFO lpDeviceInfo, int nType, int *error);

- 函数说明：注册用户到设备的扩展接口，支持一个用户指定登陆的客户端类型
- 参数说明：增加扩展参数

[in] nType

设备支持的能力，值为 2 表示主动侦听模式下的用户登陆。(车载 dvr 登录)

```
enum LoginType
{
    LOGIN_TYPE_GUI,          ///< 本地GUI登陆
    LOGIN_TYPE_CONSOLE,      ///< 控制台登陆
    LOGIN_TYPE_WEB,          ///< WEB登陆
    LOGIN_TYPE_SNS,          ///< SNS登陆
    LOGIN_TYPE_MOBIL,        ///< 移动终端登陆
    LOGIN_TYPE_NETKEYBOARD,  ///< 网络键盘登陆
    LOGIN_TYPE_SERVER,       ///< 中心服务器登陆
    LOGIN_TYPE_AUTOSEARCH,   ///< IP自动搜索工具登陆
    LOGIN_TYPE_UPGRADE,      ///< 升级工具登陆
    LOGIN_TYPE_MEGAEYE,      ///< 全球眼登陆
    LOGIN_TYPE_NR,           ///< 登陆类型
};
```

- 返回值：失败返回 0，成功返回设备 ID，登录成功之后对设备的操作都可以通过此值（设备句柄）对应到相应的设备
- 相关函数：H264_DVR_Logout
- 典型应用：升级工具等的登陆。

8. H264_DVR_API long H264_DVR_Logout(long lLoginID)

- 函数说明：注销设备用户
- 参数说明：

[in]lLoginID

H264_DVR_Login 的返回值

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_Login
- 典型应用：当需要设备主动断开时调用；

3.4 实时监视

9. H264_DVR_API long H264_DVR_RealPlay(long lLoginID, LPH264_DVR_CLIENTINFO lpClientInfo);

- 函数说明：启动实时监视
- 参数说明：

[in]lLoginID

H264_DVR_Login 的返回值

[in] lpClientInfo

实时监视参数

- 返回值：失败返回 小于等于 0，小于 0 可以用 H264_DVR_GetLastError 获得错误类型，成功返回实时监视 ID(实时监视句柄)，将作为相关函数的参数。
- 相关函数：H264_DVR_StopRealPlay，H264_DVR_SetRealDataCallBack
- 典型应用：根据登录时获取到的设备信息，调用本接口，就可以打开任何有效的一路实时监视，并通过 H264_DVR_SetRealDataCallBack 设备的回调得到原始数据（注：如果对 lpClientInfo 中的 hWnd 赋值就可以完成播放，而不需要对回调出来的数据送解码播放），成功返回实时监视 ID，用于以下对本监视通道的控制和操作；

10. H264_DVR_API bool H264_DVR_StopRealPlay(long lRealHandle,void*hWnd=NULL);

- 函数说明：停止实时监视
- 参数说明：

[in]lRealHandle

H264_DVR_RealPlay 的返回值

[in]hWnd

用于停止相应窗口的解码播放；默认值 NULL 停止所有窗口的解码播放

（网络和解码分开调用时这个值没有意义）

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_RealPlay
- 典型应用：关闭实时监视

11. H264_DVR_API long H264_DVR_PauseRealPlay(long lRealHandle, bool bPause);

- 函数说明：暂停/继续实时监视
- 参数说明：

[in]lRealHandle

H264_DVR_RealPlay 的返回值

[in] bPause

暂停使能，其中 0 表示继续，1 表示暂停

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_RealPlay
- 典型应用：暂停实时监视

```
//原始数据回调原形
typedef int(__stdcall *fRealDataCallBack) (long lRealHandle, long
dwDataType, unsigned char *pBuffer,long lbufsize,long dwUser);
```

12. `H264_DVR_API bool H264_DVR_SetRealDataCallBack(long lRealHandle,fRealDataCallBack cbRealData, long dwUser);`

- 函数说明：设置实时监视数据回调，给用户提供设备流出的数据
- 参数说明：

[in]lRealHandle
H264_DVR_RealPlay 的返回值
cbRealData
回调函数，用于传出设备流出的实时数据
[in]dwUser
用户数据

回调函数参数说明：

lRealHandle
H264_DVR_RealPlay 的返回值
dwDataType
暂时可以不需要判断
pBuffer
回调数据，根据数据类型的不同每次回调不同的长度的数据，除类型 0， 其他数据类型都是按帧，每次回调一帧数据，
dwBufSize
回调数据的长度，（单位字节）.
dwUser
用户数据，就是上面输入的用户数据

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_RealPlay、H264_DVR_StopRealPlay

```
//原始数据回调原形
typedef int(CALL_METHOD *fRealDataCallBack_V2) (long lRealHandle,
const PACKET_INFO_EX *pFrame, unsigned int dwUser);
```

13. `H264_DVR_API bool CALL_METHOD H264_DVR_SetRealDataCallBack_V2(long lRealHandle,fRealDataCallBack_V2 cbRealData, long dwUser);`

- 函数说明：设置实时监视数据回调，给用户提供设备流出的数据
- 参数说明：

[in]lRealHandle
H264_DVR_RealPlay 的返回值
cbRealData
回调函数，用于传出设备流出的实时数据
[in]dwUser

用户数据

回调函数参数说明：

lRealHandle
H264_DVR_RealPlay 的返回值
pFrame
结构参考 PACKET_INFO_EX
dwUser
用户数据，就是上面输入的用户数据

- 返回值：成功返回 TRUE，失败返回 FALSE

14. H264_DVR_API bool H264_DVR_DelRealDataCallBack(long lRealHandle,fRealDataCallBack cbRealData, long dwUser);

- 函数说明：清除回调函数,该函数需要在 H264_DVR_StopRealPlay 前调用
- 参数说明：

[in]lRealHandle
H264_DVR_RealPlay 的返回值
cbRealData
回调函数，用于传出设备流出的实时数据
[in]dwUser
用户数据

