SDK 开发手册

版权所有　侵权必究

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	作者
2018-4-17	增加人脸抓拍、对比、黑白名单接口。	
2018-10-23	重新审核整理。	
2019-03-13	增加对 NVR 的以图搜图等功能接口，并更新修正部分不完整内容。	
2019-09-20	增加了抓拍，比对通用接口，优化了逻辑。	
2019-11-01	删除了部分无用接口，更新了接口以及数据结构，修正了部分书写错误。	

目　　录

1. 简 介 6
1.1 概述 6
1.2 适用性 6
设计原则 6
1.3 编程说明 6
1.4 典型调用顺序 7
2 数据结构定义 10
2.1 客户端数据结构 10
2.1.1 常量定义 16
2.1.2 设备信息结构 23
2.1.3 时间信息 24
2.1.4 录像文件信息 26
2.1.5 配置信息结构 32
2.1.6 网络报警信息 53
2.1.7 存储设备控制信息 54
2.1.8 RTSP 信息 54
2.1.9 媒体包以及包信息 54
2.1.10 本地播放控制 57
2.1.11 主动服务 57
2.1.12 子连接类型 57
2.1.13 连接类型 58
2.1.14 搜索协议类型 58
2.1.15 人脸抓拍，比对，黑白名单 58

3 接口定义 76
3.1 SDK 初始化 76
3.2 报警状态获取 77
3.3 设备注册 78
3.4 实时监视 80
3.5 回放和下载 81
3.6 云台控制 84
3.7 系统配置 86
3.8 日志管理 88
3.9 远程控制 89
3.10 语音对讲 90
3.11 录像模式设置 92
3.12 设置系统时间 93
3.13 获取设置运行状态信息 93
3.14 网络报警 94
3.15 磁盘管理 94
3.16 抓图 94
3.17 支持强迫 I 帧 96
3.18 设置连接设备超时时间和尝试次数 96
3.19 本地用户操作界面截图 96
3.20 绑定本地 IP 97
3.21 支持设备主动注册 97
3.22 设置子连接断开回调 98
3.23 设置心跳包时间以及断线时间 98
3.24 搜索设备局域网设备 99
3.25 人脸图片抓拍、对比 104
3.26 黑白名单 106
3.27 数据回调函数 114
4 示例功能实现 120

1.简 介

1.1 概述

欢迎使用我公司 SDK 编程手册，SDK 是软件开发商在开发我司摄像机应用时的开发套件。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	**VideoNetAPI.h**	头文件
	libjvtsdk.so	动态库
	libjvtsdk.a	静态库
	libnpcmpimontsp.so	32 位 P2P 服务动态库
	libnpcmpimontsp_x64.so	64 位 P2P 服务动态库
辅助 demo	**test.cpp**	demo
	read.me	demo 编译方法

1.2 适用性

- 支持摄像机的监视、回放、报警、远程配置等功能。
- 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

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 VideoNet_Init()对系统进行初始化，应用程序退出时调用 VideoNet_Cleanup()释放所有占用的资源。
2. 大多数函数调用均应该在 VideoNet_Init()之后，VideoNet_Cleanup()之前，而 VideoNet_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 VideoNet_Login()登录到前端设备上，如果是广域网环境,可以调用VideoNet_Login_P2P(),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 VideoNet_Loginout()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

1） 初始化

SDK 初始化　　VideoNet_Init()

2） SDK 功能信息获取

设置报警消息回调　　VideoNet_SetDVRMessCallBack()

3） 登录连接设备

登入设备　　VideoNet_Login()

VideoNet_Login_P2P()

报警消息订阅　　VideoNet_SetupAlarmChan()

4） 设备功能操作与信息获取

系统参数配置　　VideoNet_GetDevConfig()

VideoNet_SetDevConfig()

查询日志　　VideoNet_FindDVRLog()

云台控制　　VideoNet_PTZControl()

VideoNet_PTZControlEx()

5） 实时监视通道

打开监视通道　　VideoNet_RealPlay()

VideoNet_StopRealPlay()

监视数据回调保存　VideoNet_SetRealDataCallBack()

6) 人脸抓拍比对通道

打开比对通道　　VideoNet_StartReceiveFaceMonitor()

设置回调　　VideoNet_SetFaceMonitorCallBack()

VideoNet_StopReceiveFaceMonitor()

7) 回放/下载通道

查询录像　　VideoNet_FindFile()

VideoNet_FindFileByTime()

回放及控制，下载

VideoNet_GetFileByName()

VideoNet_GetFileByTime()

VideoNet_GetDownloadPos()

VideoNet_StopGetFile()

8) 远程控制

远程升级　　VideoNet_Upgrade()

VideoNet_GetUpgradeState()

重启/清除日志　　VideoNet_ControlDVR（）

9) 注销断开设备

停止报警消息订阅　VideoNet_CloseAlarmChan()

断开连接　　VideoNet_Logout()

10) 释放 SDK 资源

SDK 退出　　VideoNet_Cleanup()

2 数据结构定义

2.1 客户端数据结构

```
//错误类型代号，用于GetLastError函数的返回
enum SDK_RET_CODE
{
    VIDEONET_NOERROR                    = 0,            //没有错误
    VIDEONET_SUCCESS                    = 1,            //返回成功
    VIDEONET_SDK_NOTVALID               = -10000,       //非法请求
    VIDEONET_NO_INIT                    = -10001,       //SDK未经初始化
    VIDEONET_ILLEGAL_PARAM              = -10002,       //用户参数不合法
    VIDEONET_INVALID_HANDLE             = -10003,       //句柄无效
    VIDEONET_SDK_UNINIT_ERROR           = -10004,       //SDK清理出错
    VIDEONET_SDK_TIMEOUT                = -10005,       //等待超时
    VIDEONET_SDK_MEMORY_ERROR           = -10006,       //内存错误,创建内存失败
    VIDEONET_SDK_NET_ERROR              = -10007,       //网络错误
    VIDEONET_SDK_OPEN_FILE_ERROR        = -10008,       //打开文件失败
    VIDEONET_SDK_UNKNOWNERROR           = -10009,       //未知错误
    VIDEONET_DEV_VER_NOMATCH            = -11000,       //收到数据不正确,可能版
本不匹配
    VIDEONET_SDK_NOTSUPPORT             = -11001,       //版本不支持
    VIDEONET_SDK_FILE_FORAMAT_ERROR     = -11002,       //文件格式 错误
    VIDEONET_SDK_NETWORK_LOST_PACKET    = -11003,       //前端网络丢包
    VIDEONET_SDK_FILE_DATA_LARGE_ERROR   = -11004,        //文件数据过大
    VIDEONET_ANAS_EXIST                 = -11130,       //NAS地址已存在
    VIDEONET_ANAS_ALIVE                 = -11131,       //路径被使用，无法操作
    VIDEONET_ANAS_FULL                  = -11132,       //NAS已达到支持的最大
值
    VIDEONET_OPEN_CHANNEL_ERROR         = -11200,       //打开通道失败,可能检
测到设备已经不在线
    VIDEONET_CLOSE_CHANNEL_ERROR        = -11201,       //关闭通道失败
    VIDEONET_SUB_CONNECT_ERROR          = -11202,       //建立媒体子连接失败,
网络出错或者设备可能不在线
    VIDEONET_SUB_CONNECT_SEND_ERROR     = -11203,       //媒体子连接通讯失败,
可能检测到设备已经不在线
    VIDEONET_NOTSUPPORT                 = -11205,       //版本不支持
    VIDEONET_NOTVALID                   = -11206,       //请求非法,主连接可能
已断开
    VIDEONET_TCPCONNET_REACHED_MAX      = -11207,       //Tcp视频链接达到最大,
不允许新的Tcp视频链接
    VIDEONET_OPENEDPREVIEW              = -11208,       //该通道已经打开预览(通
```

道的打开关闭需要一一对应,打开几次需要关闭几次;不一致会打开提示该错误;预防客户端开发逻辑上的不合理设计增加该错误值)

```
    // 用户管理部分错误码
    VIDEONET_NOPOWER                    = -11300,        //无权限
    VIDEONET_PASSWORD_NOT_VALID         = -11301,        //账号密码不对
    VIDEONET_LOGIN_USER_NOEXIST         = -11302,        //用户不存在
    VIDEONET_USER_LOCKED                = -11303,        //该用户被锁定
    VIDEONET_USER_IN_BLACKLIST          = -11304,            //该用户不允许访问(
在黑名单中)
    VIDEONET_USER_HAS_USED              = -11305,        //该用户已登陆
    VIDEONET_USER_NOT_LOGIN             = -11306,        //该用户没有登陆
    VIDEONET_CONNECT_DEVICE_ERROR       = -11307,        //可能设备不存在
    VIDEONET_ACCOUNT_INPUT_NOT_VALID     = -11308,     //用户管理输入不合法
    VIDEONET_ACCOUNT_OVERLAP             = -11309,     //索引重复
    VIDEONET_ACCOUNT_OBJECT_NONE         = -11310,     //不存在对象，用于查询
时
    VIDEONET_ACCOUNT_OBJECT_NOT_VALID    = -11311,     //不存在对象
    VIDEONET_ACCOUNT_OBJECT_IN_USE       = -11312,     //对象正在使用
    VIDEONET_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)
    VIDEONET_ACCOUNT_PWD_NOT_VALID       = -11314,     //密码不正确
    VIDEONET_ACCOUNT_PWD_NOT_MATCH       = -11315,     //密码不匹配
    VIDEONET_ACCOUNT_RESERVED            = -11316,     //保留帐号
    VIDEONET_ACCOUNT_SYS_MAINTAIN        = -11317,     //系统维护中，不可登陆
    VIDEONET_EE_DVR_PASSWORD_NOT_VALID2  = -11318,     //账号密码不对

    // 配置管理相关错误码
    VIDEONET_OPT_RESTART                 = -11400,     //保存配置后需要重启应
用程序
    VIDEONET_OPT_REBOOT                  = -11401,     //需要重启系统
    VIDEONET_OPT_FILE_ERROR              = -11402,     //写文件出错
    VIDEONET_OPT_CAPS_ERROR              = -11403,     //配置特性不支持
    VIDEONET_OPT_VALIDATE_ERROR          = -11404,     //配置校验失败
    VIDEONET_OPT_CONFIG_NOT_EXIST        = -11405,     //请求或者设置的配置不
存在

    VIDEONET_CTRL_PAUSE_ERROR            = -11500,     //暂停失败
    VIDEONET_SDK_NOTFOUND                = -11501,     //查找失败,没有找到对应
文件
    VIDEONET_CFG_NOT_ENABLE              = -11502,     //配置未启用
    VIDEONET_DECORD_FAIL                 = -11503,     //解码失败
```

```
    //DNS协议解析返回错误码
    VIDEONET_SOCKET_ERROR                = -11600,     //创建套节字失败
    VIDEONET_SOCKET_CONNECT              = -11601,     //连接套节字失败
    VIDEONET_SOCKET_DOMAIN               = -11602,     //域名解析失败
    VIDEONET_SOCKET_SEND                 = -11603,     //发送数据失败
    VIDEONET_NO_CONNECT_FRONT        = -11614,         //前端设备未连接或者连
接的前端设备分辨率未知
    VIDEONET_LOGIN_FULL              = -11615,         //登录句柄已达到最大值，
无法再登录
    VIDEONET_ARSP_USER_NOEXIST       = -11619,         //用户不存在
    VIDEONET_ARSP_PASSWORD_ERROR     = -11620,         //账号密码不对
    VIDEONET_ARSP_QUERY_ERROR        = -11621,         //查询失败

    //升级错误码
    VIDEONET_NOENOUGH_MEMORY         = -11804,         //内存不足
    VIDEONET_INVALID_FORMAT          = -11805,         //升级文件格式不对
    VIDEONET_UPDATE_PART_FAIL        = -11806,         //某个分区升级失败
    VIDEONET_INVALID_HARDWARE        = -11807,         //硬件型号不匹配
    VIDEONET_INVALID_VENDOR          = -11808,         //客户信息不匹配
    VIDEONET_INVALID_COMPALIBLE      = -11809,         //升级程序的兼容版本号
比设备现有的小，不允许设备升级回老程序
    VIDEONET_INVALID_VERSION         = -11810,         //非法的版本
    VIDEONET_INVALID_WIFI_DRIVE      = -11811,         //升级程序里wifi驱动和
设备当前在使用的wifi网卡不匹配
    VIDEONET_INVALID_CUR_FLASH       = -11812,         //升级程序不支持设备使
用的Flash
    VIDEONET_WEBLIST_CONNECT_MAX     = -12000,         //该设备下黑白名单连接
数已达到最大数
};

// 云台操作类型
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_OUT,                        // 变倍小
    ZOOM_IN,                         // 变倍大
```

```
    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  20
    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,
};
```

// 报警事件类型

```
enum SDK_EventCodeTypes
{
    SDK_EVENT_CODE_INIT = 0,
    SDK_EVENT_CODE_LOCAL_ALARM = 1,     // 本地报警(外部报警)
    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,     // 存储设备不存在
    SDK_EVENT_CODE_STORAGE_FAILURE,       // 存储设备访问失败
    SDK_EVENT_CODE_LOW_SPACE,             // 存储设备容量过低
    SDK_EVENT_CODE_NET_ABORT,
    SDK_EVENT_CODE_COMM,
    SDK_EVENT_CODE_STORAGE_READ_ERROR,    // 存储设备读错误
    SDK_EVENT_CODE_STORAGE_WRITE_ERROR,   // 存储设备写错误
    SDK_EVENT_CODE_NET_IPCONFLICT,        // ip冲突
    SDK_EVENT_CODE_ALARM_EMERGENCY,
    SDK_EVENT_CODE_DEC_CONNECT,
    SDK_EVENT_CODE_UPGRADE,
    SDK_EVENT_CODE_BACK_UP,
    SDK_EVENT_CODE_SHUT_DOWN,
    SDK_EVENT_CODE_REBOOT,
    SDK_EVENT_CODE_NEWFILE,
    SDK_EVENT_CODE_VideoAnalyze,
    SDK_EVENT_CODE_IPC_ALARM,
    SDK_EVENT_CODE_LOGIN_FAILED,          // 登录失败
    SDK_EVENT_CODE_LumaDetect,            //亮度侦测  28
    SDK_EVENT_CODE_MotionDetect,          //智能移动侦测29
    SDK_EVENT_CODE_OdDetect,              //遮挡侦测30
    SDK_EVENT_CODE_CrossLine,             //虚拟警戒线31
    SDK_EVENT_CODE_CrossArea,             //区域入侵32
    SDK_EVENT_CODE_DetectColor,           //颜色33
    SDK_EVENT_CODE_DetectSound,           //声音  34
    SDK_EVENT_CODE_LoseObj, };
```

// 报警信息

```
typedef struct SDK_ALARM_INFO
{
    int nChannel;
    int iEvent;          // 报警事件码:见枚举 SDK_EventCodeTypes
    int iStatus;         // 0:报警开始，1:报警结束
    SDK_SYSTEM_TIME SysTime;
    char pExtInfo[128]; // 附加信息，发送者和接收者对各种报警类型进行格式约定
}SDK_AlarmInfo;
```

```
    //错误码
    typedef enum _XM_WB_ERROR_CODE
    {
        WB_OK                    = 1,    // 成功
```

```
        WB_FAILE                = -1,  // 失败
        WB_COLLECT_ERROR        = -2,  // 提取特征值失败
        WB_FILEINDEX_ERROR      = -3,  // 文件名字编号重复
        WB_LIB_FULL             = -4,  // 库满
        WB_ADD_TIME_OUT         = -5,  // 添加超时
        WB_PARA_ERROR           = -6,  // 参数错误
        WB_FILE_BIG             = -7,  // 文件太大
        WB_SPACE_ERROR          = -8,  // 存储空间不足
        WB_FILE_OPEN_ERROR      = -9,  // 文件打开失败
        WB_NO_DBFILE            = -10, // 没有数据库
        WB_FILE_ERROR           = -11, // 图片读取失败
        WB_DBFILE_BAD           = -12, // 数据库文件损坏
        WB_PIC_QUALITY_ERROR    = -13, // 图片质量差
        WB_FILE_WHSIZE_ERROR    = -14, // 图片尺寸错误，宽高不能为奇数
        WB_FILE_FACE_ERROR      = -15, // 检测人脸失败(没检测到人脸或检测到多人脸)
        WB_PIC_FORMAT_ERROR       = -16, // 图片格式错误
        WB_FACE_AREA_FAIL       = -17, // 人脸区域错误
        WB_FACE_HAND_ERROR      = -18, // 算法创建句柄错误
        WB_FACE_BUSY            = -19, // 设备忙
        WB_FILE_WRITE_ERROR           = -20,  // 文件写入失败
        WB_GET_DATABASE_ERROR         = -21,  // 获取当前库的人数失败
        WB_GET_DATABASE_POINTER_ERROR = -22,  // 获取当前库的指针失败
        WB_FACE_COMPARE_FAILED        = -23,  // 人脸比对失败
}XM_WB_ERROR_CODE;

// 黑白名单库操作类型
typedef enum _WBLIST_ACTION_TYPE
{
    OUT_INDEX = 1,                          // 导出文件索引
    OUT_PICTURE,                            // 导出黑白名单库中单张图片
    OUT_PICTURE_COMP,                       // 比对后导出响应库图片
    OUT_PICTURE_DEL,                        // 删除库中单张图片
    OUT_PICTURE_DELALL,                     // 删除黑白名单所有库
    IN_PICTURE_ADD,                         // 通过导入图片添加一张黑白名单
    IN_PICTURE_ADD_SNAP,                    // 通过抓拍添加黑白名单
    IN_PICTURE_GET_FEATURE,                 // 通过导入一张图片获取特征值
    IN_INDEX_GET_FEATURE,                   // 通过库索引获取特征值
    OUT_PICTURE_USER_INFO,                  // 导出用户详细信息
    IN_PICTURE_USER_INFO_UPDATA,            // 用户信息更新
    IN_PICTURE_GET_SIMILARITY_PIC_MSG = 12,     // 通过导入一张图片获取特征值最
相似的底库图片信息
    IN_PICTURE_GET_FEA_AND_ALARM = 13,      // 通过导入一张图片获取特征值并接受
可能的报警
```

```
    IN_COPY_DATEBASE = 14,                    // 拷贝一个数据库到另一个数据库
    IN_MODIFY_PICTURE_MESSAGE = 15,           // 修改图片信息
    IN_FEATURE_ADD,                           // 通过特征值添加一张黑白名单
    IN_FEATURE_UPDATE,                        // 通过特征值添加一张黑白名单
    OUT_CHECKON_INFO ,                        // 考勤导出
    OUT_CMP_INFO ,                            // 比对信息导出

    NAS_GET_FEATURE_QUALITY = 50,             // 客户定制获取特征值的同时返回图片
质量以及正脸率
    OUT_WBLIST_EN
}WBLIST_ACTION_TYPE;
```

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 NET_MAX_CHANNUM              64         // 最大通道个数
#define NET_DECORDR_CH               64         // 最大解码通道个数
#define NET_MAX_USER_NUM                60         // 最多用户数
#define NET_MAX_RIGTH_NUM               200        // 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              64         // 最大解码通道个数

#define NET_MAX_DDNS_TYPE               5          // 支持的DDNS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               32          // SID最大数量，暂定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_CHANNEL_NAME_MAX_LEN        64          // 通道名称最大长度
#define NET_MAX_LINE_PER_OSD_AREA       12          // 编码中一个OSD区域最多可以显示的字符行数

//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
#define DDNS_MAX_DDNS_IDSIZE            32          // 设备标识长度
#define DDNS_MAX_SERIALINFO_SIZE        16          // 序列号以及用户名长度
#define NET_MAX_MSK_SIZE                128         // 掩码数组的大小

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

//VGA分辨率
#define VGA_MAXNUM                      32          // 分辨率 种类
#define VGA_NAME_LENGTH                 10          // 分辨率 长度

//显示的设备列表(设备端搜索)
#define DEV_LIST_SHOW                   256

//IP SIZE
#define IP_SIZE                         16

//通道模式
#define MAX_HVR_CHNCAP_CHN              32

#define  NET_NAME_TOKEN_LEN             128

#define  NET_NAME_URL_LEN               128
#define  NET_NAME_ID_LEN                64
#define  NET_MAX_POS_FUNC_KEYWORDS      4           // POS功能中最多支持的关键字个数
#define  NET_MAX_TITLE_DOT_BUF_LEN      64*24*24

//硬盘信息
#define NET_FSLEN               8

#define MAX_URL_IP_LEN          64          // IP地址域名最大长度
#define MAX_USER_PWD_LEN        32          // 用户名密码最大长度
#define MAX_DEV_NAME_LEN        64
#define BUFFER_SIZE             300*1024    // 参数备份大小
```

```
// 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_H265,        // H.265
    SDK_CAPTURE_COMPRESS_H265_S,     // H.265+
    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,
};

// 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_TRANSMISSION,    // 网传优先
    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_IFRAME_RANGE,         // I帧间隔
    SDK_ENCODE_FUNCTION_TYPE_SmartH264,            // H264+
    SDK_ENCODE_FUNCTION_TYPE_INTELLIGENT_ENCODE,   // 智能编码
    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_VIDEOANALYSE,       // 视频分析
    SDK_ALARM_FUNCTION_TYPE_NET_ABORT_EXTEND,   // 网络异常扩展
    SDK_ALARM_FUNCTION_TYPE_IPC_ALARM,          // IPC报警
    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_UPNP,                  // UPNP
    SDK_NET_SERVER_TYPES_FTP,                   // FTP
    SDK_NET_SERVER_TYPES_WIFI,                  // WIFI
    SDK_NET_SERVER_TYPES_RTSP,                  // RTSP
    SDK_NET_SERVER_TYPES_DAS,                   // 主动注册
    SDK_NET_SERVER_TYPES_SPVMN,                 // 28181协议配置
    SDK_NET_SERVER_TYPE_PROTOCOL_MAC,           // 支持MAC协议
    SDK_NET_SERVER_TYPE_XMHEARTBEAT,            // 心跳
    SDK_NET_SERVER_TYPES_IPV6,                  // 支持ipv6
    SDK_NET_SERVER_TYPES_SPVMN_SIP,             // 28181配置显示SIP 板卡地址
    SDK_NET_SERVER_TYPES_RTMP,                  // RTMP协议
    SDK_NET_SERVER_TYPES_IPADAPTIVE,            // IP自适应网络能力
    SDK_NET_SERVER_TYPES_ONVIF_PWD_CHECKOUT,    // ONVIF密码校验
};
```

```
// 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_CAMERA_PARAM,     // 网络摄像头配置
    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_KEY,         // 关键录像
    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_PIC_IDXIMG,         // 索引图片
    SDK_TYPE_NUM,
    SDK_PIC_FACE = 20,      // 人脸搜索图片
    SDK_PIC_CARNO,          // 车牌
    SDK_PIC_WIFI,
    SDK_FACE_QUERY_RECORD,  // 人脸搜索录像
};
```

2.1.2 设备信息结构

```
typedef enum _SMART_DEVICE_TYPE
{
    SMART_DEV_TYPE_IPC = 0,     // 普通摄像机
    SMART_DEV_TYPE_DVR,         // 兼容 NVR 中协议有关设备定义
    SMART_DEV_TYPE_HVR,
    SMART_DEV_TYPE_POEIPC,
    SMART_DEV_TYPE_NVR,
```

```
    SMART_DEV_TYPE_RTSPIPC,
    SMART_DEV_TYPE_BSIPC,       // 比对机
    SMART_DEV_TYPE_FPIPC,       // 抓拍机
    SMART_DEV_TYPE_CPIPC,       // 车牌抓拍机
    SMART_DEV_TYPE_END
}SMART_DEVICE_TYPE;
```

//设备结构定义如下

```
typedef struct _VIDEONET_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
    DWORD uiDeviceRunTime;         // 系统运行时间
    SDK_DeviceType deviceTye;      // 设备类型
    char sHardWare[64];            // 设备型号
    char uUpdataTime[20];          // 更新日期 例如 2013-09-03 14:15:13
    DWORD uUpdataType;             // 更新内容
    char sDeviceModel[16];         // 设备型号(底层库从加密里获得，sHardWare 针对
多个设备用同一个程序这种情况区分不了)
    int nLanguage;                 // 国家的语言 ID,0 英语 1 中文 2 中文繁体 3 韩语 4
德语 5 葡萄牙语 6 俄语
    char szOnvifModel[32];
    char sProtocol[16];
    SMART_DEVICE_TYPE smarttype;
    char res[112];
}VIDEONET_DEVICEINFO,*LPVIDEONET_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, 以此类推。
    int  day;         // 日。
    int  wday;        // 星期，Sunday = 0, Monday = 1, 以此类推。
    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 SDK_TIMESECTION
{
    int enable;          // 使能
    int startHour;       // 开始时间:小时
    int startMinute;     // 开始时间:分钟
    int startSecond;     // 开始时间:秒钟
    int endHour;         // 结束时间:小时
    int endMinute;       // 结束时间:分钟
    int endSecond;       // 结束时间:秒钟
}SDK_TIMESECTION;


typedef struct{
    int dwYear;      // 年
    int dwMonth;     // 月
    int dwDay;       // 日
    int dwHour;      // 时
    int dwMinute;    // 分
    int dwSecond;    // 秒
}VIDEONET_TIME,*LPVIDEONET_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:从 2  3  4 开始,  -1:最后一周    0:表示使用按日计算的方法
[-1,4]
    int iWeekDay; // 一周从 星期天=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 VIDEONET_FINDINFO
{
    int nChannelN0;             // 通道号
    int nFileType;              // 文件类型, 见 SDK_File_Type
    VIDEONET_TIME startTime;    // 开始时间
    VIDEONET_TIME endTime;      // 结束时间
    char szFileName[32];        // 文件名，为空的话，系统处理，有值，系统采用
    int  StreamType;            // 查询的码流类型,主码流:0,辅码流:1
    VIDEONET_FINDINFO()
    {
        memset(szFileName,0,sizeof(szFileName));
        StreamType = 0;
    }
}VIDEONET_FINDINFO;


// 返回录像信息结构体
typedef struct VIDEONET_FILE_DATA
{
  int ch;                           // 通道号
  int size;                             // 文件大小
```

```
  int FileType;                        // 文件类型  （0）h264 （1）mp4
  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
  int StreamType;                      // 码流类型是回放主码流（0）还是辅助码流（1
  ）
  VIDEONET_FILE_DATA()
  {
     memset(sFileName,0,sizeof(sFileName));
     StreamType = 0;
  };
}VIDEONET_FILE_DATA;
```

// 按时间段查询

```
typedef 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
  unsigned int   nHighStreamType; // 33~64 录像的码流类型,二进制位为 0 代表主码流
  , 1 代表辅码流
  unsigned int nLowStreamType;    // 1~32 录像的码流类型,二进制位为 0 代表主码流,
  1 代表辅码流
}SDK_SearchByTime;
```

// 每个通道的录像信息

```
struct SDK_SearchByTimeInfo
{
  int iChannel;                           // 录像通道号
  unsigned char cRecordBitMap[720];       // 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
};

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
typedef struct SDK_COMMATTRI
{
  int  iDataBits;    // 数据位取值为 5,6,7,8
  int  iStopBits;    // 停止位
  int  iParity;      // 校验位
  int  iBaudRate;    // 实际波特率
  int  flowControl; // 流控制  0: 无  1: 软流控  2:硬流控
}SDK_COMMATTRI;

// 串口协议
enum SDK_CommProtocol
{
  SDK_CONSOLE = 0,
  SDK_KEYBOARD,
  SDK_COM_TYPES,
};

// 串口协议
typedef struct SDK_COMMFUNC
{
  int nProNum;
  char vCommProtocol[100][32]; // 每个协议最多由 32 个字符组成
};

// 串口配置
typedef 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
}SDK_CONFIG_COMM_X;

typedef struct SDK_CommConfigAll
{
   SDK_CONFIG_COMM_X vCommConfig[SDK_COM_TYPES];
}SDK_CommConfigAll;

// 云台设置
typedef 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
}SDK_STR_CONFIG_PTZ;

// 云台协议
struct SDK_PTZPROTOCOLFUNC
{
    int nProNum;
    char vPTZProtocol[100][NET_MAX_PTZ_PROTOCOL_LENGTH]; // 每个协议最多由个
32 字符组成
};

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

typedef struct SDK_CONFIG_WORKSHEET
{
    SDK_TIMESECTION  tsSchedule[NET_N_WEEKS][NET_N_TSECT];    // 时间段
}SDK_CONFIG_WORKSHEET;

// 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，0-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];
};

#define NET_MAX_RETURNED_LOGLIST  128    //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 = 4,  // 每个磁盘最多的分区数
    SDK_MAX_DISK_PER_MACHINE = 8, // 最多支持块硬盘
};

// 硬盘管理
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 VIDEONET_CLIENTINFO
{
    int nChannel; //  通道号
    int nStream;  //  0 表示主码流，为 1 表示子码流
    int nMode;    //  0 TCP 方式,1 UDP 方式,2 多播方式,3 RTP 方式,4 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
}*LPVIDEONET_CLIENTINFO;
```

2.1.5 配置信息结构

VideoNet_GetDevConfig、VideoNet_SetDevConfig 的命令定义(参考 VideoNetAPI.h 中枚举_SDK_CONFIG_TYPE)

```
typedef enum _SDK_CONFIG_TYPE
{
    E_SDK_CONFIG_NOTHING = 0,
    E_SDK_CONFIG_USER,                          // 用户信息，包含了权限列表，用户列
    表和组列表    对应结构体USER_MANAGE_INFO
    ……
    …….
    ……..
    E_SDK_CFG_SMARTDETECTION_CODEC,          // 智能编码
    E_SDK_CFG_SMARTDETECTION_DETECT,         // 智能检测
    E_SDK_CFG_SMARTDETECTION_CONTROL,        // 智能控制
    E_SDK_CFG_SMARTDETECTION_LUMINANCE,      // 亮度侦测
    E_SDK_CFG_SMARTDETECTION_MOVE,           // 移动侦测
    E_SDK_CFG_SMARTDETECTION_OCCLUSION,      // 遮挡侦测
    E_SDK_CFG_SMARTDETECTION_CROSSLINE,      // 虚拟警戒线
    E_SDK_CFG_SMARTDETECTION_CROSSAREA,                 // 区域入侵
    E_SDK_CFG_SMARTDETECTION_COLOR,                     // 颜色
    E_SDK_CFG_SMARTDETECTION_SOUND,                     // 声音
    E_SDK_CFG_SMARTDETECTION_LOSEOBJ,                   // 物品预留
    E_SDK_CFG_CALIBRATION_GET_CAMERA_PARAMETER,         // 获取摄像机是否已经标
定
    E_SDK_CFG_CALIBRATION_SET_SUBCAMERA_CONNECTINFO,    // 设置从机登录信息
    E_SDK_CFG_CALIBRATION_SET_POINTS,                   // 设置标定坐标(增、删，
3D)
    E_SDK_CFG_CALIBRATION_ACTION,                       // 设置标定（开始、完成）
    E_SDK_CFG_CALIBRATION_MODE,                         // 标定模式
    E_SDK_CFG_CALIBRATION_ZOOMTIMES = 310,              // 变倍系数
    E_SDK_CFG_CALIBRATION_PTZINFO,                      // PTZ参数
```