- 返回值：成功返回 TRUE，失败返回 FALSE

15. H264_DVR_API bool CALL_METHOD H264_DVR_DelRealDataCallBack_V2(long lRealHandle,fRealDataCallBack_V2 cbRealData, long dwUser);

- 函数说明：清除回调函数,该函数需要在 H264_DVR_StopRealPlay 前调用
- 参数说明：

[in]lRealHandle
H264_DVR_RealPlay 的返回值
cbRealData
回调函数，用于传出设备流出的实时数据
[in]dwUser
用户数据

- 返回值：成功返回 TRUE，失败返回 FALSE

3.5 回放和下载

16. H264_DVR_API long H264_DVR_FindFile(long lLoginID, H264_DVR_FINDINFO* lpFindInfo, H264_DVR_FILE_DATA *lpFileData, int lMaxCount, int *findcount, int waittime = 5000);

- 函数说明：查询录像文件
- 参数说明：

[in]lLoginID
H264_DVR_Login 的返回值

[in] lpFindInfo
查询条件 H264_DVR_FINDINFO

[out]nriFileinfo
返回的录像文件信息，是一个 H264_DVR_FILE_DATA 结构数组

[in]maxlen
nriFileinfo 缓冲的最大长度；(单位字节，建议在
100-200*sizeof(H264_DVR_FILE_DATA)之间)

[out]filecount
返回的文件个数，属于输出参数最大只能查到缓冲满为止的录像记录；

[in]waittime
等待时间

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_Login、H264_DVR_PlayBackByName、H264_DVR_StopPlayBack，H264_DVR_PlayBackControl，H264_DVR_GetFileByName
- 典型应用：在回放之前需要先调用本接口查询录像记录，当根据输入的时间段查询到的录像记录信息大于定义的缓冲区大小，则只返回缓冲所能存放的录像记录，可以根据需要继续查询

17. H264_DVR_API long H264_DVR_FindFileByTime(long lLoginID, SDK_SearchByTime* lpFindInfo, SDK_SearchByTimeResult *lpFileData, int waittime = 10000);

- 函数说明：按时间查询文件
- 参数说明：

[in]lLoginID
H264_DVR_Login 的返回值

[in] lpFindInfo
查询条件 SDK_SearchByTime

[out] lpFileData
返回的录像文件信息，是一个 SDK_SearchByTimeResult 结构数组，外部开内存

[in]waittime
等待时间，单位 ms

- 返回值：成功返回 TRUE，失败返回 FALSE

```
typedef void(__stdcall *fDownLoadPosCallBack) (long lPlayHandle,
long lTotalSize, long lDownLoadSize, long dwUser)
```

18. `H264_DVR_API long H264_DVR_PlayBackByName(long lLoginID, H264_DVR_FILE_DATA *sPlayBackFile, fDownLoadPosCallBack cbDownLoadPos, fRealDataCallBack fDownLoadDataCallBack, long dwDataUser);`

- 函数说明：网络回放，需要说明的是，用户登录一台设备后，每通道同一时间只能播放一则录像，不能同时播放同一通道的多条记录。
- 参数说明：

[in]lLoginID
H264_DVR_Login 的返回值

[in] sPlayBackFile
录像文件信息，由 H264_DVR_FindFile 返回
[in] cbDownLoadPos
进度回调函数
[in] fDownLoadDataCallBack
原始数据回调函数
[in]dwUserData
用户自定义数据

- 回调函数说明：

lPlayHandle
H264_DVR_PlayBackByName 的返回值
dwTotalSize
指本次播放总大小，单位为 KB
dwDownLoadSize
指已经播放的大小，单位为 KB，当其值为-1 时表示本次回放结束
dwUser
用户数据，就是上面输入的用户数据

- 返回值：成功返回网络回放 ID，失败返回 0
- 相关函数：H264_DVR_Login、H264_DVR_FindFile，H264_DVR_StopPlayBack，H264_DVR_PlayBackControl

```
typedef int(CALL_METHOD *fRealDataCallBack_V2) (long lRealHandle,
const PACKET_INFO_EX *pFrame, unsigned int dwUser);
```

19. `H264_DVR_API long H264_DVR_PlayBackByName_V2(long lLoginID, H264_DVR_FILE_DATA *sPlayBackFile, fDownLoadPosCallBack cbDownLoadPos, fRealDataCallBack_V2 fDownLoadDataCallBack, long dwDataUser);`

- 函数说明：网络回放，按文件名回放
- 参数说明：

[in]lLoginID
H264_DVR_Login 的返回值
[in] sPlayBackFile
录像文件信息，由 H264_DVR_FindFile 返回
[in] cbDownLoadPos
目前只用于播放结束回调（获取位置请调用 H264_DVR_GetPlayPos 接口）
[in] fRealDataCallBack_V2
原始数据回调函数
[in]dwUserData
用户自定义数据

- 回调函数说明：

lRealHandle
H264_DVR_RealPlay 的返回值
pFrame
参考结构 PACKET_INFO_EX
dwUser
用户数据，就是上面输入的用户数据

- 返回值：成功返回网络回放 ID，失败返回 0

20. H264_DVR_API long H264_DVR_PlayBackByTime(long lLoginID, H264_DVR_FINDINFO* lpFindInfo, fDownLoadPosCallBack cbDownLoadPos, fRealDataCallBack fDownLoadDataCallBack, long dwDataUser);

- 函数说明：（老的接口，请用下面 H264_DVR_PlayBackByTimeEx）按时间进行录像回放
- 参数说明：

[in]lLoginID
H264_DVR_Login 的返回值
[in] lpFindInfo
查询条件 H264_DVR_FINDINFO
[in] cbDownLoadPos
进度回调函数
[in] fDownLoadDataCallBack
原始数据回调函数
[in]dwUserData
用户自定义数据

- 返回值：成功返回网络回放 ID，失败返回 0

21. H264_DVR_API long H264_DVR_PlayBackByTimeEx(long lLoginID,

H264_DVR_FINDINFO*lpFindInfo,fRealDataCallBack cbDownLoadDataCallBack, long dwDataUser,fDownLoadPosCallBack cbDownLoadPos, long dwPosUser);