```
    E_SDK_CFG_CALIBRATION_NAMING,                          // 点名球设置获取

    E_SDK_CFG_FACECOMPARE_RESULT_ALARM,                    // 对比结果报警
    E_SDK_CFG_SMARTFACE_COMMON_PARAMETER,                  // 人脸识别参数
    E_SDK_CFG_SMARTFACE_AREADETECTION,                     // 人脸识别侦测区域
    E_SDK_CFG_SMARTFACE_AREASHIELDED,                      // 人脸区域屏蔽
    E_SDK_CFG_SMARTFACE_VERSION,                           // 识别算法版本号
    E_SDK_CFG_SMARTFACE_ALARM,                             // 人脸报警参数
    E_SDK_CFG_SMARTFACE_COUNT,                             // 人脸统计参数
    E_SDK_CFG_EMMCINFO,                                    // EMMC参数
    E_SDK_CFG_AWBAUTO_MODE,     // 白平衡设置,对应结构体SDK_AWBAUTO_MODE
    E_SDK_CFG_SMART_CAR,        // 车牌抓拍参数配置, 对应结构体SDK_SMART_CAR_S
    E_SDK_CFG_GB28181,          // GB28181注册信息
    E_SDK_CFG_CENTERCONNECT,    // 中心连接
    E_SDK_CFG_REPORTIP = 325,   // 外网IP参数
    E_SDK_CFG_HTTPAPI,          // 服务器参数
    E_SDK_CFG_HTTPAPI2,         // 服务器参数
    E_SDK_CFG_SENIORSET,        // 视频设置.视频参数.高级设置
    E_SDK_CFG_GETPARAM,             // 系统参数.参数备份
    E_SDK_CFG_GUI,              // 输出模式(含OSD字符叠加)
};

// 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;
};

// 编码信息
typedef struct SDK_EncodeInfo
{
    bool bEnable;                   // 使能项
    bool bHaveAudio;                // 是否支持音频
    int  iStreamType;               // 码流类型，capture_channel_t
    unsigned int uiCompression;     // capture_comp_t的掩码
    unsigned int uiResolution;      // SDK_CAPTURE_SIZE_t的掩码
```

```
}SDK_EncodeInfo;

// 编码能力
typedef 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
    unsigned int CompressionMask[NET_MAX_CHANNUM];        // 编码模式 capture_comp_t掩码
    unsigned int ThridImageSize[NET_MAX_CHANNUM];         // CHL_3IRD_T码流支持的图像分辨率
    unsigned int FourthImageSize[NET_MAX_CHANNUM];        // CHL_4RTH_T码流支持的图像分辨率
        SDK_EncodeInfo vEncodeInfo[SDK_CHL_FUNCTION_NUM]; // 编码信息,暂时最大就4中码流
        SDK_EncodeInfo vCombEncInfo[SDK_CHL_FUNCTION_NUM];// 组合编码信息,暂时最大就4中码流
        int iMaxBps;                                      // 最高码流Kbps
        unsigned int
ExImageSizePerChannelEx[NET_MAX_CHANNUM][SDK_CAPTURE_SIZE_EXT_V3_NR];
// 每个通道支持的辅码流图像分辨率
}CONFIG_EncodeAbility;

//支持的系统功能
typedef struct SDK_SystemFunction
{
    bool vEncodeFunction[SDK_ENCODE_FUNCTION_TYPE_NR];    // 编码功能
    bool vAlarmFunction[SDK_ALARM_FUNCTION_TYPE_NR];      // 报警功能
    bool vNetServerFunction[SDK_NET_SERVER_TYPES_NR];     // 网络服务功能
    bool vPreviewFunction[SDK_PREVIEW_TYPES_NR];          // 预览功能
    bool vCommFunction[SDK_COMM_TYPES_NR];                // 串口类型
    bool vInputMethodFunction[SDK_NO_SUPPORT_NR];         // 输入法限制
    bool vTipShowFunction[SDK_NO_TIP_SHOW_NR];            // 报警标签显示
    bool vOtherFunction[SDK_OTHER_NR];                    // 其他功能
}SDK_SystemFunction;

// 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
typedef struct SDK_PtzLinkConfig
{
  int iType;      // 联动的类型--对应结构体 PtzLinkTypes
  int iValue;     // 联动的类型对应的值
}SDK_PtzLinkConfig;

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

// 联动操作
typedef 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
    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
    bool   bShortMsg;                   // 发送短信
    bool   bMultimediaMsg;              // 发送彩信
    int    dwSnapNum;                   // 连续抓拍张数
    int    SnapMailEnable;              // 联动抓拍发送 E-mail
    int    RecordFTPEnable;             // 联动录像 Ftp 上传
    int    aIoType;                     // IO 输出类型
} SDK_EventHandler;

// 遮挡检测配置
typedef struct SDK_BLINDDETECTCONFIG
{
    bool   bEnable;             // 遮挡检测开启
    int    iLevel;              // 灵敏度：1?6
    SDK_EventHandler hEvent;    // 遮挡检测联动
}SDK_BLINDDETECTCONFIG;

// 全通道遮挡检测配置
struct SDK_BLINDDETECTCONFIG_ALL
{
    SDK_BLINDDETECTCONFIG vBlindDetectAll[NET_MAX_CHANNUM];
} SDK_BLINDDETECTCONFIG_ALL;

// 报警输入配置
typedef 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
}SDK_ALARM_INPUTCONFIG;

// 网路报警
typedef struct SDK_NETALARMCONFIG
{
    bool bEnable;               // 使能
    SDK_EventHandler hEvent;    // 处理参数
}SDK_NETALARMCONFIG;
```

```
// 所有通道的网路报警结构
typedef struct SDK_NETALARMCONFIG_ALL
{
   SDK_NETALARMCONFIG vNetAlarmConfig[NET_MAX_CHANNUM];
}SDK_NETALARMCONFIG_ALL;

// 本地报警输出配置
typedef 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
}SDK_AlarmOutConfig;

// 所有通道的报警输出配置
typedef struct SDK_AlarmOutConfigAll
{
   SDK_AlarmOutConfig vAlarmOutConfigAll[NET_MAX_CHANNUM];
}SDK_AlarmOutConfigAll;

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

// 驱动器信息结构
typedef 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
   char      fsType[NET_FSLEN];        // 文件类型
}SDK_DriverInformation;
```

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

// 动态检测设置
typedef 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
}SDK_MOTIONCONFIG;

// 全通道动态检测配置
typedef struct SDK_MOTIONCONFIG_ALL
{
    SDK_MOTIONCONFIG vMotionDetectAll[NET_MAX_CHANNUM];
}SDK_MOTIONCONFIG_ALL;

// 视频丢失
typedef struct SDK_VIDEOLOSSCONFIG
{
    bool bEnable;               // 使能
    SDK_EventHandler hEvent;    // 处理参数
}SDK_VIDEOLOSSCONFIG;

// 所有通道的视频丢失结构
typedef struct SDK_VIDEOLOSSCONFIG_ALL
{
    SDK_VIDEOLOSSCONFIG vGenericEventConfig[NET_MAX_CHANNUM];
}SDK_VIDEOLOSSCONFIG_ALL;

// 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
typedef struct SDK_RECORDCONFIG
{
    int    iPreRecord;          // 预录时间，为零时表示关闭
    bool   bRedundancy;         // 冗余开关
    int    iPacketLength;       // 录像打包长度（分钟）[1, 255]
    int    iRecordMode;         // 录像模式，0 关闭，1 禁止 2 配置
    SDK_CONFIG_WORKSHEET wcWorkSheet;                    // 录像时间段
    unsigned int typeMask[NET_N_WEEKS][NET_N_TSECT];  // 录像类型掩码
} SDK_RECORDCONFIG;

//录像设置结构体
typedef struct SDK_RECORDCONFIG_ALL
{
    SDK_RECORDCONFIG vRecordConfigAll[NET_MAX_CHANNUM];
} SDK_RECORDCONFIG_ALL;

// 图片设置
typedef 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
} SDK_SnapshotConfig;

typedef struct SDK_SnapshotConfigAll
{
    SDK_SnapshotConfig vSnapshotConfigAll[NET_MAX_CHANNUM];
}SDK_SnapshotConfigAll;

//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"
    char cIranCalendarEnable;  // 是否启用伊朗日历，1表示启用，0表示不启用
    char cFontSize;            // 矢量字体大小
    char reserved[2];
    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
    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;
    bool bEnable;
    int  dwTimeZone;                // 时区
    char szAddr[MAX_URL_IP_LEN];  // IP地址 域名ntp1.aliyun.com（源）
    int  dwPort;                    // 端口
    int  time_inter;                // 校时间隔
    int  bTimeZoneConversionMode;  // 时区转换模式0:时区转换类型1,1:时区转换类型2
}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];
};

// 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
typedef struct SDK_EncodeConfigAll_SIMPLIIFY
{
    SDK_CONFIG_ENCODE_SIMPLIIFY vEncodeConfigAll[NET_MAX_CHANNUM];
```

```
}SDK_EncodeConfigAll_SIMPLIIFY;

typedef struct  SDK_CombineEncodeConfigAll
{
    SDK_CONFIG_ENCODE vEncodeConfigAll[NET_MAX_COMBINE_NUM];
}SDK_CombineEncodeConfigAll;

typedef struct SDK_CombEncodeParam
{
    int iEncodeMode;  // 见 CombineEncodeMode
}SDK_CombEncodeParam;

struct SDK_CombEncodeModeAll
{
    SDK_CombEncodeParam vEncodeParam[NET_MAX_COMBINE_NUM];
};

// 视频物件结构
typedef 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,是否显示，1：显示 0：隐藏
    bool         bEncodeBlend;    // 编码叠加
}SDK_VIDEO_WIDGET;

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

// 视频物件(输出模式对话框)
typedef struct SDK_VideoWidgetConfigAll
```

```
{
    SDK_CONFIG_VIDEOWIDGET vVideoWidegetConfigAll[NET_MAX_CHANNUM];
}SDK_VideoWidgetConfigAll;

// 所有通道名称标题
typedef struct SDK_ChannelNameConfigAll
{
    char channelTitle[NET_MAX_CHANNUM][NET_NAME_PASSWORD_LEN];
} SDK_ChannelNameConfigAll;

// 输出模式
typedef 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
    int    nOsdAttr;            // OSD属性1：透明,闪烁 2：透明,不闪烁 3：闪烁,不透明
4： 不透明,不闪烁
    int    nOsdColor;           // 字体颜色
    bool   bShowDate;           // 显示日期
    bool   bShowTime;           // 显示几点
    bool   bShowWeek;           // 显示星期
    int    TimePosX;            // 时间距原点x轴位置
    int    TimePosY;            // 时间距原点y轴位置
    int    nDateType;           // 0：YYYY-MM-DD 年月日 1：MM-DD-YYYY 月日年 2：
YYYY年MM月DD日 3：MM月DD日YYYY年
    int    nHourType;           // OSD小时制：0-24小时制,1-12小时制
    bool   bBitrateShow;        // 显示比特率
    int    BitratePosX;         // 时间距原点x轴位置
    int    BitratePosY;         // 时间距原点y轴位置
    OSD_TITLE_S stTitleOSD[4]; // 4个标题
    bool   bClientNumOSD;       // 是否显示用户连接数
} SDK_GUISetConfig;
```

```
// 普通网络设置
typedef 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
    int MaxBps;                             // 限定码流值
    int TransferPlan;                       // 传输策略
    bool bUseHSDownLoad;                    // 是否启用高速录像下载测率
    char sMac[NET_MAX_MAC_LEN];             // MAC地址
    int DataPort;
    int OnvifPort;
} SDK_CONFIG_NET_COMMON;

// DHCP
typedef struct SDK_NetDHCPConfig
{
   bool bEnable;
   char ifName[32];
}SDK_NetDHCPConfig;

// 所有网卡的DHCP配置
#define SDK_MAX_ETH_NUM 4
typedef struct SDK_NetDHCPConfigAll
{
   SDK_NetDHCPConfig vNetDHCPConfig[SDK_MAX_ETH_NUM];
}SDK_NetDHCPConfigAll;

// 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
};

// IP权限设置
typedef 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
}SDK_NetIPFilterConfig;

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

// pppoe设置
typedef 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
}SDK_NetPPPoEConfig;

// DDNS设置
typedef 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, 目前有: JUFENG
  char HostName[NET_NAME_PASSWORD_LEN]; // 主机名
  SDK_RemoteServerConfig Server;            // DDNS服务器
  DWORD   RefreshInterval;                  // 更新间隔
  DWORD   MapHttpPort;                      // 本地WEB服务端口映射到外网的端口号
  DWORD   MapDataPort;                      // 本地数据服务端口映射到外网的端口号
}SDK_NetDDNSConfig;

// DDNS设置
typedef struct SDK_NetDDNSConfigALL
{
    SDK_NetDDNSConfig ddnsConfig[NET_MAX_DDNS_TYPE];
```

```
}SDK_NetDDNSConfigALL;

// FTP设置
typedef 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
    int  FtpPathClassify1;                  // 二级路径0无1按日期2按设备编号3按IP地址
    int  FileUploadMode1;                   // 文件上传方式0直接上传1防读取上传
}SDK_FtpServerConfig;

// NTP设置
typedef struct SDK_NetNTPConfig
{
    bool Enable;                            // 是否开启
    SDK_RemoteServerConfig Server;          // PPPOE服务器
    int  UpdatePeriod;                      // 更新周期
    int  TimeZone;                          // 时区
}SDK_NetNTPConfig;

#define  NET_MAX_EMAIL_TITLE_LEN 64
#define  NET_MAX_EMAIL_RECIEVERS  5
#define  NET_EMAIL_ADDR_LEN  64

// EMAIL设置
typedef struct SDK_NetEmailConfig
{
    bool Enable;                                // 是否开启
    SDK_RemoteServerConfig Server;              // 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
    bool bUseSSL;
    char SendAddr[NET_EMAIL_ADDR_LEN];          // 发送地址
    char Recievers[NET_MAX_EMAIL_RECIEVERS][NET_EMAIL_ADDR_LEN]; // 接收人地
址
    char Title[NET_MAX_EMAIL_TITLE_LEN];        // 邮件主题
    SDK_TIMESECTION Schedule[NET_N_MIN_TSECT]; // email有效时间
}SDK_NetEmailConfig;

// 解码器地址设置
typedef struct SDK_NetDecoderConfig
{
```

```
   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)
}SDK_NetDecoderConfig;

// 解码器地址设置
typedef struct SDK_NetDecorderConfigV3
{
    bool Enable;                                   // 是否开启
    char UserName[NET_NAME_PASSWORD_LEN];          // DDNS类型名称
    char PassWord[NET_NAME_PASSWORD_LEN];          // 主机名
    char Address[NET_NAME_PASSWORD_LEN];
    int Protocol;                                  // 见SDK_TransferProtocol_V2
    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
    char MainRtspUrl[NET_NAME_URL_LEN];            // rtsp协议时前端设备的主码流地址
    char SubRtspUrl[NET_NAME_URL_LEN];             // rtsp协议时前端设备的辅码流地址
} SDK_NetDecorderConfigV3;

// 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
typedefstruct 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
    int nNetDeorde;                // 有多少个
    bool EnSynchResolution;        // 同步分辨率使能开关,0表示打开，1表示关闭
} SDK_NetDigitChnConfig;

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

// UPNP设置
typedef 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
    int UpnpEthNo;                 // UPNP网卡 0－有线 1－无线
    int UpnpMode;                  // UPNP模式 0: 指定 1: 自动
    CONFIG_IPAddress UpnpHostIP;   // 服务器地址
    int SetMapDataPort;            // 希望数据映射端口
    int SetMapHttpPort;            // 希望WEB映射端口
} SDK_NetUPNPConfig;

// WIFI设置
typedef 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
} SDK_NetWifiConfig;
```

```
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
};

typedef 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
} SDK_NetWifiDevice;

typedef struct SDK_NetWifiDeviceAll
{
    int nDevNumber;
    SDK_NetWifiDevice vNetWifiDeviceAll[NET_MAX_AP_NUMBER];
}SDK_NetWifiDeviceAll;

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

// 告警状态
typedef struct SDK_DVR_ALARMSTATE
{
        BYTE iVideoMotion[NET_MAX_MSK_SIZE]; // 移动侦测状态,用掩码表示通道号
,byte0 代表通道一,以此类推 1: 有告警 0: 无告警
        BYTE iVideoBlind[NET_MAX_MSK_SIZE];  // 视频遮挡状态,用掩码表示通道号
,byte0 代表通道一,以此类推 1: 有告警 0: 无告警
        BYTE iVideoLoss[NET_MAX_MSK_SIZE];   // 视频丢失状态,用掩码表示通道号
```

```
,byte0 代表通道一,以此类推 1: 有告警 0: 无告警
        BYTE iAlarmIn[NET_MAX_MSK_SIZE];      // 告警输入状态,用掩码表示通道号
,byte0 代表通道一,以此类推 1: 有告警 0: 无告警
        BYTE iAlarmOut[NET_MAX_MSK_SIZE];     // 告警输出状态,用掩码表示通道号
,byte0 代表通道一,以此类推 1: 有告警 0: 无告警
}SDK_DVR_ALARMSTATE;

// 通道状态
typedef struct SDK_DVR_CHANNELSTATE
{
    bool bRecord;     // 是否正在录像
    int iBitrate;     // 当前码率
}SDK_DVR_CHANNELSTATE;

// DVR工作状态
typedef struct SDK_DVR_WORKSTATE
{
    SDK_DVR_CHANNELSTATE vChnState[NET_MAX_CHANNUM];
    SDK_DVR_ALARMSTATE vAlarmState;
}SDK_DVR_WORKSTATE;

// 支持语言
typedef struct SDK_MultiLangFunction
{
    //每个协议最多由64个字符组成
    int nLangNum;
    char vLanguageName[128][64];
}SDK_MultiLangFunction;

// 支持的视频制式
typedef struct SDK_MultiVstd
{
    int nVstdNum;
    char vVstdName[3][64]; // 每个协议最多由3个字符组成
}SDK_MultiVstd;

// 恢复的默认配置种类
typedef struct SDK_SetDefaultConfigTypes
{
    bool vSetDefaultKinds[SDK_DEFAULT_CFG_END];
    bool vDefaultFactory;          // 恢复出厂设置配置
}SDK_SetDefaultConfigTypes;
```

```
// 语音对讲格式
struct SDK_AudioInFormatConfigAll
{
    SDK_AudioInFormatConfig
vAudioInFormatConfig[SDK_AUDIO_ENCODE_TYPES_NR];
};

// 升级信息获取
typedef struct SDK_UpgradeInfo
{
    char szSerial[64];
    char szHardware[64];
    char szVendor[64];
    unsigned int uiLogoArea[2];
    char szLogoPartType[64];
} SDK_UpgradeInfo;

typedef struct {
    int left;
    int top;
    int right;
    int bottom;
}sdkRect;

// 音频输入格式，语音对讲用
typedef 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
} SDK_AudioInFormatConfig;

// 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  VIDEONET_TALKDECODE_INFO
{
    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];
}VIDEONET_TALKDECODE_INFO;

typedef 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，2-12
    int nprofile;           // 0: baseline 1:main profile 2:high profile
    int nPicLevel;          // 质量0 ~ 5
    int nCaptureVencType;   // 图片类型0:jpg1:bmp
    int dwPicWidth;         // 图片宽
    int dwPicHeight;        // 图片高
} SDK_VIDEO_FORMAT;

typedef struct  SDK_AUDIO_FORMAT
{
    int    nBitRate;            // 码流kbps
    int    nFrequency;          // 采样频率
    int    nMaxVolume;          // 最大音量阈值
    int    nMicLine;            // 音频输入0麦克1线输入
    int    nAencType;           // 压缩格式PT_AENC_TYPE_E
    int    nVolumeIn;           // 音量输入
    int    nVolumeOut;          // 音量输出
} SDK_AUDIO_FORMAT;

// 媒体格式
typedef 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
} SDK_MEDIA_FORMAT;

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

2.1.6 网络报警信息

```
// 网络报警
struct SDK_NetAlarmInfo
{
    int iEvent;   // 目前未使用
    int iState;   // 每 bit 表示一个通道,bit0:第一通道,0-无报警 1-有报警, 依次类推
}
```

2.1.7 存储设备控制信息

```
// 存储设备控制
typedef struct SDK_StorageDeviceControl
{
  int iAction;       // 见 enum SDK_StorageDeviceControlTypes
  int iSerialNo;     // 磁盘序列号
  int iPartNo;       // 分区号
  int iType;     //SDK_StorageDeviceClearTypes 或 SDK_FileSystemDriverTypes
  int iPartSize[4]; // 各个分区的大小
}SDK_StorageDeviceControl;
```

2.1.8 RTSP 信息

```
typedef struct SDK_NetRTSPConfig
{
    bool bServer;
    bool bClient;
    SDK_RemoteServerConfig Server;        // 服务器模式
    SDK_RemoteServerConfig Client;        // 客户端模式
    bool             bEnable;             // RTSP服务开关
    bool             bAuth;               // 是否认证
```

```
    DWORD            dwMTU;              // RTSP MTU  maxPacketSize
    DWORD            dwMultiMode;        // 0: 单播  1: 多播
    CONFIG_IPAddress MulticastIP;        // 多播地址
    DWORD            wVideoPort1;        // primary video port
    DWORD            wAudioPort1;        // primary audio port
    DWORD            wVideoPort2;        // secondary video port
    DWORD            wAudioPort2;        // secondary audio port
    char             byOnvifAuth;        // onvif 密码验证
} SDK_NetRTSPConfig;
```

2.1.9 媒体包以及包信息

```
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
};

// 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
typedef 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

    unsigned int PictureFlip;           // 图片上下翻转
    unsigned int PictureMirror;         // 图片左右翻转(镜像)
    unsigned int RejectFlicker;         // 日光灯防闪功能
    unsigned int EsShutter;             // 电子慢快门功能
    int ircut_mode;                     // IR-CUT切换，0红外灯同步切换，1自动切换
    int dnc_thr;                        // 日夜转换阈值
    int ae_sensitivity;                 // ae灵敏度配置
    int Day_nfLevel;                    //  noise filter 等级，0-5,0不滤波，1-5
值越大滤波效果越明显
    int Night_nfLevel;
    int Ircut_swap;                     // ircut 正常序= 0   反序= 1
    int high_light;                     // 强光抑制功能 0~255,默认是16
    int byPwdFreq1;                     // 照明频率1 (0)60HZ,(1)50HZ (视频制式)
    int byWDRLevel;                     // 宽动态       0 ~ 255
    int byLensCorrection;               // 镜头校正 (0)关 (1)开
    int byHCL;                          // 强光抑制
    int byTRCutLevel;                   //IRCUT切换 (0)低电平有效(1)高电平有效 为
夜晚模式
    int byIRLevel;                      // 夜间启动点(黑转彩)
    int byInfraredDetectMode;           // (0)视频检测(1)时间控制(2)光敏检测
    SDK_DAY_TIME uDayDetecttime;        // 转白天检测时间
    SDK_DAY_TIME uNigntDetecttime;      // 转夜晚检测时间
    bool byphotoresistorLevel;     // 光敏电阻(0)低电平有效 (1)高电平有效为黑夜
    int byIrCToBlackTime;          // 光敏模式，黑转彩转换时间
    int byIrTime;                  // 0~60秒可设，光敏模式，彩转黑转换时间
    int byInfraredLamp;            //红外灯切换 （红外灯方向） (0)低电平有效 (1)高
电平有效  打开红外灯
    int    byIRDayTONigLevel;      // 彩转黑 灵敏度值
    int byLedControl;              // (0)自动 (1)开(2)关（红外灯控制）
```

```
} SDK_CameraParam;

// 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  status;                                 // 工作状态  >= 0 正常    < 0 异常
    int  elecLevel;                              // 参考电平值
    int  luminance;                              // 平均亮度
   char pVersion[64];                            // 版本
   char isFishLens;     // 是否是鱼眼镜头，如果是，则需要app和PC端做图像校正
   char resv1[3];       // 保留，如果增加char型的成员先用这里的
   char reserve[28];    // 保留，如果增加int型的成员用这里
};
```

2.1.10 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
};
```

2.1.11 主动服务

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

2.1.12 子连接类型

```
enum SubConnType
{
    conn_realTimePlay=1,
    conn_talk,
    conn_playback
    conn_facepicture,
    conn_faceCompareResult,
    conn_faceBlackAndWhitelList,
    conn_faceMonitor,
    conn_ListManage,
};
```

2.1.13 连接类型

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

2.1.14 搜索协议类型

```
typedef enum SDK_TransferProtocol_V2
{
    SDK_TRANSFER_PROTOCOL_NETIP,
    SDK_TRANSFER_PROTOCOL_ONVIF,
    SDK_TRANSFER_PROTOCOL_MAC,
    SDK_TRANSFER_PROTOCOL_RTSP,
    SDK_TRANSFER_PROTOCOL_NETIPV6,
    SDK_TRANSFER_PROTOCOL_NR_V2,
    SDK_TRANSFER_PROTOCOL_ONVIF_DEFAULT = 128,
    SDK_TRANSFER_PROTOCOL_ONVIF_NR_V2
}SDK_TransferProtocol_V2;
```

2.1.15 人脸抓拍，比对，黑白名单

```
// 图片属性头
typedef struct SDK_FA_PICTURE_HEAD_S
{
```

```
    WORD   headflag;                // 包头标志 0x4A50
    BYTE   tag;                     // 固定值0x01
    BYTE   frametype;               // 帧类型 0x4A 表示图片
    BYTE   frameinfo;               // 暂时保留
    BYTE   version;                 // 暂时保留
    WORD   messageid;               // 消息ID
    WORD   width;                   // 暂时保留
    WORD   height;                  // 暂时保留
    DWORD   ndatasize;        // 暂时保留
    DWORD   deviceid;         // 用户用来区分设备ID
    DWORD   channelid;        // 用户用来区分通道ID
    DWORD   streamid;         // 用户用来区分图片流ID
    BYTE   picnumber[64];     // 暂时保留
    BYTE   filename[64];      // 图片name
    DWORD  featureEn;
    DWORD  nFeatureLen;
    DWORD  nDevKeyId;         // 设备ID
    BYTE   res[84];           // 扩展使用
    DWORD   headend;          // 结束标志
}SDK_FA_PICTURE_HEAD, *P_SDK_FA_PICTURE_HEAD;

//人脸识别参数
typedef struct  SDK_FA_SMART_FACE_S
{
    int        nEnable;
    int        nquality;            // 灵敏度
    BYTE       nExposureMode;       // 人脸曝光模式
    BYTE       nSnapMode;           // 抓拍模式
    BYTE       nTrackmode;          // 人脸跟踪  0:关闭 1:过滤跟踪 2:非过滤跟踪
    BYTE        nFtpEnable;         // FTP上传开关 0:关闭 1:开启
    int        nFaceSize;           // 人脸识别最小尺寸(30~300)
    BYTE       nFaceBLC;            // 人脸场景模式
    BYTE       nPicMode;            // 图片上传格式(0人脸；1原图；2人脸及原图)
    BYTE       nParamCheckFlag;
    BYTE       byFaceAreaExpand;    // 人脸周边区域扩展系数
    BYTE       byPicPrefixEnable;   // 人脸图片自定义前缀使能
    char       szPicPrefix[16];     // 人脸图片自定义前缀
    int        nTrackFrameNum;      // 快速抓拍帧数-----快速抓拍
    BYTE       bySnapNum;           // 最大抓拍次数-----离开后抓拍模式
    BYTE       byIntervalTime;      // 间隔时间-----间隔抓拍，秒为单位
    BYTE       byFacePicQual;       // 人脸图片编码质量
    BYTE       byFramePicQual;      // 原图编码质量
    int        nIntervalFrame;      // 间隔帧数---间隔抓拍，帧为单位
```

```
    BYTE      byAttributeEnable;  // 人脸属性开关
    BYTE      byIntervalSnapNum;  // 抓拍次数------间隔抓拍模式
    WORD      wMaxFaceSize;       // 人脸识别最大像素
    int       nGateIntervalFrame; // 间隔帧数----闸机模式
    int       nFaceQuality;       // 人脸质量分数
    BYTE      byTimeOsdEnable;    // 原图叠加时间戳OSD (0关闭，1开启)
    BYTE      byFacePicDPI;       // 人脸图片源分辨率(0 4K, 1 1080P)
    BYTE      byFramePicDPI;      // 场景图分辨率(0 4K, 1 1080P)
    BYTE      byResendEnable;     // EMMC断网续传（0关闭，1开启）
    BYTE      byYaw;              // 人脸左右偏转度，偏航角
    BYTE      byPitch;            // 人脸上下角度，俯仰角
    BYTE      byRoll;             // 人脸头歪的角度，翻滚角
    BYTE      byLivenessEnable;   // 比对机活体检测使能(0开启 1关闭)
    BYTE      byCompressFlag;
    BYTE      byRes[1];
    WORD       nCompressSize;
}SDK_FA_SMART_FACE;

// 人脸私有协议上传
typedef struct SDK_FA_FS_PROTOCOL_S
{
    char szUDPaddr[64];     // 服务器IP
    DWORD dwUDPport;        // 服务器端口号
    BYTE nUDPSDKEnable;         // 私有协议上传使能开关
    BYTE szRes[3];          // 保留
    BYTE byRes[64];         // 保留
}SDK_FA_FS_PROTOCOL;

typedef struct  SDK_FA_SNAP_FACE_CONFIG_S
{
    SDK_SCHWEEK_TIME    aTime;        // 布防时间
    SDK_FA_SMART_FACE   sSmartFace;
    SDK_FA_FS_PROTOCOL  stFsProtocols;
}SDK_FA_SNAP_FACE_CONFIG;

// 人脸属性检测
typedef struct _FACE_ATTRIBUTE_S
{
    char byAge;          //年龄 [0, )
    char byGender;       //性别 -1 未知, 1 男, 2 女
    char byGlasses;      //眼镜   -1 未知, 0 不戴眼镜, 1 戴眼镜, 2 戴墨镜
    char byMask;         //口罩 -1 未知, 0 不戴口罩, 1 戴口罩
    char byRace;         //种族   -1 未知, 1 黄种人, 2 黑种人, 3 白种人
```

```
    char byEye;          //眼睛 -1 未知, 0 闭合, 1 睁开
    char byMouth;        //嘴巴 -1 未知, 0 闭合, 1 张开
    char byBeard;        //胡子   -1 未知, 0 无胡子, 1 有胡子
    char byEmotion;      //表情   -1 未知, 1 生气, 2 平静, 3 高兴, 4 悲伤, 5 惊讶, 6 害怕, 7 厌恶, 8 斜视, 9 尖叫
    char byAttractive;   //魅力 [0, ）
    char bySmile;        //笑脸 -1 未知, 0 微笑, 1 平静
    char bySunglass;     //墨镜 -1 未知, 0 不戴墨镜, 1 戴墨镜
    char byHat;          //带帽 -1 未知, 0 不戴帽子, 1 戴帽子
    char byIsCalling;    //打电话 -1 未知, 0 没打电话, 1 在打电话
    char byUpperbodyColor; //上衣颜色 -1 未知, 0 黑色, 1 白色, 2 灰色, 3 红色, 4 蓝色, 5 黄色, 6 橘色, 7 棕色, 8 绿色, 9 紫色, 10 青色, 11 粉红色, 12 条纹, 13 格子, 14 图案
    char byUpperbodyStyle; //上衣款式 -1 未知, 0 背心, 1 T 恤, 2 衬衫, 3 罩衫, 4 连衣裙, 5 卫衣, 6 夹克, 7 针织毛衣, 8 西装, 9 马甲, 10 大衣, 11 羽绒服, 12 牛仔服
    char byHairStyle;    //发型 -1 未知, 0 光头, 1 秃头, 2 平头, 3 短发, 4 及肩短发, 5 披肩发, 6 束发
    char byRes[15];      //预留字节
}FACE_ATTRIBUTE_S;

// 人脸图片尺寸
typedef struct _FACEPIC_SIZE_S
{
    int X;
    int Y;
    int nWidth;
    int nHeight;
}FACEPIC_SIZE_S;

// 人脸坐标
typedef struct _FACE_POSITION_S
{
    int left;
    int top;
    int right;
    int bottom;
}FACE_POSITION_S;

typedef struct _SDK_PICTURE_HEAD_S   //256
{
    WORD   headflag;
    BYTE   tag;
    BYTE   frametype;
```

```
    BYTE   frameinfo;
    BYTE   version;
    WORD   messageid;
    WORD   width;
    WORD   height;
    DWORD  ndatasize;
    DWORD  deviceid;
    DWORD  channelid;
    DWORD  streamid;
    BYTE   picnumber[64];
    BYTE   filename[64];
    DWORD  featureEn;
    DWORD  nFeatureLen;
    DWORD  nDevKeyID;       //设备编号
    DWORD  nAttrInfoLen;    //人脸属性信息大小
    DWORD  nCmpInfoLen;     //比对详细信息大小
    DWORD  nUserInfo1Len;   //用户自定义信息大小
    DWORD  nFacePicLen;     //抓拍图片大小，等于0时没有数据
    DWORD  nFramePicLen;    //背景图片大小，等于0时没有数据
    DWORD  nLibPicLen;      //底库图片大小，等于0时没有数据
    DWORD  nBodyPicLen;     //人形照大小
    DWORD  nPicTotalLen;    //图片总大小
    BYTE   res[52];
    DWORD  headend;
} SDK_PICTURE_HEAD, *PSDK_PICTURE_HEAD;

// 车牌信息
typedef struct _SMART_CAR_INFO_XM_S
{
    char szCarPlateNum[16];    //车牌号
    char szCarLogName[20];     //主品牌
    char szCarSubLogName[20];  //子品牌
    int  nCarColor;            //车身颜色
    char byRes[100];
}SMART_CAR_INFO_XM_S;

//车牌抓拍参数
typedef struct _SDK_SMART_CAR_S
{
    char      byCarDetectEnable;  // 算法使能开关(1开，0关)
    char      byFTPEnable;        // FTP传输(1开，0关)
    char      byPicMode;          // 大小图(0场景图，1车牌图片，2场景图加车牌图)
    char      bySameCarIntervel;  // 相同车牌抓拍间隔(秒)（1~255）
```

```
    char      bySnapMode;          // 抓拍模式:线圈、区域、地感（不支持）
    char      byDetectAspect;      // 抓拍方向（安装环境） 1:由上至下（入口） 2:由
下至上（出口） 3:双向（卡口）
    short     nMinSize;            // 最小识别宽度（像素）
    short     nMaxSize;            // 最大识别宽度（像素）
    short     nLocalProvince;      // 默认省份（不支持）
    SDK_SCHWEEK_TIME    stCarSchTime;    // 布防时间336bytes（不支持）
    char      byBigPicQuality;     // 大图质量（0~99）
    char      byLittlePicQuality; // 小图质量（0~99）
    char      byThreshold;         // 抓拍阈值（不支持）
    char      byRes[161];
}SDK_SMART_CAR_S;

// wifi探针信息
typedef struct _SMART_WIFIPROBE_INFO_XM_S
{
    int   nSignalIntensity;     // wifi信号强度
    int  nChannel;              // 信道
    char szPhoneWifiMac[20];    // 手机MAC
    char szRouteMac[20];        // 路由MAC
    char byRes[112];
}SMART_WIFIPROBE_INFO_XM_S;

// 图片属性
typedef struct _FACE_ATTR_S
{
    int nvisits;                        // 来访次数 用于比对机
    int nPicType;                       // 图片类型
    int nFaceQuality;                   // 质量分数
    int nPicLen;                        // 图片大小
    int nPicID;                         // 图片计数
    int nFaceID;                        // 人脸ID（人脸、车牌用到）
    int nFaceOrHead;                    // 人脸和人头区分(1人头，2人脸)
    int  byIsScenePic;                  // 场景图标识（0：小图，1：场景图）
    FACE_ATTRIBUTE_S stFaceAttribute; // 人脸属性
    FACEPIC_SIZE_S  stFacePicSize;      // 用于人脸抠图，原图不用
    FACE_POSITION_S stFacePosition;     // 人脸坐标位置
    char stFaceTime[32];                // 抓拍时间
    char stDevUUID[32];                 // 设备UUID
    char stOsd[64];                     // OSD信息
    union{
        SMART_CAR_INFO_XM_S stCarInfo;              // 车牌信息
        SMART_WIFIPROBE_INFO_XM_S stWifiProbeInfo;  // wifi探针信息
```

```
    };
    int  nFeatureType;                  // 特征值类型:0 float, 1 char ,2 int
    int  nFeatureLen;                   // 结合nFeatureType类型，特征值数组长度
    union{
        float  fFeature[512];
        char   cFeature[2048];
        int    nFeature[512];
    }facefeature;

    char stFaceUTCTime[32];     // 抓拍的UCT时间
    char byHasFaceAttr;             // 是否带人脸属性信息

    char stFacePicname[80];     // 抓拍图片名
    char stFramePicname[80];    // 背景图片名
    int  nFacePicLen;           // 抓拍人脸图大小
    int  nFramePicLen;          // 原图图片大小

    int  nLiveNess;             // 活体检测 0 表示未检测或者检测未成功  1 表示活体
    float fLiveScore;           // 活体分数
    char szBodyPicPath[80];     // 人形照
    int  nBodyPicLen;           // 人形照大小
    int  nStructureInfoLen;     // 结构化信息长度
    char sRes[332];             // 预留
}SDK_FACE_ATTR,*PSDK_FACE_ATTR;
```