- 函数说明：按时间进行录像回放
- 参数说明：

[in]lLoginID
H264_DVR_Login 的返回值
[in] lpFindInfo
查询条件 H264_DVR_FINDINFO
[in] cbDownLoadDataCallBack
回放视数据回调
[in] dwDataUser
数据回调自定义数据

[in] cbDownLoadPos
目前只用于播放结束回调（获取位置请调用 H264_DVR_GetPlayPos 接口）
[in] dwPosUser
用户自定义数据

- 返回值：成功返回网络回放 ID，失败返回 0

22. H264_DVR_API bool H264_DVR_StopPlayBack(long lPlayHandle);

- 函数说明：网络回放停止
- 参数说明：

[in]lPlayHandle
回放句柄，如 H264_DVR_PlayBackByName 的返回值

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_PlayBackByName
- 典型应用：输入上一接口返回的播放 ID，调用本接口就可以停止控制。

23. H264_DVR_API long H264_DVR_GetFileByName(long lLoginID,H264_DVR_FILE_DATA *sPlayBackFile,char *sSavedFileName, fDownLoadPosCallBack cbDownLoadPos = NULL, long dwDataUser = NULL);

- 函数说明：按文件下载录像文件， 通过查询到的文件信息下载
- 参数说明：

[in]lLoginID
H264_DVR_Login 的返回值
[in] sPlayBackFile

录像文件信息指针

[in]sSavedFileName

要保存的录像文件名，全路径

cbDownLoadPos

下载进度回调函数，可以为空，用户自己调用 H264_DVR_GetDownloadPos 得到进度

[in]dwUserData

下载进度回调用户自定义数据

- 下载进度回调函数参数说明：参见 H264_DVR_PlayBackByName

- 返回值：成功返回下载 ID，失败返回 0
- 相关函数：H264_DVR_StopGetFile、H264_DVR_GetDownloadPos
- 典型应用：根据上面查询的记录，就可以将录像保存到指定的文件，下载进度回调与回放进度类似

24. H264_DVR_API long H264_DVR_GetFileByTime(long lLoginID, H264_DVR_FINDINFO* lpFindInfo, char *sSavedFileDIR, fDownLoadPosCallBack cbDownLoadPos = NULL , bool bMerge = false)

- 函数说明：按时间下载录像文件， 通过查询到的文件信息下载
- 参数说明：

[in]lLoginID

H264_DVR_Login 的返回值

[in] lpFindInfo

查询条件 H264_DVR_FINDINFO

[in] sSavedFileDIR

下载文件保存的目录

cbDownLoadPos

下载进度回调函数，可以为空，用户自己调用 H264_DVR_GetDownloadPos 得到进度

[in] bMerge

是否将下载的文件合并成一个文件

- 返回值：成功返回下载 ID，失败返回 0

25. H264_DVR_API bool H264_DVR_StopGetFile(long lFileHandle);

- 函数说明：停止下载录像文件
- 参数说明：

[in]lFileHandle

H264_DVR_GetFileByName 的返回值

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_GetFileByName、H264_DVR_GetDownloadPos
- 典型应用：根据需要可以等文件下载完了关闭下载，也可以下载到一部分停止下载；

26. H264_DVR_API int H264_DVR_GetDownloadPos(long lFileHandle);

- 函数说明：获得下载录像的当前位置，可以用于不需要实时显示下载进度的接口，与下载回调函数的功能类似
- 参数说明：

[in]lFileHandle

H264_DVR_GetFileByName 的返回值

- 返回值：成功返回 pos(百分比)
- 相关函数：H264_DVR_GetFileByName、H264_DVR_StopGetFile
- 典型应用：用于不打算通过回调计算进度，可定时调用本接口获取当前进度；

3.6 回放控制

27. H264_DVR_API bool H264_DVR_PlayBackControl(long lPlayHandle, long lControlCode,long lCtrlValue);

- 函数说明：网络回放暂停与恢复以及进度控制
- 参数说明：

[in]lPlayHandle

回放句柄，如 H264_DVR_GetFileByName 的返回值

[in] lControlCode

控制类型

```
enum SEDK_PlayBackAction
{
   SDK_PLAY_BACK_PAUSE,        /*<! 暂停回放*/
   SDK_PLAY_BACK_CONTINUE,     /*<! 继续回放*/
   SDK_PLAY_BACK_SEEK,         /*<! 回放定位，时间s为单位 */
   SDK_PLAY_BACK_FAST,          /*<! 加速回放 */
   SDK_PLAY_BACK_SLOW,          /*<! 减速回放 */
   SDK_PLAY_BACK_SEEK_PERCENT, /*<! 回放定位百分比 */
};
```

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_PlayBackByName、H264_DVR_StopPlayBack
- 典型应用：对已经打开的播放进行暂停和恢复控制

3.7 云台控制

28. H264_DVR_API bool H264_DVR_PTZControl(long lLoginID,int nChannelNo, long lPTZCommand, bool bStop = false, long lSpeed = 4)

- 函数说明：云台控制

- 参数说明：

[in]lLoginID

H264_DVR_Login 的返回值

[in] nChannelNo

控制的设备通道号

[out] lPTZCommand

控制类型。PTZ_ControlType

[in] bStop

是否是停止

[out] lSpeed

速度，默认 4

- 返回值：成功返回 TRUE，失败返回 FALSE。
- 相关函数：H264_DVR_Login，H264_DVR_RealPlay
- 典型应用：控制云台，但是必须在当前通道打开的情况下使用。

29. H264_DVR_API bool H264_DVR_PTZControlEx(long lLoginID,int nChannelNo, long lPTZCommand, long lParam1, long lParam2, long lParam3, bool bStop = false)