// 比对结果使能设置

```
typedef struct _sdk_compinfo_flag
{
    int           iCaptureEn;       // 设置相机发送抓拍图使能
    int           iLibPicEn;        // 设置相机发送底库图使能
    int           iAlarmEn;             // 设置相机发送报警事件使能（预留）
    int           iFeatureEn;       // 设置相机发送人脸特征值使能
}SDK_COMPINFO_FLAG;

typedef struct SDK_FILEINDEX_INFO_EX_S  //1024
{
    int    syslistid;        // 系统ID 在添加时不用赋值，由系统生成
    int    libindex;         // 库编号 在添加时不用赋值，由系统生成
    int    fileindex;        // 图片索引编号 在添加时不用赋值，由系统生成
    int    enable;           // 使能开关
    int    timelimit;        // 名单有效期 0 永久有效,1 周期有效,有效时间参照sTime
和 eTime
    int    listmode;         // 名单模式: 1 黑名单，2白名单，3 VIP名单
```

```
    int    count;           // 统计次数
    int    sex;             // 性别
    char   usrid[36];       // 用户ID或者编号
    char   idcard[32];      // 身份ID
    char   name[64];        // 姓名
    char   picname[64];     // 图片名    系统生成，不支持修改
    char   nation[12];      // 民族
    char   birth[24];       // 生日
    char   phone[16];       // 号码
    char   areacode[32];    // 区域编码
    char   usrcode1[32];    // 社区编码
    char   usrcode2[32];    // 小区编码
    char   type[12];        // 类型
    char   dTime[20];       // 入库时间
    char   sTime[20];       // 开始时间
    char   eTime[20];       // 结束时间
    int    usrparam1;       // 备用参数1
    int    usrparam2;       // 备用参数2
    int    usrparam3;       // 备用参数3
    int    usrparam4;       // 备用参数4
    int    usrparam5;       // 备用参数5
    char   resID[32];       // 房屋ID     //备用ID
    char   areaAddress[100];// 区域地址
    char   resname[32];     // 房屋名称 //备用名称
    char   usrdata1[32];    // 用户备用数据1,
    char   usrdata2[24];    // 用户备用数据2
    char   usrdata3[32];    // 用户备用数据3
    char   usrdata4[36];    // 用户备用数据4
    char   usrdata5[88];    // 用户备用数据5
    char   remark[180];     // 标签
}SDK_FILEINDEX_INFO_EX;

typedef struct _SDK_FACE_COMPARE_INFO_S//2048
{
    float fCompareResult;             // 比对相似度
    int  nFaceAlarm;                  // 人脸报警
    char stCompareTime[32];           // 比对时间
    SDK_FILEINDEX_INFO_EX aiFaceLib; // 人脸底库信息
    char stRes[984];
}SDK_FACE_COMPARE_INFO;
```

// 比对结果

```
typedef struct SDK_FACECOMPARE_RESULT_S
```

```
{
    DWORD         deviceid;
    DWORD         channelid;
    DWORD         streamid;
    DWORD         wbmode;                  // 黑白名单
    DWORD         libindex;                // 黑白名单库索引
    DWORD         libnumber;               // 黑白名单库里编号
    char          picnameComp[64];         // 实时比对库图片名字
    char          *pCompImgData;           // 图片数据指针
    int           iCompImgSize;            // 图片数据长度
    char          picnameSnap[64];         // 实时比对图片名字
    char          *pSnapImgData;           // 图片数据指针
    int           iSnapImgSize;            // 图片数据长度
    float         result;                  // 比对结果
    BYTE          SnapTime[32];            // 抓拍时间
    long long     TCPSocket;               // 存放对应比对链路的类指针转换后的值
    DWORD         guest;                   // 客流统计
    DWORD         Age;                     // 年龄
    DWORD         Gender;                  // 性别
    DWORD         Glasses;                 // 眼镜
    DWORD         Mask;                    // 面具
    DWORD         Race;                    // 种族
    DWORD         Eye;                     // 眼睛
    DWORD         Attractive;              // 魅力值
    DWORD         Beard;                   // 胡须
    DWORD         Emotion;                 // 表情
    BYTE          ByRes[117];              // 预留
    int           facealarm;               // 人脸报警, 预留
    float         facecharacter[512];
}SDK_FACECOMPARE_RESULT;

#define SDK_DB_NUM 3   //DB_NUM分别对应  白名单0  黑名单1  VIP名单2
//黑白名单链路用来返回传入的图片获取到的特征值以及报警信息
typedef struct SDK_Feature_Compare_RESLUT
{
    int    Queuesize;                     // 当前放入获取特征值的队列的图片个数
    int    Percentage;                    // 当前放入获取特征值的队列的图片个数占
总的缓存图片的百分比  -1表示获取百分比失败
    float  fQuality;                      // 人脸的综合质量
    float  fConfidence;                   // 人脸的置信度  正脸率
    float  fSimilarity[SDK_DB_NUM];       // 相似度
    float  fThreshold[SDK_DB_NUM];        // 阈值
    char   libPicPath[SDK_DB_NUM][64];
```

```
    char   id[SDK_DB_NUM][32];
    int    extLen;                          // 结构体之后紧接的数据长度
    char   pdata[0];                        // 暂时用来存放特征值的buf
}SDK_Feature_Compare_RESLUT_S;

typedef struct SDK_Feature_RESLUT
{
    float  fSimilarity[SDK_DB_NUM];         // 相似度
    int    sex[SDK_DB_NUM];                 // 性别
    char   libPicPath[SDK_DB_NUM][64];
    char   id[SDK_DB_NUM][32];
    int    extLen;                          // 结构体之后紧接的数据长度
    char   pdata[0];                        // 存放附加信息，暂时为空
}SDK_Feature_RESLUT_S;

//返回特征值,图片质量,正脸率
typedef struct SDK_NAS_PIC_QUALITY {
    float  fQuality;            // 人脸的综合质量
    float  fConfidence;         // 人脸的置信度
    int    s32FeatureLen;       // 人脸特征的特征向量长度
    float  fFeature[512];       // 人脸特征的特征向量
}SDK_NAS_PIC_QUALITY;

// 黑白名单
typedef struct SDK_FA_BLACKANDWHITELIST_CONFIG_S
{
    int                byOutMode;       // 1 库文件索引; 2 图片
    int                byWbMode;           // 0 陌生人;1 黑名单;2 白名单;3 VIP名
单
    int                byAllNums;       // 实际库数目
    int                byLibIndex;      // 库索引
    int                byFileIndex;     // 文件索引
    int                byDataSize;      // 文件大小
    char               byFileName[64]; // 图片名
    int                byErrorCode;     // 错误码，参照XM_WB_ERROR_CODE
    int                byDataType;      // 特征值类型  0 float,1 char ,2 int
    BYTE               byRes[28];
}SDK_FA_BLACKANDWHITELIST_CONFIG;

// 用来给VideoNet_OperationBlackAndWhiteListData_Ex接口传参
// type值为0时，修改人名，图片名第一个下划线后面的数字一定要正确，代表图片id号，图片底
库里面必须有这个id号
// type值为1时，picname为需要替换的名字的完整路径
```

```
typedef struct SDK_WEBLIST_MODIFY_MSG
{
    int type;
    char srcname[128];
    char picname[128];
    char name[64];
    int len;
    char pdata[0];
}SDK_WEBLIST_MODIFY_MSG_S;

typedef struct _SMART_WB_CONFIG_EX      //2048
{
    int    iOpMode;          // 名单库操作模式，内容见WBLIST_ACTION_TYPE值
    int    iWbmode;          // 名单模式，内容见WBLIST_MODE值
    int    iOpFlag;          // 操作唯一标识:0 通过i_sUserID来操作，1 通过
i_iSyslistID来操作，默认0
    int    iSyslistID;       // 名单系统ID;
    char   sUserID[36];      // 用户ID，唯一标识
    char   sFaceID[36];      // 人脸ID，用于表示同一用户多张人脸ID,如果不需要的话就
不用赋值
    int    iBodyLen;         // 配置结构体长度，主要是指sdk_fileindex_info_Ex1长
度，如果为0，则没有该参数
    int    iErrorCode;       // 错误码
    int    iLibIndex;        // 库索引
    int    iFileIndex;       // 文件索引
    int    iAllNums;         // 实际库数目
    int    iSyslistid;
    int    iFeatureType;     // 特征值类型  0 float,1 char ,2 int
    int    iDataSize;
    char   sRes0[104];       // 预留
    SDK_FILEINDEX_INFO_EX   stUserInfo;      //用户名单详细信息，在添加、更新时候
需要添加该信息
    char   sRes[800];        // 后期可用于SDK_FILEINDEX_INFO_EX扩展
}SDK_SMART_WB_CONFIG_EX;


// 人脸报警   384字节
typedef struct SDK_SMART_ALARM_S
{
    BYTE          aEnable;              // 报警开关
    BYTE          aBEnable;             // 黑名单报警开关
    BYTE          aWEnable;             // 白名单报警开关
    BYTE          aIoEnable;            // IO输出使能
```

```
    short         aIoTime;              // IO输出时间
    BYTE          aIoType;              // IO输出类型
    BYTE          aSimilarity;          // 相似度 (0-100)
    SDK_SCHWEEK_TIME aTime;             // 布防时间
    short         aCmpMode;             // 识别模式 (0单次识别  ,1 一直识别)
    short         aCmpNum;              // 识别次数 (1-10)

    //手机APP消息推送
    BYTE          aPushEnable;          // 推送开关
    BYTE          aBPushEnable;         // 黑名单消息推送
    BYTE          aWPushEnable;         // 白名单消息推送
    BYTE          aNWPushEnable;        // 非白名单消息推送

    //20180528
    BYTE          aAudioInpute;         // 音频输入
    BYTE          aVEnable;             // VIP报警
    BYTE          aVPushEnable;         // VIP消息推送
    BYTE          aNonWhite;            // 非白名单报警

    int           aAlarmInfoType;       // 推送消息内容类型:  0 中文信息  1英文信息
    2 手机默认报警信息
    BYTE          aWSimilarity;         // 相似度 (0-100)      白名单相似度    比对
服务器使用
    BYTE          aVSimilarity;         // 相似度 (0-100)      VIP名单相似度    比
对服务器使用
    BYTE          aIoOutType;           // IO输出类型 (0)持续输出 (1)脉冲输出
    BYTE          byRes[22];
}SDK_SMART_ALARM;


// 人脸算法版本信息查询
typedef struct SDK_FA_SMART_VERSION_INFO_S
{
    char szLib_version[16];     // 算法库版本
    char szModel_version[16];   // 模型版本
    BYTE  nbyLic;               // 保留
    BYTE  byRes[7];             // 保留
}SDK_FA_SMART_VERSION_INFO;

// 人脸统计参数查询
typedef struct SDK_FA_SMART_FACE_COUNT_S
{
    int           nEnable;          // 人脸统计 开启/关闭
```

```
    int           nTotalClear;     // 人脸统计值 清零
    long          lBeginTime;
    long          lEndTime;
    int           nFaceTotleNum;
    int           nFaceTotal;      // 人脸统计值
    int           nPerSecFaceNum;  // 单位时间统计人脸数量
    char          byRes[232];
}SDK_FA_SMART_FACE_COUNT;

// 图片查询条件，inputtype为0时按照传图片路径的方式，1按照传图片指针的方式
typedef struct PICTURE_FINDINFO
{
    int    lChannel;               // 通道号:掩码的方式
    char   sorttype;               //排序方式，0按相似度排序，1按时间排序(从小到大)，2按时间排序(从大到小)
    char   inputtype;              // 0传入图片的路径  1传入图片的地址
    short  byFileType;             // 查找图片类型：见SDK_File_Type
    int    photosize;              //对应inputtype值，0传入的图片的路径长度，1传入图片的长度
    float  similarity;             // 相似度
    SDK_SYSTEM_TIME  startTime;    //  查找图片的开始时间
    SDK_SYSTEM_TIME  endTime;      //查找图片的结束时间
    char   *photodata;             // inputtype 0图片完整路径(包含文件名)  1传入图片的地址
}*LPPICTURE_FINDINFO;

typedef struct PICTURE_DATA
{
    short            index;
    short            channel;         // 图片或者录像所属通道号
    int              size;            // 图片大小或者录像时间
    float            similarity;      // 相似度
    SDK_SYSTEM_TIME  capturetime;     // 图片抓拍时间或者录像开始时间
    char             filepath[128];   // 图片或录像完整路径(包含文件名)
}*LPPICTURE_DATA;

//图片查询返回
typedef struct VIDEONET_PICTURE_DATA
{
    int              photonum;        // 图片数目
    PICTURE_DATA     photodata[5000];
}*LPVIDEONET_PICTURE_DATA;
```

```
// EMMC卡信息
typedef struct SDK_EMMC_MODEL_INFO_S
{
    unsigned int  nTotalSize;  // 总空间(单位:M)
    unsigned int  nFreeSize;   // 剩余可用空间 (单位:M)
    BYTE byIsInit;             // 格式化状态(0:未格式化 1:已经格式化)
    BYTE byEmmcFormatFlag;     // 格式化标志位(赋值1时将重新格式化EMMC模块)
    BYTE byRes[54];
}SDK_EMMC_MODEL_INFO;

// 白平衡模式
typedef struct SDK_AWBAUTO_MODE_S
{
    DWORD dwAwbMode;     // 白平衡模式
    DWORD dwAutoAwb;     // 白平衡   0自动, 1 手动
    DWORD dwAwBblue;     // 白平衡   蓝  0~255
    DWORD dwAwBRed;      // 白平衡   红  0~255
    DWORD dwAwBGreen;    // 白平衡   绿  0~255
}SDK_AWBAUTO_MODE;

// 点名球参数设置
typedef struct SDK_MS_NAMING_S
{
#define CMD_NAMING_SET_PARAM      0x1  // 标定区域    nData[1-4]
#define CMD_NAMING_SET_ZOOM       0x2  // 设置倍率    nData
#define CMD_NAMING_SET_SCANMODE   0x3  // 设置扫描模式 nData
#define CMD_NAMING_SET_SCANTIMES  0X4  // 扫描开始后扫描n次后停止 nData[1-4]

#define CMD_NAMING_SCAN_START     0x20 // 扫描开始
#define CMD_NAMING_SCAN_STOP      0x21 // 扫描手动停止

    int nCmd;
    int nData;                          // [1,4]
    char byRes[128];
}SDK_MS_NAMING;

//点名球参数获取
typedef struct SDK_GET_NAMING_PARAM_S
{
    char byEnable;       // 0:  1：功能可用
    char byCalibrate;    // 0:未标定  1:已标定存在轨迹或标定区域点名可用。
    char byDomeState;    // 球机空闲状态0：空闲 1：球机点名中 2：轨迹学习中/标定
    char byRes[128];
```

```
}SDK_GET_NAMING_PARAM;


typedef struct _SDK_FACEMONITOR_INFO_FLAG
{
    int nDeviceId;        // 用户输入参数
    int nChannelId;       // 用户输入参数
    int nStreamId;        // 用户输入参数
    int nFaceAttrEn;      // 设置相机发送图片属性，用于请求FACE_ATTR数据，所有机型默认打开
    int nFaceCmpInfoEn; // 图片比对信息使能，用于请求FACE_COMPARE_INFO数据，不用该数据不用请求
    int nUserExDataEn;    // 留给客户扩展，NVR不用请求
    int nCaptureEn;       // 抓拍人脸图使能，用于请求抓拍图片数据，不需要可以不用请求
    int nFramePicEn;      // 背景图使能，用于请求抓拍图片时对应的背景图数据，不需要可以不用请求
    int nLibPicEn;        // 人脸库图片，用于请求比对成功时的人脸库图片数据  ，不需要可以不用请求
    int nBodyPicEn;       // 传送人形图使能
}SDK_FACEMONITOR_INFO_FLAG;

// 国标信息配置结构体
typedef struct SDK_GB28181_CONFIG_INFO_S
{
    char          serverID[32];
    char          serverAddr[64];
    unsigned int serverPort;
    char          deviceID[32];
    unsigned int devicePort;
    char          userName[32];              // 用户名
    char          password[64];              // 密码
    char          chanid[32];                // 媒体通道ID
    char          alarmid[32];               // 报警ID
    char          administrativeregion[32];  // 行政区域
    char          deviceAssignment[32];      // 设备归属
    char          policeregion[32];          // 警区
    char          manufacturer[32];          // 制造商
    char          secrecyAttr[32];           // 保密属性
    char          secrecyAttr1[32];
    int           longitude;
    int           latitude;
    int           activeTime;                // 心跳时间
    int           activeCount;               // 心跳次数
```

```
    int           registEffectTime;          // 注册有效时间
    char          deviceName[32];
}SDK_GB28181_CONFIG_INFO;

// 中心服务器配置结构体
typedef struct _SDK_CENTER_SERVER_CONFIG_S
{
    int    bEnable;
    char   szServerUrl[64];
    int    dwServerPort;
    char   szDeviceID[32];
    char   username[32];
    char   password[32];
}SDK_CENTER_SERVER_CONFIG;

//人脸监视返回信息
typedef struct _SDK_FACE_MONITOR_INFO_S
{
    SDK_PICTURE_HEAD        *stPictureHead;      // 图片头信息
    SDK_FACE_ATTR           *stFaceAttr;         // 人脸信息
    SDK_FACE_COMPARE_INFO   *stFaceCompareInfo; // 比对信息
    char                    *pUserData;          // 用户定义数据长度,最多1024
    char                    *pSnapPicture;       // 抓拍图片数据
    char                    *pBackGroundPic;     // 原图
    char                    *pLibPicture;        // 库图片数据
    char                    *pBodyPicture;       // 人形图
}SDK_FACE_MONITOR_INFO,    *LPSDK_FACE_MONITOR_INFO;

typedef struct _SDK_VPNConfig
{
    bool             bEnable;                            // 网络是否使用
    char             szAddr[MAX_URL_IP_LEN];             // IP地址 域名
    char             szUserName[MAX_USER_PWD_LEN];       // 用户名
    char             szPassword[MAX_USER_PWD_LEN];       // 密码
    bool             bOnline;                            // 状态
    CONFIG_IPAddress dwIPAddr;                           // IP地址
    DWORD            dwSubmask;                          // 子网掩码
    char             szRes[64];
}SDK_VPNConfig;

typedef struct _SDK_NETAPP_REPORTIP_S
{
    DWORD            dwSize;
```

```
    bool            bEnable;
    DWORD           dwRefreshInterval;      // 刷新间隔
    DWORD           dwIPAddr;
    char            byRes[20];
}SDK_NETAPP_REPORTIP_S;

typedef struct _JIUXIANG_CONFIG_S
{
    int    bEnable;
    char   serverip[128];       // (或域名)
    int    serverport;
    char   serverpath[128];
    int    resendnum;           // 重传次数
    int    bImgEnable;          // 图片：0不上传图片 1上传人脸图 2上传人脸及原图
    int    nListenPort;
    char   szDeviceId[32];      // 自定义设备ID 中性开放 定制使用
//HTTP_SEND_DEEPCAM_1400

//消息类型:
//0:所有人上传 1: 比对成功上传 2:黑名单上传 3:白名单上传 4: VIP名单上传
//5:陌生人上传 6: 非白名单上传
    int        msgType;
    int        featureEn;               // 特征值使能: 0不上传特征值 1 上传特征值
    int        bKeepAliveEnable;
    char       keepaliveaddr[64];       // (ip或域名)
    int        keepaliveport;
    char       keepalivepath[128];
    int        keepaliveInterval;       // 心跳间隔 (秒)
    int        bRemoteUpgrade;          // 远程升级
    int        bGetVersionInterval;     // 检查版本时间间隔(分钟)
    char       GetVersionURL[128];          // 获取版本地址
    char       postUrl[128];            // 推送识别信息地址
    char       postKeepAliveUrl[128];   // 推送心跳信息地址
    int        bSignEnable;             // 用户权限验证 //#if HTTP_CHECK_SIGN
    int        nHTTPVersion;
} SDK_JIUXIANG_CONFIG_S;

typedef struct _SDK_HTTPAPI_CONFIG_S
{
    int     bCaptureEnabled;    // 上传抓拍信息
    int     nResendTimes;       // 重传次数
    int     bFacePicture;       // 上传 人脸图片数据
    int     bRegisterEnabled;   // 注册功能
```

```
    int     nCaptureType;       // 消息类型: //0:所有人上传 1: 比对成功上传 2:黑名单
上传 3:白名单上传 4: VIP名单上传 5:陌生人上传 6: 非白名单上传
    int     bHeartbeatEnabled;          // 上传心跳信息
    int     nHeartbeatInterval;         // 心跳间隔 (秒)
    char    sCaptureAddress[128];     // 抓拍信息推送地址
    char    sRegisterAddress[128];    // 注册信息推送地址
    char    sHeartbeatAddress[128];   // 推送心跳信息地址
    int     bSignCheck;               // Sign 验证 //#if HTTP_CHECK_SIGN
    int     bFaceInfo;                // 上传 FaceInfo
    int     bCompareInfo;             // 上传 CompareInfo
    int     bBackgroundPicture;         // 上传 背景图片数据
    int     bPersonPhoto;             // 上传人员相片数据
    int     nHTTPVersion;             // HTTP版本
} SDK_HTTPAPI_CONFIG_S;

typedef struct SDK_NetMoblieConfig_s
{
    char HKUUid[64];
}SDK_NetMoblieConfig;

typedef struct SDK_NetHighSet_s
{
    int    byMaxAgc;                  // 最大增益值   (模拟增益增益)
    int    byAutoMaxDGainExLevel;     // DGAIN  vaule
    int    byAutoMaxDGainEx;          // 0自动, 1 手动   DGAIN
    int    byAutoMaxAGainEx;          // 0自动, 1 手动   AGAIN
    int    byAutoExposureEx;          // 0自动   1 手动  曝光
    int    wExpTimeMax;               // 自动曝光快门最大值  (曝光时间)
    int    byAutoAwb;                 // 自动白平衡   0自动, 1 手动
    int    byAwbRed;                  // 白平衡 红    0 ~ 255
    int    byAwbGreen;                // 白平衡 绿    0 ~ 255
    int    byAwBblue;                 // 白平衡 蓝    0 ~ 255
    int    byAntiFog;                 // 去雾
    int    byAntiFalseColor;          // 去伪彩
    int    byAntiDIS;                 // 数字防抖
}SDK_NetSeniorSet;
```

3 接口定义

3.1 SDK 初始化

1. long **VideoNet_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。

2. long **VideoNet_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：

  [in]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 VideoNet_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。

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

3. void **VideoNet_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：VideoNet_Init。
- 典型应用：在应用程序关闭时调用。

3.2 报警状态获取

4. bool **VideoNet_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 VideoNet_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：

  [in] 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。
- 返回值：TRUE 回调函数执行正确，FALSE 执行错误。
- 相关函数：VideoNet_SetupAlarmChan、VideoNet_CloseAlarmChan。

5. long **VideoNet_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 VideoNet_SetDVRMessCallBack 的设置值回调出来。
- 参数说明：

  [in] lLoginID

  VideoNet_Login 的返回值。
- 返回值：成功返回 TRUE，失败返回 FALSE。
- 相关函数：VideoNet_SetDVRMessCallBack，VideoNet_CloseAlarmChan。
- 典型应用：在设备连接后调用本函数进行消息订阅。

6. bool **VideoNet_CloseAlarmChan(**long **lLoginID);**

- 函数说明：停止对某个设备侦听消息。
- 参数说明：

  [in] lLoginID

  VideoNet_Login 返回值。
- 返回值：成功返回 TRUE，失败返回 FALSE。

- 相关函数：VideoNet_SetupAlarmChan。
- 典型应用：参见 demo 程序。

3.3 设备注册

7. long **VideoNet_Login (**char *__sDVRIP__, unsigned short __wDVRPort__, char *__sUserName__, char *__sPassword__, LPVIDEONET_DEVICEINFO __lpDeviceInfo__, int *__error__,int __socketType = 0);__

- 函数说明：注册用户到设备。
- 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，成功返回登陆 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：VideoNet_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。

8. long **VideoNet_Login_P2P**(char *__puuid__, unsigned short __wDVRPort__, char

*sUserName, char *sPassword, LPVIDEONET_DEVICEINFO lpDeviceInfo, int *error,int socketType = 0);

- 函数说明：广域网环境下，使用 P2P 模式注册用户到设备。
- 参数说明：

  [in] puuid

  设备关联的 UUID。

  [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，成功返回登陆 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：VideoNet_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。

9. long **VideoNet_Logout(**long **lLoginID);**

- 函数说明：注销设备用户。
- 参数说明：

  [in]lLoginID

  VideoNet_Login 的返回值。

- 返回值：成功返回 TRUE，失败返回 FALSE。

- 相关函数：VideoNet_Login。
- 典型应用：当需要设备主动断开时调用。

3.4 实时监视

10. long **VideoNet_RealPlay(**long **lLoginID,** LPVIDEONET_CLIENTINFO **lpClientInfo);**

- 函数说明：启动实时监视。
- 参数说明：

  [in] lLoginID

  VideoNet_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 VideoNet_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：VideoNet_StopRealPlay，VideoNet_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 VideoNet_SetRealDataCallBack 设备的回调得到原始数据，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。

11. bool **VideoNet_StopRealPlay(**long **lRealHandle);**

- 函数说明：停止实时监视。
- 参数说明：

  [in] lRealHandle

  VideoNet_RealPlay 的返回值。

- 返回值：成功返回 TRUE，失败返回 FALSE。
- 相关函数：VideoNet_RealPlay。

12. bool **VideoNet_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

  VideoNet_RealPlay 的返回值。

  [in] 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。
- 相关函数：VideoNet_RealPlay、VideoNet_StopRealPlay。

3.5 回放和下载

13. long **VideoNet_FindFile(**long **lLoginID,** VideoNet_FINDINFO* **lpFindInfo,** VideoNet_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

  VideoNet_Login 的返回值。

  [in] lpFindInfo

  查询条件 VideoNet_FINDINFO。

  [out] nriFileinfo

  返回的录像文件信息，是一个 VideoNet_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(VideoNet_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：VideoNet_Login、VideoNet_PlayBackByName、VideoNet_StopPlayBack，VideoNet_PlayBackControl，VideoNet_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。

14. long **VideoNet_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

  VideoNet_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。

15. long **VideoNet_GetFileByName(**long **lLoginID,**VideoNet_FILE_DATA ***sPlayBackFile,**char ***sSavedFileName,** fDownLoadPosCallBack **cbDownLoadPos = NULL,** long **dwDataUser = NULL** ,fRealDataCallBack **fDownLoadDataCallBack = 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

  VideoNet_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。

[out] cbDownLoadPos

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

[in] dwUserData

下载进度回调用户自定义数据。

[in] fDownLoadDataCallBack

数据回调。

- 返回值：成功返回下载 ID，失败返回 0。
- 相关函数：VideoNet_StopGetFile、VideoNet_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。

16. long **VideoNet_GetFileByTime(**long **lLoginID,** VIDEONET_FINDINFO* **lpFindInfo,** char ***sSavedFileDIR,** bool **bMerge = false,**fDownLoadPosCallBack **cbDownLoadPos = NULL,** long **dwDataUser = NULL ,**fRealDataCallBack **fDownLoadDataCallBack =NULL)**