- 函数说明：扩展云台设置，包括预置点设置，巡航路线，快速定位等
- 参数说明：

[in]lLoginID

H264_DVR_Login 的返回值

[in] nChannelNo

控制的设备通道号

[in] lPTZCommand

控制类型。PTZ_ControlType

[in] lParam1, lParam2, lParam3

扩展云台设置参数，根据扩展功能不同意义不同：

- 设置，删除，转到预置点时：lParam1 为预置点值
- 加入预置点到巡航，删除巡航中预置点时：lParam1 为巡航线路值，lParam2 为预置点值

- 开始巡航，停止巡航，清除巡航线路时：lParam1 为巡航线路值
- 云台方向设置时：lParam1 为水平步长，lParam2 为垂直步长

[in] bStop

是否是停止

- 返回值:成功返回 TRUE，失败返回 FALSE。
- 典型应用：扩展云台控制，但是必须在当前通道打开的情况下使用。

3.8 系统配置

30. H264_DVR_API long H264_DVR_GetDevConfig(long lLoginID, unsigned long dwCommand, int nChannelNO, char * lpOutBuffer, unsigned long dwOutBufferSize, unsigned long* lpBytesReturned,int waittime = 1000);

- 函数说明: 获取设备配置。
- 参数说明 ：

[in]lLoginID

H264_DVR_Login 的返回值

[in]dwCommand

配置类型 具体定义见数据结构定义中的SDK_CONFIG_TYPE

[in] nChannelNO

配置通道号，-1表示所有通道

[out] lpOutBuffer

存放输出参数的缓冲区， 根据不同的类型， 输出不同的配置结构， 具体见数据结构定义中各配置结构

[in]dwOutBufferSize

输入缓冲区的大小，（单位字节）.

[out]lpBytesReturned

实际返回的缓冲区大小,对应配置结构的大小，（单位字节）.

[in]waittime

等待时间

- 返回值：大于 0 成功，小于 0 失败 （可根据错误类型查找）。

31. H264_DVR_API long H264_DVR_SetDevConfig(long lLoginID, unsigned long dwCommand, int nChannelNO, char * lpInBuffer, unsigned long dwInBufferSize, int waittime = 1000);

- 函数说明： 获取设备配置。
- 参数说明 ：

[in]lLoginID
H264_DVR_Login 的返回值
[in]dwCommand
配置类型 具体定义见数据结构定义中的SDK_CONFIG_TYPE
[in]nChannelNO
配置通道号，-1表示所有通道
[in]lpInBuffer
存放输入参数的缓冲区， 根据不同的类型， 输入不同的配置结构， 具体见数据结构定义中各配置结构
[in]dwInBufferSize
输入缓冲区的大小，（单位字节）.
[in]waittime
等待时间

- 返回值：大于 0 成功，小于 0 失败 （可根据错误类型查找）。

32. H264_DVR_API long H264_DVR_SetConfigOverNet(unsigned long dwCommand, int nChannelNO, char * lpInBuffer, unsigned long dwInBufferSize, int waittime = 1000);

- 函数说明：跨网段设置设备配置。目前只支持对网络配置进行设置
- 参数说明：

[in]dwCommand
配置类型 具体定义见数据结构定义中的SDK_CONFIG_TYPE
[in]nChannelNO
配置通道号，-1表示所有通道
[in]lpInBuffer
存放输入参数的缓冲区， 根据不同的类型， 输入不同的配置结构， 具体见数据结构定义中各配置结构
[in]dwInBufferSize
输入缓冲区的大小，（单位字节）.
[in]waittime
等待时间

- 返回值：TRUE 表示成功，FALSE 表示失败。

3.9 日志管理

33. H264_DVR_API bool H264_DVR_FindDVRLog(long lLoginID, SDK_LogSearchCondition *pFindParam, SDK_LogList *pRetBuffer, long lBufSize, int waittime = 2000);

- 函数说明：查询日志
- 参数说明：

[in]lLoginID
H264_DVR_Login 的返回值
[in] pFindParam

日志查询条件
[in] pRetBuffer
返回日志信息
[in] lBufSize
日志返回长度
[in] waittime
等待时间

3.10 远程控制

34. H264_DVR_API bool H264_DVR_ControlDVR(long lLoginID, int type, int waittime = 2000)

- 函数说明：重启和清除日志
- 参数说明：

[in]lLoginID
H264_DVR_Login 的返回值

[in] type

0 重启设备，1 清除日志

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：

```
typedef void(__stdcall *fUpgradeCallBack) (long lLoginID, long
   lUpgradechannel, int nTotalSize, int nSendSize, long
   dwUser);
```

35. `H264_DVR_API long H264_DVR_Upgrade(long lLoginID, char *sFileName, int nType = 0, fUpgradeCallBack cbUpgrade = NULL, long dwUser = 0);;`

- 函数说明：设置对前端设备网络升级程序
- 参数说明：

[in]lLoginID

H264_DVR_Login 的返回值

[in] sFileName

要升级的文件名

[in] nType

要升级的文件类型

```
enum UpgradeTypes
{
    UPGRADE_TYPES_SYSTEM,   ///< 升级系统
    UPGRADE_TYPES_NR,
};
```

//回调说明

fUpgradeCallBack

回调升级进度，其中 lUpgradechannel 为升级句柄

nTotalSize

为升级文件的总长度，(单位字节)

nSendSize

为已升级的长度，(单位字节)

[in]dwUser

用户数据

- 返回值：成功返回升级句柄 ID，失败返回 0
- 相关函数：H264_DVR_GetUpgradeState，H264_DVR_CloseUpgradeHandle
- 典型应用：设置远程程序的升级，返回程序升级句柄

36. `H264_DVR_API long H264_DVR_CloseUpgradeHandle(long lUpgradeHandle);`

- 函数说明：停止升级
- 参数说明：

[in]lUpgradeID

升级句柄 ID

- 返回值：成功返回 TRUE，失败返回 FALSE
- 相关函数：H264_DVR_Upgrade
- 典型应用：停止升级

37. `H264_DVR_API int H264_DVR_GetUpgradeState(long lUpgradeHandle)`

- 函数说明：获取升级状态
- 参数说明：

  0- [in]lUpgradeID

  升级句柄 ID

  返回值：1 成功，2 正在升级 3 失败
- 相关函数: H264_DVR_Upgrade，H264_DVR_CloseUpgradeHand

38. `H264_DVR_API bool H264_DVR_SearchDevice(char* szBuf, int nBufLen, int* pRetLen, int nSearchTime);`

- 函数说明：搜索局域网内设备信息
- 参数说明：

  0- *[in]* szBuf

  接收搜索到的设备信息缓冲

  1- *[in]* nBufLen

  接收搜索到的设备信息缓冲的长度

  2- *[in]* pRetLen

  实际搜索到的设备信息的长度，用来判断缓冲大小是否合适

  3- *[in]* nSearchTime

  指定设备搜索的总时间，超过时间则认为搜索失败

  返回值：1 成功，0 失败
- 相关函数:

3.11 语音对讲

```
// 语音对讲的音频数据回调函数原形
typedef void (__stdcall *pfAudioDataCallBack)(long lVoiceHandle, char *pDataBuf,
                                              long dwBufSize, char byAudioFlag, long dwUser);
```

39. `H264_DVR_API long H264_DVR_StartVoiceCom_MR (long lLoginID, pfAudioDataCallBack pVcb, long dwDataUser);`

- 函数说明：开启语音对讲，负责数据转发
- 参数说明：

0- *[in]* lLoginID

设备登陆句柄：H264_DVR_Login 返回值

1- *[in]* pVcb

从设备接收到的语音对讲数据回调

2- *[in]* dwDataUser

接收数据对象

返回值：> 0 对讲句柄 <= 0 失败

- 相关函数:

H264_DVR_VoiceComSendData H264_DVR_StopVoiceCom H264_DVR_SetTalkMode

40. H264_DVR_API bool H264_DVR_VoiceComSendData (long lVoiceHandle, char *pSendBuf, long lBufSize);

- 函数说明：转发 PC 采集到的语音对讲数据
- 参数说明：

0- *[in]* lVoiceHandle

对讲句柄：H264_DVR_StartVoiceCom_MR 返回值

1- *[in]* pSendBuf

从 PC 采集到的音频数据

2- *[in]* lBufSize

音频数据长度

- 返回值：1 成功 0 失败
- 相关函数：

H264_DVR_StartVoiceCom_MR H264_DVR_StopVoiceCom H264_DVR_SetTalkMode

41. H264_DVR_API bool H264_DVR_StopVoiceCom (long lVoiceHandle);

- 函数说明：停止语音对讲
- 参数说明：

0- *[in]* lVoiceHandle

对讲句柄：H264_DVR_StartVoiceCom_MR 返回值

- 返回值：1 成功 0 失败
- 相关函数：

H264_DVR_StartVoiceCom_MR H264_DVR_VoiceComSendData H264_DVR_SetTalkMode

42. `H264_DVR_API bool H264_DVR_SetTalkMode (long lLoginID, SDK_AudioInFormatConfig* pTalkMode);`

- 函数说明：指定设备的语音对讲模式
- 参数说明：

0- *[in]* lLoginID

设备登陆句柄：`H264_DVR_Login` 返回值

1- *[in]* pTalkMode

对讲模式结构体 参见：SDK_AudioInFormatConfig

- 返回值：1 成功 0 失败
- 相关函数：

H264_DVR_StartVoiceCom_MR H264_DVR_VoiceComSendData H264_DVR_StopVoiceCom

3.12 录像模式设置

43. `H264_DVR_API bool H264_DVR_StartDVRRecord(long lLoginID, int nChannelNo ,long lRecordType);`

- 函数说明：该接口是为了方便手动开启录像而增加，也可以通过系统配置设置接口（`H264_DVR_SetDevConfig`）设置录像为手动模式
- 参数说明：

0- *[in]* lLoginID

设备登陆句柄：`H264_DVR_Login` 返回值

1- *[in]* nChannelNo

通道号，`-1` 所有通道

2- `[in]` lRecordType

录像类型 参见：SDK_RecordModeTypes

- 返回值：1 成功　0 失败
- 相关函数：

H264_DVR_StopDVRRecord　　H264_DVR_GetDevConfig　　H264_DVR_SetDevConfig

```
44.  H264_DVR_API bool H264_DVR_StopDVRRecord(long lLoginID, int
     nChannelNo);
```

- 函数说明：该接口是为了方便手动关闭录像而增加，也可以通过系统配置设置接口（H264_DVR_SetDevConfig）设置录像为关闭模式
- 参数说明：

0- *[in]* lLoginID

设备登陆句柄：H264_DVR_Login 返回值

1- *[in]* nChannelNo

通道号，-1 所有通道

- 返回值：1 成功　0 失败
- 相关函数：

H264_DVR_StartDVRRecord　　H264_DVR_GetDevConfig　　H264_DVR_SetDevConfig

3.13 设置系统时间

```
45.  H264_DVR_API bool H264_DVR_SetSystemDateTime (long lLoginID,
     SDK_SYSTEM_TIME *pSysTime);
```

- 函数说明：设置系统时间
- 参数说明：

0- *[in]* lLoginID

设备登陆句柄：H264_DVR_Login 返回值

1- *[in]* pSysTime

系统时间　参见：SDK_SYSTEM_TIME

- 返回值：1 成功　0 失败

3.14 获取设置运行状态信息

46. H264_DVR_API bool H264_DVR_GetDVRWorkState(long lLoginID, SDK_DVR_WORKSTATE *pWorkState);

- 函数说明：获取设备工作状态信息
- 参数说明：

  0- *[in]* lLoginID

  设备登陆句柄：H264_DVR_Login 返回值

  1- *[in]* pWorkState

  运行状态 参见：SDK_DVR_WORKSTATE

- 返回值：1 成功 0 失败

3.15 网络键盘

47. H264_DVR_API bool H264_DVR_ClickKey(long lLoginID, SDK_NetKeyBoardData *pKeyBoardData);

- 函数说明：发送网络键盘按键消息
- 参数说明：

  0- *[in]* lLoginID

  设备登陆句柄：H264_DVR_Login 返回值

  1- *[in]* pKeyBoardData

  按键信息 参见：SDK_NetKeyBoardData

- 返回值：1 成功 0 失败

3.16 网络报警

48. H264_DVR_API bool H264_DVR_SendNetAlarmMsg(long lLoginID, SDK_NetAlarmInfo *pAlarmInfo);

- 函数说明:发送网络报警信息
- 参数说明：

  [in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值

[in] pAlarmInfo

网络报警信息　参见：SDK_NetAlarmInfo

- 返回值：1 成功　0 失败

3.17 报警中心

//消息（报警）回调原形

```
typedef bool (__stdcall *fMessCallBack)(long lLoginID, char *pBuf,
                                        unsigned long dwBufLen, long
dwUser);
```