- 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

VideoNet_Login 的返回值。

[in] lpFindInfo

查询条件 VideoNet_FINDINFO。

[in] sSavedFileDIR

下载文件保存的目录。

[in] bMerge

是否将下载的文件合并成一个文件。

[out] cbDownLoadPos

下载进度回调函数，可以为空，用户自己调用 VideoNet_GetDownloadPos 得

到进度。

[in] dwUserData

下载进度回调用户自定义数据。

[in] fDownLoadDataCallBack

数据回调。

- 返回值：成功返回下载 ID，失败返回 0。

17. bool **VideoNet_StopGetFile(**long **lFileHandle);**

- 函数说明：停止下载录像文件。
- 参数说明：

[in]lFileHandle

VideoNet_GetFileByName 的返回值。

- 返回值：成功返回 TRUE，失败返回 FALSE。
- 相关函数：VideoNet_GetFileByName、VideoNet_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。

18. int **VideoNet_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

VideoNet_GetFileByName 的返回值。

- 返回值：成功返回 pos(百分比)。
- 相关函数：VideoNet_GetFileByName、VideoNet_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 云台控制

19. bool **VideoNet_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

  VideoNet_Login 的返回值。

  [in] nChannelNo

  控制的设备通道号。

  [out] lPTZCommand

  控制类型。见 enum 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：VideoNet_Login，VideoNet_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。

20. bool **VideoNet_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

  VideoNet_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。
- ✧ 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。

3.7 系统配置

21. long **VideoNet_GetDevConfig(**long**lLoginID,**unsignedlong**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

VideoNet_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，0-n 代表单个通道。

[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）。

22. long **VideoNet_SetDevConfig(**long**lLoginID,**unsignedlong**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

VideoNet_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，0-n 代表单个通道。

[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）。

23. long **VideoNet_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。

3.8 日志管理

24. bool **VideoNet_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

VideoNet_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。

3.9 远程控制

25. bool **VideoNet_ControlDVR(**long **lLoginID,** int **type,** int **waittime = 2000);**

- 函数说明：设备控制。
- 参数说明：

  [in]lLoginID

  VideoNet_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。

26. long **VideoNet_Upgrade(**long **lLoginID,** char ***sFileName,** int **nType = 0,** fUpgradeCallBack **cbUpgrade = NULL,** long **dwUser = 0);**

- 函数说明：本地升级。
- 参数说明：

  [in]lLoginID

  VideoNet_Login 的返回值。

  [in] sFileName

  升级文件路径。

  [in] nType

  升级类型,1-开始升级 0-终止升级。

  [in] cbUpgrade

  升级进度回调。

  [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：VideoNet_GetUpgradeState， VideoNet_CloseUpgradeHandle

27. int **VideoNet_GetUpgradeState(**long **lUpgradeHandle)**

- 函数说明：获取升级状态。
- 参数说明：

  [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：VideoNet_Upgrade。

28. bool **VideoNet_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：

  [in] szBuf

  接收搜索到的设备信息缓冲。

  [in] nBufLen

  接收搜索到的设备信息缓冲的长度。

  [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。

  [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：true 成功， false 失败。

3.10 语音对讲

29. long **VideoNet_StartVoiceCom_MR (**long **lLoginID,** pfAudioDataCallBack **pVcb,** long **dwDataUser);**

- 函数说明：开启语音对讲。
- 参数说明：

  [in] lLoginID

  设备登陆句柄，VideoNet_Login 返回值。

[in] pVcb

从设备接收到的语音对讲数据回调。

[in] dwDataUser

接收数据对象。

- 返回值：> 0 对讲句柄 <= 0 失败。
- 相关函数：

VideoNet_VoiceComSendData VideoNet_StopVoiceCom VideoNet_SetTalkMode。

30. bool **VideoNet_VoiceComSendData (**long **lVoiceHandle,** char ***pSendBuf,** long **lBufSize);**

- 函数说明：发送对讲数据。
- 参数说明：

[in] lVoiceHandle

对讲句柄，VideoNet_StartVoiceCom_MR 返回值。

[in] pSendBuf

音频数据。

[in] lBufSize

音频数据长度。

- 返回值：true 成功， false 失败。

31. bool **VideoNet_StopVoiceCom (**long **lVoiceHandle);**

- 函数说明：停止语音对讲。
- 参数说明：

[in] lVoiceHandle

对讲句柄，VideoNet_StartVoiceCom_MR 返回值。

- 返回值：true 成功， false 失败。
- 相关函数：

VideoNet_StartVoiceCom_MR VideoNet_VoiceComSendData VideoNet_SetTalkMode

32. bool **VideoNet_SetTalkMode (lon**g **lLoginID,** SDK_AudioInFormatConfig* **pTalkMode);**

- 函数说明：设置对讲音频编码方式，用户可以不设置，默认为 G711A 编码。
- 参数说明：

  [in] lLoginID

  设备登陆句柄，VideoNet_Login 返回值。

  [in] pTalkMode

  对讲模式结构体，SDK_AudioInFormatConfig。
- 返回值：true 成功， false 失败。

3.11 录像模式设置

33. bool **VideoNet_StartDVRRecord(**long **lLoginID,** int **nChannelNo** ,long **lRecordType);**

- 函数说明：设置录像模式。
- 参数说明：

  [in] lLoginID

  设备登陆句柄，VideoNet_Login 返回值。

  [in] nChannelNo

  通道号，-1 所有通道，0-n 代表单个通道。

  [in] lRecordType

  录像类型 参见：SDK_RecordModeTypes 。
- 返回值：true 成功， false 失败。
- 相关函数：

VideoNet_StopDVRRecord VideoNet_GetDevConfig
VideoNet_SetDevConfig

34. bool **VideoNet_StopDVRRecord(**long **lLoginID,** int **nChannelNo);**

- 函数说明：关闭录像。

- 参数说明：

  [in] lLoginID

  设备登陆句柄，VideoNet_Login 返回值。

  [in] nChannelNo

  通道号，-1 所有通道，0-n 代表单个通道。

- 返回值：true 成功， false 失败。

- 相关函数：

VideoNet_StartDVRRecord VideoNet_GetDevConfig VideoNet_SetDevConfig

3.12 设置系统时间

35. bool **VideoNet_SetSystemDateTime (**long **lLoginID,** SDK_SYSTEM_TIME ***pSysTime);**

- 函数说明：设置系统时间。

- 参数说明：

  [in] lLoginID

  设备登陆句柄，VideoNet_Login 返回值。

  [in] pSysTime

  系统时间，参见 SDK_SYSTEM_TIME。

- 返回值：true 成功， false 失败。

3.13 获取设置运行状态信息

36. bool **VideoNet_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：

  [in] lLoginID

  设备登陆句柄，VideoNet_Login 返回值。

  [in] pWorkState

运行状态，参见 SDK_DVR_WORKSTATE。

- 返回值：true 成功， false 失败。

3.14 网络报警

37. bool **VideoNet_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：VideoNet_Login 返回值。

  [in] pAlarmInfo

  网络报警信息，参见 SDK_NetAlarmInfo。

- 返回值：true 成功， false 失败。

3.15 磁盘管理

38. int**VideoNet_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，VideoNet_Login 返回值。

  [in] pStorageCtl

  存储设备控制 参见：SDK_StorageDeviceControl。

- 返回值：=1 成功，<=0 失败。

3.16 抓图

39. bool **VideoNet_CatchPic(**long **lLoginID,** int **nChannel,** char ***sFileName,** int **nType=0);**

- 函数说明：抓图。
- 参数说明：

  [in] lLoginID

  设备登陆句柄，VideoNet_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。

  [in] nType

  图片类型默认为 jpg，（0）jpg（1）YUV。
- 返回值：true 成功， false 失败。

40. bool **VideoNet_CatchPicInBuffer(**long **lLoginID**, int **nChannel**, char ***pBuffer**, int **nBufLen**, int ***pPicLen**, int **nType=0);**

- 函数说明：抓图。
- 参数说明：

  [in] lLoginID

  设备登陆句柄，VideoNet_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。

  [in] nType

  图片类型默认为 jpg，（0）jpg（1）YUV。
- 返回值：true 成功， false 失败。

3.17 支持强迫 I 帧

41. bool **VideoNet_MakeKeyFrame(**long **lLoginID,** int **nChannel,** int **nStream);**

- 函数说明： 强制 I 帧。
- 参数说明：

  [in] lLoginID

  设备登陆句柄，VideoNet_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：true 成功， false 失败

3.18 设置连接设备超时时间和尝试次数

42. bool **VideoNet_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：true 成功， false 失败。

3.19 本地用户操作界面截图

43. bool **VideoNet_CatchPicUI(**long **lLoginID,**char ***saveFileName,**int **type=1);**

- 函数说明：获取本地 ui 图片。
- 参数说明：

  [in] lLoginID

  设备登陆句柄，VideoNet_Login 返回值。

  [in] saveFileName

  图片保存路径。

  [in] type

  保存图片类型,1:为全图片 2:活动区域图片。
- 返回值：true 成功， false 失败。

3.20 绑定本地 ip

44. bool **VideoNet_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：true 成功， false 失败。

3.21 支持设备主动注册

45. bool **VideoNet_StartActiveRigister(**int **nPort,** fMessCallBack **cbFunc,** unsigned long **dwDataUser);**

- 函数说明：开始主动注册。
- 参数说明：

  [in] nPort

  监听端口号。

  [in] cbFunc

  设备注册回调（回调参数，具体参考 demo）。

  [in] dwDataUser

用户数据。

- 返回值：true 成功， false 失败。

46. bool **VideoNet_StopActiveRigister()**

- 函数说明：关闭主动注册。
- 返回值：true 成功， false 失败。

3.22 设置子连接断开回调

47. long **VideoNet_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。

3.23 设置心跳包时间以及断线时间

48. long **VideoNet_SetKeepLifeTime(**long **lLoginID,**unsigned int **perKeeplifeTime,**unsigned int **detectDisconTime);**

- 函数说明：设置保活时间。
- 参数说明：

  [in] lLoginID

  登录句柄。

  [in] perKeeplifeTime

  发送心跳包的间隔时间，默认 10 秒。

  [in] detectDisconTime

断线检测时间，默认 60 秒。

- 返回值：1 成功，0 失败。

3.24 搜索设备局域网设备

49. bool **VideoNet_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，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。

  [in] 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：true 成功， false 失败。

50. bool **VideoNet_SetDVRMessCallBack(**fMessCallBack **cbAlarmcallback,** unsigned long **lUser);**

- 函数说明： 设置接收 DVR 信息回调，例如报警信息,以 ID 号区分。
- 参数说明：

  [in] cbAlarmcallback

  回调函数。

  [in] lUser

  回调函数参数。

- 返回值：true 成功， false 失败。

51. bool **VideoNet_CheckEncrypt(**char ***pMac);**

- 函数说明： MAC 是否得到加密。
- 参数说明：

  [in] pMac

  MAC 地址。
- 返回值：true 加密， false 没有加密。

52. bool **VideoNet_PTZPostion(**long **lLoginID,**int **nChannel,**SDK_PositionPoint ***pPositionPoint);**

- 函数说明：云台快速定位,使用前判断是否支持。
- 参数说明：

  [in] lLoginID

  登陆句柄。

  [in] nChannel

  通道号。

  [in] pPositionPoint

  位置。
- 返回值：true 成功， false 失败。

53. long **VideoNet_RecordInfo(**long **lLoginID,** SDK_SearchByTime* **lpFindInfo,** SDK_RecordStorageInfo ***lpRecordData,** int **waittime = 15000)**

- 函数说明：硬盘录像信息查询。
- 参数说明：

  [in] lLoginID

  登录句柄。

  [in] lpFindInfo

  查询条件。

  [in] lpRecordData

  查找到的录像数据，外部开内存。

[in] waittime

查询超时时间。

- 返回值：1 成功，0 失败。

54. bool **VideoNet_ControlFile(**long **lLoginID,** SDK_NetFileControl ***lpFileControl,**int **waittime = 15000);**

- 函数说明：控制设备上文件。
- 参数说明：

[in] lLoginID

登录句柄。

[in] lpFileControl

需要控制的文件和控制命令。

[in] waittime

查询超时时间。

- 返回值：true 成功，false 失败。

55. long **VideoNet_CmdGeneral(**long **lLoginID,** int **nCmdReq,** const **char *szCmd,** void ***lpOutBuffer,** unsigned long **dwOutBufferSize,** unsigned long*** lpBytesReturned,** int **nTimeout,** char ***pInParam=0,** int **nInParamLen=0,** int **nCmdRes=-1);**

- 函数说明：既可以获取也可以配置设备参数。
- 参数说明：

[in] lLoginID

登录句柄。

[in] nCmdReq

请求命令。

[in] szCmd

json 名称。

[out] lpOutBuffer

接收 json 缓冲。

[in] dwOutBufferSize

接收缓冲大小(字节)。

[out] lpBytesReturned

接收到的 json 大小。

[in] nTimeout

等待时间。

[in] pInParam

配置时使用的 json 字符串。

[in] nInParamLen

配置时 json 大小(字节)。

- 返回值：=0 成功， <0 失败。

56. long **VideoNet_GetDevConfig_Json(**long **lLoginID,** const char*** dwCommand,** int **nChannelNO,**char *** lpOutBuffer,**unsigned long **dwOutBufferSize,**unsigned long*** lpBytesReturned,**int **waittime=1000);**

- 函数说明：获取配置的 json 格式。
- 参数说明：

[in] lLoginID

登录句柄。

[in] dwCommand

json 名。

[in] nChannelNO

通道号,-1:得到全通道配置,0-n:得到单个通道的配置,从 0 开始。

[out] lpOutBuffer

接收 json 缓冲。

[in] dwOutBufferSize

接收缓冲大小(字节)。

[out] lpBytesReturned

接收到的 json 大小。

[in] nTimeout

等待时间。

- 返回值：=1 成功， <0 失败

57. long **VideoNet_SetDevConfig_Json(**long **lLoginID,**const char*** dwCommand,** int **nChannelNO,** char *** lpInBuffer,**int **waittime=1000);**

- 函数说明：设置配置的 json 格式。
- 参数说明：

[in] lLoginID

登录句柄。

[in] dwCommand

json 名。

[in] nChannelNO

通道号,-1:得到全通道配置,0-n:得到单个通道的配置,从 0 开始。

[in] lpInBuffer

设置的 json 字符串。

[in] nTimeout

等待时间。

- 返回值：=1 成功， <0 失败

58. long **VideoNet_GPIOControl(**long **lLoginID,** int **iSecond);**

- 函数说明：GPIO 控制。
- 参数说明：

[in] lLoginID

登录句柄。

[in] iSencond

持续时间。

- 返回值：=0 成功， <0 失败

3.25 人脸图片抓拍、对比

59. unsigned long **VideoNet_StartReceiveFaceImageFile (**long **lLoginID,** int **nChannel);**

- 函数说明：开始抓拍图片传送请求。
- 参数说明：

  [in] lLoginID

  登录句柄。

  [in] nChannel

  通道号。

- 返回值：>0 为正常，=0 异常，使用 GetLastError（）函数获取错误码。

60. unsigned long **VideoNet_StartReceiveFaceImageFileEx(**long **lLoginID,**int **nChannel,** int **enable);**

- 函数说明：开始抓拍图片传送请求（扩展接口）。
- 参数说明：

  [in] lLoginID

  登录句柄。

  [in] nChannel

  通道号。

  [in] enable

  接收图片属性使能 1：接收，0：不接收。

- 返回值：>0 为正常，=0 异常。

61. bool **VideoNet_StopReceiveFaceImageFile(**long **lSubHandle);**

- 函数说明：停止抓拍图片传送请求。
- 参数说明：

  [in] lSubHandle

  抓拍句柄，VideoNet_StartReceiveFaceImageFile 返回值。

- 返回值：>0 为正常，<=0 异常。

62. bool **VideoNet_SetFaceImageDataCallBack (**long **lSubHandle,**fFaceImageDataCallBack **cbFacePictureData,** long **dwUser);**