49. H264_DVR_API bool H264_DVR_StartAlarmCenterListen(int nPort, fMessCallBack cbAlarmCenter, unsigned long dwDataUser);

- 函数说明：开启报警中心
- 参数说明：

[in] nPort

端口号

[in] cbAlarmCeter

消息（报警）回调参数

[in] dwDataUser

用户自定义数据

- 返回值：1 成功　0 失败

50. H264_DVR_API bool H264_DVR_StopAlarmCenterListen();

- 函数说明：关闭报警中心
- 返回值：1 成功　0 失败

3.18 磁盘管理

51. H264_DVR_API int H264_DVR_StorageManage(long lLoginID, SDK_StorageDeviceControl *pStorageCtl);

- 函数说明：磁盘管理
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值

[in] pStorageCtl

存储设备控制 参见：SDK_StorageDeviceControl

- 返回值：大于零，成功；小于零，失败；

3.19 抓图

52. H264_DVR_API bool H264_DVR_CatchPic(long lLoginID, int nChannel, char *sFileName);

- 函数说明：抓图
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值

[in]nChannel

控制的设备通道号

[in] sFileName

要保存的图片名，全路径

- 返回值：1 成功 0 失败

53. H264_DVR_API bool H264_DVR_CatchPicInBuffer(long lLoginID, int nChannel, char *pBuffer, int nBufLen, int *pPicLen);

- 函数说明：抓图
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值

[in]nChannel

控制的设备通道号

[out] pBuffer

接受到的图片信息放在缓冲区内

[in] nBufLen

缓冲去内的数据长度

[out] pPicLen

实际得到的数据长度

- 返回值：1 成功 0 失败

3.20 透明 232,485

54. H264_DVR_API bool H264_DVR_SerialWrite(long lLoginID, SERIAL_TYPE nType, char *pBuffer, int nBufLen);

- 函数说明：透明 232，485
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值

[in] nType

类型详见 SERIAL_TYPE

[int] pBuffer

数据缓冲区

[in] nBufLen

数据缓冲的长度

- 返回值：1 成功 0 失败

55. H264_DVR_API bool H264_DVR_SerialRead(long lLoginID, SERIAL_TYPE nType, char *pBuffer, int nBufLen, int *pReadLen);

- 函数说明：透明
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值

[in] nType

类型详见 SERIAL_TYPE

[out] pBuffer

读取数据后缓冲区

[in] *nBufLen*

缓冲区内的数据长度

[out] pReadLen

实际接收到的长度

- 返回值：1 成功 0 失败

3.21 获取 DDNS 信息

- H264_DVR_API int H264_DVR_GetDDNSInfo(SearchMode &searchmode, DDNS_INFO *pDevicInfo, int maxDeviceNum, int &nretNum)
- 函数说明：获取 DDNS 信息
- 参数说明：

[in] searchmode

查询信息

[*out*] pDevicInfo

返回设备信息

[in] maxDeviceNum

最大设备查询数

[in] nretNum

实际获得的设备数

- 返回值：>=0 成功；<0 失败

3.22 支持强迫 I 帧

56. H264_DVR_API bool H264_DVR_MakeKeyFrame(long lLoginID, int nChannel, int nStream);

- 函数说明： 支持强迫 I 帧
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值

[in]nChannel

控制的设备通道号

[in] nStream

码流类型，其中 0 表示主码流，1 表示子码流

- 返回值：1 成功 0 失败

3.23 设置连接设备超时时间和尝试次数

57. H264_DVR_API bool CALL_METHOD H264_DVR_SetConnectTime(long nWaitTime, long nTryTimes);

- 函数说明：设置连接设备超时时间和尝试次数
- 参数说明：

[in] nWaitTime

尝试连接的等待时间，单位 ms 不设置时默认 5000ms，

[in] nTryTimes

尝试连接的次数，不设置时默认 3 次

- 返回值：1 成功 0 失败

3.24 透明串口

//透明串口回调

typedef void (CALL_METHOD *fTransComCallBack) (long lLoginID, long lTransComType, char *pBuffer, unsigned long dwBufSize, unsigned long dwUser);

58. H264_DVR_API bool CALL_METHOD H264_DVR_OpenTransComChannel(long lLoginID, TransComChannel *TransInfo, fTransComCallBack cbTransCom, unsigned long lUser);