- 函数说明：抓拍图片回调。
- 参数说明：

  [in] lSubHandle

  抓拍句柄，VideoNet_StartReceiveFaceImageFile 返回值。

  [in] cbFacePictureData

  回调函数指针。

  [in] dwUser

  用户数据。
- 返回值：true 为正常，false 异常

63. long **VideoNet_StartReceiveFaceCompareResult (**long **lLoginID,** int **nChannel);**

- 函数说明：开始人脸图片对比结果请求。
- 参数说明：

  [in] lLoginID

  登录句柄。

  [in] nChannel

  通道号。
- 返回值：>0 为正常，<=0 异常。

64. long **VideoNet_StartReceiveFaceCompareResultEx(**long **lLoginID,**int **nChannel,** SDK_COMPINFO_FLAG ***pstCompinfoFlag);**

- 函数说明：开始人脸图片比对结果请求（扩展接口）。
- 参数说明：

  [in] lLoginID

  登录句柄。

  [in] nChannel

通道号。

[in] pstCompinfoFlag

人脸比对信息使能设置。

- 返回值： >0 为正常，<=0 异常。

65. bool **VideoNet_StopReceiveFaceCompareResult(**long **lSubHandle);**

- 函数说明：停止人脸图片对比结果请求。
- 参数说明：

[in] lSubHandle

比对句柄，VideoNet_StartReceiveFaceCompareResult 返回值。

- 返回值：>0 为正常，<=0 异常。

66. bool **VideoNet_SetFaceCompareResultCallBack (**long **lSubHandle,** fFaceCompaeResultCallBack **cbFacePictureData,** long **dwUser);**

- 函数说明：图片对比回调。
- 参数说明：

[in] lSubHandle

比对句柄，VideoNet_StartReceiveFaceCompareResult 返回值。

[in] cbFacePictureData

回调函数。

[in]dwUser

用户数据。

- 返回值：true 为正常，false 异常

3.26 黑白名单

67. long **VideoNet_StartSendBlackAndWhiteList (**long **lLoginID,** int **nChannel);**

- 函数说明：开始黑白名单请求。
- 参数说明：

[in] lLoginID

登录句柄。

[in] nChannel

通道号。

- 返回值：>0 为正常，<=0 异常。

68. bool **VideoNet_StopSendBlackAndWhiteList (**long **lSubHandle);**

- 函数说明：停止黑白名单请求。
- 参数说明：

[in] lSubHandle

黑白名单操作句柄，VideoNet_StartSendBlackAndWhiteList 返回值。

- 返回值：>0 为正常，<=0 异常。

69. bool **VideoNet_SetBlackAndWhiteListCallBack (**long **lSubHandle,** fFaceBlackAndWhiteListCallBack **cbFacePictureData,** long **dwUser);**

- 函数说明：黑白名单回调。
- 参数说明：

[in] lSubHandle

黑白名单操作句柄，VideoNet_StartSendBlackAndWhiteList 返回值。

[in] cbFacePictureData

回调函数。

[in] dwUser

用户数据。

- 返回值：true 为正常，false 异常。

70. bool **VideoNet_OperationBlackAndWhiteListData (**long **lSubHandle,**WBLIST_ACTION_TYPE **ActionType,**int **nWBType,**int **nlibraryIndex,**int **nFileIndex,**char ***pFile);**

- 函数说明：名单库操作。
- 参数说明：

[in] lSubHandle

黑白名单操作句柄，VideoNet_StartSendBlackAndWhiteList 返回值。

[in] ActionType

黑白名单操作枚举 （参考头文件 WBLIST_ACTION_TYPE）。

[in] nWBType

名单类型，1 黑名单，2 白名单，3 VIP。

[in] nlibraryIndex

库编号。

[in] nFileIndex

图片索引编号。

[in] pFile

传入的文件名。

- 返回值：true 为正常，false 异常。

71. bool **VideoNet_OperationBlackAndWhiteListData_Ex (**long **lSubHandle,** SDK_FA_BLACKANDWHITELIST_CONFIG ***pwebconfig,** char ***parg** = NULL**);**

- 函数说明：黑白名单操作（扩展接口）。
- 参数说明：

[in] lSubHandle

黑白名单操作句柄，VideoNet_StartSendBlackAndWhiteList 返回值。

[in] pwebconfig

SDK_FA_BLACKANDWHITELIST_CONFIG 类型操作结构体。

[in] parg

SDK_WEBLIST_MODIFY_MSG_S 数据结构指针，参考 test.cpp 示例。

- 返回值：true 为正常，false 异常。

72. ulong **VideoNet_StartReceiveFaceMonitor(**long **lLoginID,**int **nChannel,**SDK_FACEMONITOR_INFO_FLAG ***pstPicminfoFlag);**

- 函数说明：新协议打开抓拍比对功能。

- 参数说明：

  [in] lLoginID

  登陆句柄。

  [in] nChannel

  通道号。

  [in] pstPicminfoFlag

  SDK_FACEMONITOR_INFO_FLAG 类型指针。

- 返回值：=0 失败，>0 成功。

73. bool **VideoNet_StopReceiveFaceMonitor(**ulong **lSubHandle);**

- 函数说明：新协议停止抓拍比对。
- 参数说明：

  [in] lSubHandle

  操作句柄。

- 返回值：=true 成功，=false 失败。

74. bool **VideoNet_SetFaceMonitorCallBack(**ulong **lSubHandle,**fFaceMonitorCallBack **cbFacePictureData,** long **dwUser);**

- 函数说明：设置新协议抓拍比对数据回调函数。
- 参数说明：

  [in] lSubHandle

  操作句柄。

  [in] cbFacePictureData

  fFaceMonitorCallBack 类型回调函数。

  [in] dwUser

  用户数据。

- 返回值：=true 成功，=false 失败。

75. ulong **VideoNet_StartSendBlackAndWhiteListV2(**long **lLoginID,**int

nChannel);

- 函数说明：新协议打开黑白名单请求。
- 参数说明：

  [in] lLoginID

  登陆句柄。

  [in] nChannel

  通道号。
- 返回值：=0 失败，>0 成功。

76. bool **VideoNet_StopSendBlackAndWhiteListV2(**long **lSubHandle);**

- 函数说明：新协议停止黑白名单请求。
- 参数说明：

  [in] lSubHandle

  操作句柄。
- 返回值：=true 成功，=false 失败。

77. bool **VideoNet_SetBlackAndWhiteListCallBackV2(**long **lSubHandle,fListManagementCallBack cbFacePictureData,** long **dwUser);**

- 函数说明：新协议设置黑白名单数据回调。
- 参数说明：

  [in] lSubHandle

  操作句柄。

  [in] cbFacePictureData

  fListManagementCallBack 类型回调函数。

  [in] dwUser

  用户数据。
- 返回值：=true 成功，=false 失败。

78. bool **VideoNet_OperationBlackAndWhiteListDataV2(**long **lSubHandle,**

char* **lpBuffer**, ulong **dwBufferSize**, char ***pFile**);

- 函数说明：新协议黑白名单操作。
- 参数说明：

  [in] lSubHandle

  操作句柄。

  [in] * lpBuffer

  SDK_SMART_WB_CONFIG_EX 类型缓存区。

  [in] dwBufferSize

  SDK_SMART_WB_CONFIG_EX 对象长度。

  [in] *pFile

  文件路径。
- 返回值：无。

79. long **VideoNet_SendForHttp_P2P**(char ***puuid**, char ***pDataBuf**, int **iDateLen**);

- 函数说明：基于 HTTP 协议的广域网数据发送。
- 参数说明：

  [in] puuid

  设备 UUID。

  [in] pDataBuf

  发送数据缓存区。

  [in] iDateLen

  发送数据长度。
- 返回值：成功返回发送的字节数，失败返回-1。

80. long **VideoNet_RecvForHttp_P2P**(char ***puuid**, char ****pDataBuf**, int ***iBufSize**,int **timeout**);

- 函数说明：基于 HTTP 协议的广域网数据接收。
- 参数说明：

  [in] puuid

设备 UUID。

[in] pDataBuf

接收数据缓存区。

[in] iDateLen

接收缓存区大小。

[in] timeout

接收超时时间。

- 返回值：成功返回接收的字节数，失败返回-1。

81. int **VideoNet_InitQuerythread();**

- 函数说明：开启查询设备在线状态的线程。
- 参数说明：无。
- 返回值：=0 成功，<0 失败。

82. int **VideoNet_StartQueryUUID(**char ***puuid[],**int **num,**OnFoundDevCB **findDevCB);**

- 函数说明：查询指定 UUID 的设备状态。
- 参数说明：

[in] puuid

存放 UUID 的数组。

[in] num

数组中 UUID 的个数。

[in] OnFoundDevCB

数组中 UUID 的个数。

- 返回值：=0 成功，<0 失败。

83. int **VideoNet_GetP2PLogData();**

- 函数说明：开启 p2p 连接的打印。
- 参数说明：无。

- 返回值：=0 成功，<0 失败。

84. long **VideoNet_Query_Photo_Record(**long **lLoginID,** PICTURE_FINDINFO*** lpFindInfo,** VIDEONET_PICTURE_DATA ***lpFileData,** int **lMaxCount,**int **waittime** = 5000**);**

- 函数说明：通过图片查询图片或者录像。
- 参数说明：

  [in] lLoginID

  登陆句柄。

  [in] lpFindInfo

  PICTURE_FINDINFO 类型指针。

  [in] lpFileData

  VIDEONET_PICTURE_DATA 类型指针。

  [in] lMaxCount

  图片或者录像的数量最大值。

  [in] waittime

  等待时间。

- 返回值：=1 成功，=0 失败。

85. void **VideoNet_SwitchTime(**int **delsecond,**VIDEONET_TIME ***querytime);**

- 函数说明：设置时间偏移。
- 参数说明：

  [in] delsecond

  时间偏移值。

  [in] querytime

  VIDEONET_TIME 类型指针，指定图片的抓拍时间。

- 返回值：无。

3.27 数据回调函数

注意：所有回调函数里面，第三方不应该做复杂的逻辑操作，需要把各自的业务逻辑分开，尤其是不能在回调里面出现阻塞 IO，以免引起线程阻塞，而导致一系列的问题。

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

- 函数说明：断线回调原形。
- 参数说明：

[out] lLoginID

VideoNet_Login 的返回值。

[out] pchDVRIP

设备 IP。

[out] nDVRPort

端口。

[out] dwUser

用户数据。

87. `typedef int(*fRealDataCallBack) (long lRealHandle, long dwDataType, BYTE *pBuffer,long lbufsize,long dwUser);`

- 函数说明：实时监视数据回调原形。
- 参数说明：

[out] lRealHandle

VideoNet_RealPlay 的返回值。

[out] dwDataType

暂时可以不需要判断。

[out] 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。

[out] lbufsize

回调数据的长度，(单位字节)。

[out] dwUser

用户数据。

88. typedef void(***fDownLoadPosCallBack**) (long **lPlayHandle,** long **lTotalSize,** long **lDownLoadSize,** long **dwUser**)

- 函数说明：回放录像进度回调原形。
- 参数说明：

[out] lPlayHandle

VideoNet_PlayBackByName 的返回值。

[out] lTotalSize

指本次播放总大小，单位为 KB。

[out] lDownLoadSize

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。

[out] dwUser

用户数据。

89. typedef bool (***fMessCallBack**)(long **lLoginID,** char ***pBuf,**unsigned long **dwBufLen,** long **dwUser,** int **nType**);

- 函数说明：消息（报警,主动注册）回调原形。
- 参数说明：

[out] lLoginID

VideoNet_Login 的返回值。

[out] pBuf

根据不同的消息类型，对应不同的数据结构，具体请参考 demo。

[out] dwBufLen

接收的数据长度。

[out] dwUser

用户数据。

[out] nType

消息类型。

90. typedef int(***fFaceImageDataCallBack**) (long **lSubFaceHandle,** char ***pBuffer,**long **lbufsize,**SDK_FA_PICTURE_HEAD ***pPictureHead,** SDK_FACE_ATTR* **pPictureAttr,** long **dwUser**);

- 函数说明：人脸抓拍数据回调原形。
- 参数说明：

[out] lSubHandle

抓拍句柄，VideoNet_StartReceiveFaceImageFile 返回值。

[out] pBuffer

人脸图片数据 buffer。

[out] lbufsize

数据 buffer 大小。

[out] pPictureHead

人脸图片信息头。

[out] pPictureAttr

图片属性信息。

[out] dwUser

用户数据。

91. typedef int(***fFaceCompaeResultCallBack**) (long **lSubFaceHandle,** char ***pBuffer,**long **lbufsize,**long **dwUser**);

- 函数说明：人脸比对数据回调原形。
- 参数说明：

[out] lSubHandle

比对句柄，VideoNet_StartReceiveFaceCompareResult 返回值。

[out] pBuffer

人脸比对数据 buffer(返回数据类型参照 SDK_FACECOMPARE_RESULT)。

[out] lbufsize

数据 buffer 大小。

[out] dwUser

用户数据。

92. typedefint(***fFaceBlackAndWhiteListCallBack**)(long**lSubFaceHandle,** char *__pBuffer__,long **lbufsize,**SDK_FA_BLACKANDWHITELIST_CONFIG ***pConfig,**long **dwUser**);

- 函数说明：黑白名单库回调原形。
- 参数说明：

  [out] lSubHandle

  VideoNet_StartSendBlackAndWhiteList 返回值。

  [out] pBuffer

  名单库数据 buffer。

  [out] lbufsize

  数据 buffer 大小。

  [out] pConfig

  名单库图片信息。

  [out] dwUser

  用户数据。

93. typedef void (***OnFoundDevCB**)(char ***** **uuid,**int **state**);

- 函数说明：设备状态回调原形。
- 参数说明：

  [out] *uuid

  设备的 UUID。

  [out] state

  对应 UUID 的设备状态。

94. typedef int(***fFaceMonitorCallBack**) (long **lSubFaceHandle,** char ***pBuf fer,**long **lbufsize,**long **dwUser**);

- 函数说明：抓拍比对数据回调原形。
- 参数说明：

  [out] lSubFaceHandle

操作句柄。

[out] pBuffer

数据缓存。

[out] lbufsize

数据缓存长度。

[out] dwUser

用户数据。

95. typedef int(***fListManagementCallBack**) (long **lSubFaceHandle**, char***pBuffer**,long **lbufsize**, char* **pConfig**, long **nUser**);

- 函数说明：黑白名单操作回调原形。
- 参数说明：

[out] lSubFaceHandle

操作句柄。

[out] pBuffer

数据缓存。

[out] lbufsize

数据缓存长度。

[out] *pConfig

SDK_SMART_WB_CONFIG_EX 类型指针。

[out] dwUser

用户数据。

96. typedef void(***fUpgradeCallBack**) (long **lLoginID**, long **lUpgradechannel**,int **nTotalSize**, int **nSendSize**, long **dwUser**);

- 函数说明：升级进度回调原形。
- 参数说明：

[out] lLoginID

操作句柄。

[out] lUpgradechannel

升级通道号。

[out] nTotalSize

升级文件的总长度，(单位字节)。

[out] nSendSize

已升级的长度，(单位字节)。

[out] dwUser

用户数据。

97. typedef void (*__pfAudioDataCallBack__)(long __lVoiceHandle__, char *__pDataBuf__,long __dwBufSize__, char __byAudioFlag__, long __dwUser__);

- 函数说明：音频数据回调原形。
- 参数说明：

[out] lVoiceHandle

操作句柄。

[out] pDataBuf

音频数据缓存区。

[out] dwBufSize

音频数据长度。

[out] byAudioFlag

音频数据标志。

[out] dwUser

用户数据。

98. typedef void (__*fSubDisConnectCallBack__)(long __lLoginID__, SubConnType __type__, long __nChannel__, long __dwUser__);

- 函数说明：子连接断线回调原形。
- 参数说明：

[out] lLoginID

操作句柄。

[out] type

SubConnType 枚举类型。

[out] nChannel

通道号。

[out] dwUser

用户数据。

4 示例功能实现

请参看 Test.cpp 程序和”read.txt”