- 函数说明：打开透明串口
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值，

[in] TransInfo

串口参数，具体参考 TransComChannel

[in] cbTransCom

回调

lLoginID：设备登陆句柄

lTransComType：串口类型，见 SERIAL_TYPE

pBuffer:回调的数据缓冲

dwBufSize：回调的数据长度

dwUser：用户数据

[in] lUser：用户数据

- 返回值：1 成功 0 失败

59. H264_DVR_API bool CALL_METHOD H264_DVR_CloseTransComChannel(long lLoginID, SERIAL_TYPE nType);

- 函数说明：关闭透明串口
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值，

[in] nType

串口类型，具体参考 SERIAL_TYPE

- 返回值：1 成功 0 失败

60. H264_DVR_API bool CALL_METHOD H264_DVR_GetDeviceState(long lLoginID, SDK_State_Type type, char *pState);

- 函数说明：获取设备状态
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值，

[in] nType

具体参考 SDK_State_Type

[in] pState

数据返回

- 返回值：1 成功 0 失败

3.25 DVR 本地用户操作界面截图

61. H264_DVR_API bool CALL_METHOD H264_DVR_CatchPicUI(long lLoginID,char *saveFileName,int type=1)

- 函数说明：获取 DVR 本地端用户操作界面截图
- 参数说明：

[in] lLoginID

设备登陆句柄：H264_DVR_Login 返回值，

[in] saveFileName

保存路径

[in] type

截图类型：（1：截取用户整体界面；2：截取活动操作界面），默认为 1；

- 返回值：1 成功 0 失败

3.26 客户端录像

62. H264_DVR_API bool H264_DVR_StartLocalRecord (long lRealHandle,char*szSaveFileName,long type)

- 函数说明：对预览进行 pc 端录像
- 参数说明：

[in] lRealHandle

H264_DVR_RealPlay 的返回值

[in] szSaveFileName

保存路径

[in] type

录像类型：（0：文件名后缀为.h264；2：文件名后缀.avi），默认为 0；

- 返回值：1 成功 0 失败

63. H264_DVR_API bool H264_DVR_StopLocalPlay(long lPlayHandle)

- 函数说明：停止 pc 端录像
- 参数说明：

  [in] lRealHandle

  H264_DVR_RealPlay 的返回值
- 返回值：1 成功 0 失败

3.27 打开语言对讲（2）

64. H264_DVR_API long H264_DVR_StartLocalVoiceCom(long lLoginID)

- 函数说明：打开与设备的对讲，H264_DVR_StopVoiceCom 关闭对讲
- 参数说明：

  [in] lLoginID

  设备登陆句柄：H264_DVR_Login 返回值，
- 返回值：失败返回 0，成功返回对讲 ID，将作为 H264_DVR_StopVoiceCom 的参数。
- 典型应用：H264_DVR_StartLocalVoiceCom 开对讲 H264_DVR_StopVoiceCom 关对讲

3.28 客户端音频

65. H264_DVR_API bool H264_DVR_OpenSound(long lHandle)

- 函数说明：打开视频通道的的音频
- 参数说明：

  [in] lHandle

  H264_DVR_RealPlay 或 H264_DVR_StartLocalPlay 或 H264_DVR_PlayBackByName 或 H264_DVR_PlayBackByTimeEx 的返回值
- 返回值：1 成功 0 失败

66. H264_DVR_API bool H264_DVR_CloseSound(long lHandle)

- 函数说明：关闭视频通道音频
- 参数说明：

  [in] lHandle

  H264_DVR_RealPlay 或 H264_DVR_StartLocalPlay 或 H264_DVR_PlayBackByName 或 H264_DVR_PlayBackByTimeEx 的返回值

- 返回值：1 成功 0 失败

3.29 客户端抓图

67. H264_DVR_API bool H264_DVR_LocalCatchPic(long lHandle)

- 函数说明：视频通道 pc 端抓图
- 参数说明：

[in] lHandle

H264_DVR_RealPlay 或 H264_DVR_StartLocalPlay 或 H264_DVR_PlayBackByName 或 H264_DVR_PlayBackByTimeEx 的返回值

[in] szSaveFileName

保存路径

- 返回值：1 成功 0 失败

3.30 播放定位

68. H264_DVR_API float H264_DVR_GetPlayPos(long lPlayHandle)

- 函数说明：获取回放或本地播放的播放进度
- 参数说明：

[in] lPlayHandle

H264_DVR_StartLocalPlay 或 H264_DVR_PlayBackByName 或 H264_DVR_PlayBackByTimeEx 的返回值

- 返回值：播放百分比

69. H264_DVR_API bool H264_DVR_SetPlayPos(long lPlayHandle,float fRelativPos)

- 函数说明：设置回放或本地播放的播放进度
- 参数说明：

[in] lPlayHandle

H264_DVR_StartLocalPlay 或 H264_DVR_PlayBackByName 或 H264_DVR_PlayBackByTimeEx 的返回值

[in] fRelativPos

播放百分比

- 返回值：1 成功 0 失败

3.31 设置信息帧回调

70. H264_DVR_API bool CALL_METHOD H264_DVR_SetInfoFrameCallBack(long lPlayHandle,InfoFramCallBack callback,long user)

回调函数原形：

typedef void (*InfoFramCallBack)(long lPlayHand, long nType, LPCSTR pBuf,long nSize, long nUser);(nType: （0x03 代表 gprs 信息帧））

- 函数说明：设置信息帧回调
- 参数说明：

[in] lPlayHandle

H264_DVR_RealPlay 或 H264_DVR_StartLocalPlay 或 H264_DVR_PlayBackByName 或 H264_DVR_PlayBackByTimeEx 的返回值

[in] callback

回调函数

[in] user

用户自定义数据

- 返回值：1 成功 0 失败

3.32 客户端视频颜色

71. H264_DVR_API bool CALL_METHOD H264_DVR_LocalGetColor(long lHandle, DWORD nRegionNum, LONG *pBrightness, LONG *pContrast, LONG *pSaturation, LONG *pHue)

- 函数说明：获取播放视频颜色信息
- 参数说明：

[in] lHandle

H264_DVR_RealPlay 或 H264_DVR_StartLocalPlay 或 H264_DVR_PlayBackByName 或 H264_DVR_PlayBackByTimeEx 的返回值

[in] nRegionNum

区域（暂时没有：可设为 0）

[in] pBrightness

亮度

[in] pContrast

对比度

[in] pSaturation

饱和度

[in] pHue

色度

- 返回值：1 成功 0 失败

72. H264_DVR_API bool CALL_METHOD H264_DVR_LocalSetColor(long lHandle, DWORD nRegionNum, LONG nBrightness, LONG nContrast, LONG nSaturation, LONG nHue)

- 函数说明：设置播放视频颜色信息
- 参数说明：

[in] lHandle

H264_DVR_RealPlay 或 H264_DVR_StartLocalPlay 或 H264_DVR_PlayBackByName 或 H264_DVR_PlayBackByTimeEx 的返回值

[in] nRegionNum

区域（暂时没有：可设为 0）

[in] pBrightness

亮度

[in] pContrast

对比度

[in] pSaturation

饱和度

[in] pHue

色度

- 返回值：1 成功 0 失败

3.33 播放客户端本地文件

73. H264_DVR_API long H264_DVR_StartLocalPlay(char*pFileName,void* hWnd,fPlayDrawCallBack drawCallBack=NULL,long user=NULL)

回调函数原形：

typedef void (CALL_METHOD * fPlayDrawCallBack)(long lPlayHand,HDC hDc,long nUser);

- 函数说明：播放本地.h264 视频文件
- 参数说明：

  [in] pFileName

  播放文件名

  [in] hWnd

  播放窗口句柄

  [in] drawCallBack

  叠加绘制回调函数（不用可以设为 NULL）

  [in] user

  用户自定义数据

- 返回值：失败返回 0，成功返回播放 ID(本地播放句柄)，将作为相关函数的参数。

74. H264_DVR_API bool H264_DVR_StopLocalPlay(long lPlayHandle)

- 函数说明：停止 PC 端播放
- 参数说明：

  [in] lPlayHandle

  H264_DVR_StartLocalPlay 返回值

- 返回值：1 成功 0 失败

75. H264_DVR_API bool H264_DVR_SetFileEndCallBack(long lPlayHandle,fLocalPlayFileCallBack callBack,long user)

回调函数原形：

typedef void (CALL_METHOD * fLocalPlayFileCallBack)(long lPlayHand, long nUser);

- 函数说明：本地文件播放结束回调
- 参数说明：

[in] lPlayHandle

H264_DVR_StartLocalPlay 返回值

[in] callBack

结束回调

[in] user

用户自定义数据

- 返回值：1 成功 0 失败

76. H264_DVR_API bool CALL_METHOD H264_DVR_LocalPlayCtrl(long lPlayHandle,SDK_LoalPlayAction action,long lCtrlValue)

- 函数说明：pc 端文件播放控制
- 参数说明：

[in] lPlayHandle

H264_DVR_StartLocalPlay 返回值

[in] action

参见：SDK_LoalPlayAction

[in] lCtrlValue

快放（1,2,3,4 级别），和慢放（1,2,3,4 级别）

- 返回值：1 成功 0 失败

3.34 绑定本地 ip

77. H264_DVR_API bool CALL_METHOD H264_DVR_SetLocalBindAddress(char*szIP)

- 函数说明：设置绑定的 ip 地址（在多网卡的时候，可以指定绑定的 ip 地址）
- 参数说明：

[in] szIP

绑定的 ip 地址

- 返回值：1 成功 0 失败

3.35 设置上报数据回调

78. H264_DVR_API bool CALL_METHOD H264_DVR_StartUploadData(long lLoginID,UploadDataType upLoadType,fUploadDataCallBack callBack,long lUser);

- 函数说明：打开数据上报，目前只有车载数据上传
- 参数说明：

  [in] lLoginID

  登录句柄

  [in] upLoadType

  上传数据类型

  [in] callBack

  上传数据回调

  [in] lUser

  用户数据

- 返回值：1 成功 0 失败

79. H264_DVR_API bool CALL_METHOD H264_DVR_StopUploadData(long lLoginID,UploadDataType upLoadType);

- 函数说明：停止上传数据
- 参数说明：

  [in] lLoginID

  登录句柄

  [in] upLoadType

  登入类型

  返回值：1 成功 0 失败

3.36 支持设备主动注册

80. H264_DVR_API bool CALL_METHOD H264_DVR_StartActiveRigister(int nPort, fMessCallBack cbFunc, unsigned long dwDataUser);

- 函数说明：开始主动注册的监听

- 参数说明：

  [in] nPort

  监听端口号

  [in] cbFunc

  设备注册回调（回调参数中有：设备登录句柄 相当于 H264_DVR_Login 的返回值，设备信息 H264_DVR_ACTIVEREG_INFO buf）

  [in] dwDataUser

  用户数据

- 返回值：1 成功 0 失败

81. H264_DVR_API bool CALL_METHOD H264_DVR_StopActiveRigister()

- 函数说明：停止主动注册的监听
- 返回值：1 成功 0 失败

3.37 设置子连接断开回调

82. H264_DVR_API long CALL_METHOD H264_DVR_SetSubDisconnectCallBack(fSubDisConnect callBack,DWORD userData);

- 函数说明：设置子连接断线回调
- 参数说明：

  [in] callBack

  子连接断线回调 （参数中的类型参见 SubConnType）

  [in] userData

  用户数据

- 返回值：1 成功 0 失败

3.38 设置心跳包时间以及断线时间

83. H264_DVR_API long CALL_METHOD H264_DVR_SetKeepLifeTime(long lLoginID,unsigned int perKeeplifeTime,unsigned int detectDisconTime);

- 函数说明：设置发送心跳包的间隔时间和设备断线时间（即多久没有收到心跳包就表示设备断开）

- 参数说明：

  [in] lLoginID

  登录句柄

  [in] perKeeplifeTime

  发送心跳包的间隔时间

  [in] detectDisconTime

  设备断线时间

- 返回值：1 成功 0 失败

3.39 搜索设备局域网设备

84. H264_DVR_API bool CALL_METHOD H264_DVR_SearchDeviceEX(long lLoginID,SDK_NetDevList *pDevlist,SDK_TransferProtocol_V2 transferProtocol = SDK_TRANSFER_PROTOCOL_NETIP,int waittime = 15000)

- 函数说明： 搜索设备局域网设备（设备设备，返回搜索的结果）
- 参数说明：

  [in] lLoginID

  登录句柄

  [out] pDevlist

  搜索到的设备，

  [int] transferProtocol

  搜索一下，参考类型 [SDK_TransferProtocol_V2]，默认 SDK_TRANSFER_PROTOCOL_NETIP

  [in] waittime

  指定设备搜索的总时间，超过时间则认为搜索失败,默认 15s(一个协议)

- 返回值：1 成功 0 失败

4 示例功能实现

请参看 ClientDemo 程序和” DEMO 说明.doc。