

Net**SDK** Programming Manual

VERSION 1.4.0.0 (Build 090926)

2009-01-10

All rights reserved

Foreword

Thank you for using our VANGUARD devices. We are going to provide best service for you. This manual may contain spelling grammar and punctuation errors. We will update this manual regularly.

Modification History

Date	Content
2008.10.22	Create
2009.02.12	Add H264_DVR_SearchDevice for search device at LAN
2009.02.18	Add API with talk: H264_DVR_StartVoiceCom_MR , H264_DVR_VoiceComSendData , H264_DVR_StopVoiceCom , H264_DVR_SetTalkMode
2009.09.26	Add API below: H264_DVR_StartDVRRecord , H264_DVR_StopDVRRecord , H264_DVR_SetSystemDateTime , H264_DVR_GetDVRWorkState, H264_DVR_ClickKey

Catalogue

1. GENERAL INTRODUCTION	5
1.1 INTRODUCTION	5
1.2 APPLICABILITY	5
DESIG OF PRINCIPIA	6
1.3 PROGRAMMING DESCRIPTION	6
1.4 API CALLING REFERENCE	7
2 DATA STRUCTURE DESCRIPTION	9
2.1 CLIENT DATA STRUCTURE DESCRIPTION	9
2.1.1 Structure of device information	12
2.1.2 Date Information	12
2.1.3 Record File Information	12
2.1.4 Structure of Configuration	16
2.1.5 Network keyboard define	28
3 API DEFINITION.....	31
3.1 SDK INITIALIZATION.....	31
3.2 GET ALARM STATUS	32
3.3 DEVICE REGISTRATION.....	34
3.4 REAL-TIME MONITOR.....	35
3.5 PLAYBACK AND DOWNLOAD	37
3.6 PLAYBACK CONTROL	40
3.7 PTZ CONTROL.....	40
3.8 SYSTEM CONFIGURATION.....	41
3.9 LOG MANAGEMENT	42
3.10 REMOTE CONTROL	43
3.11 UPGRADE	43
3.11 AUDIO TALK	45
3.12 RECORD MODE.....	47
3.13 SET SYSTEM TIME	47
3.14 GET DEVICE STATUS.....	48
3.15 NET KEYBOARD.....	48

1. General Introduction

1.1 Introduction

This SDK is for VANGUARD DVR, network video server and etc. The text describes functions and interfaces as well as relations among them. It also provide detailed demonstration.

This package includes two parts: operation and equipment management:

Operation

state listening, real-time monitor, real-time preview, character overlay, audio control, record playback and download, data storage, PTZ control, alarm deployment, voice dialogue, log management, user management, action on alarm, transparent com port and etc.

Device management

remote upgrade, remote restart/shutdown, equipment parameters setup such as general setting, alarm, record, serial port, image, log, user management, device timing, motion detection, network and etc.

The development package includes the following files:

Network library	NetSDK	Head file
	NetSDK.lib	Lib file
	NetSDK.dll	Interface library
Assitante library	H264Play.dll	Decode assistant library
	DllDeinterlace.dll	Decode assistant library
	hi_h264dec_w.dll	Decode assistant library

1.2 Applicability

- Support real-time monitor playback, Alarm, Remote config, log management and etc.
- Support TCP network transmisson, Max 10 TCP connections
- You can use this SDK to connect to device to develop client-end program, or you can use SDK callback interface to develop server program such as stream media transfer, playback, alarm and etc.
- The client supports multi- image resolutions including QCIF、CIF、2CIF、

HalfD1、D1, VGA (640×480) and etc.

- When SDK is playback/download, one logged ID can not operate playback and download for the same channel at the same time.
- SDK performance has relationship with device running status and client-end PC capability. Basically, it can support 1024 users to register at the same time, 1024-ch network preview and playback at the same time. Support 1024-ch alarm upload at the same time .And 100-ch video display

Desig of principia

1.3 Programming Description

■ *Initialization and clean up*

- 1、First calling `H264_DVR_Init()` to initialize SDK, when Application exit, calling `H264_DVR_Cleanup()` to release all occupied resource.
- 2、Most API shall call **after** `H264_DVR_Init()`, **before** `H264_DVR_Cleanup()` , But `H264_DVR_GetLastError` can be called anywhere.

■ *Login in and login out*

Before access device, you shall call `H264_DVR_Login()` **login first, also you can call `H264_DVR_LoginEx()`** to assign your client type logged. If login succeeded, will return a global handle, the handle is a session channel, you can use it to operate device. Call `H264_DVR_Logout()` **to close this session.**

■ *Callback Fuction*

Callback function have a parameter: `dwUser`, it is elf-define parameter, you can define your owner data.

1.4 API Calling Reference

The below diagram describes a brief API calling reference for basic client application, and users can add in other function modules according to actual application requirements.

A. Initialization

SDK Initialization	<i>H264_DVR_Init</i> ()
--------------------	-------------------------

B. Set callback for alarm message

Set callback	<i>H264_DVR_SetDVRMessCallBack</i> ()
--------------	---------------------------------------

C. Login in

Login in	<i>H264_DVR_Login</i> ()
	<i>H264_DVR_LoginEx</i> ()
Subscribe for alarm message	<i>H264_DVR_SetupAlarmChan</i> ()

D. Operation of device ,get and set parameter configuration

Parameter configuration	<i>H264_DVR_GetDevConfig</i> ()
	<i>H264_DVR_SetDevConfig</i> ()
Log searching	<i>H264_DVR_FindDVRLog</i> ()
PTZ control	<i>H264_DVR_PTZControl</i> ()
	<i>H264_DVR_PTZControlEx</i> ()

E. Real-time preview

Open and close monitor channel	<i>H264_DVR_RealPlay</i> ()
	<i>H264_DVR_StopRealPlay</i> ()
Callback for saving monitor data	<i>H264_DVR_SetRealDataCallBack</i> ()

F. Playback/Download

Record searching	<i>H264_DVR_FindFile</i> ()
Playback control	<i>H264_DVR_PlayBackByName</i> ()
	<i>H264_DVR_PlayBackControl</i> ()
	<i>H264_DVR_StopPlayBack</i> ()
Download	<i>H264_DVR_GetFileByName</i> ()

H264_DVR_GetDownloadPos() H264_DVR_StopGetFile ()
--

G. Remote control

Upgrade	H264_DVR_Upgrade() H264_DVR_GetUpgradeState() H264_DVR_CloseUpgradeHandle()
Reboot /Clear log	H264_DVR_ControlDVR ()

H. Login out

Cancel subscribe for alarm message	H264_DVR_CloseAlarmChan ()
Disconnect	H264_DVR_Logout ()

I. Release SDK resource

SDK exit	H264_DVR_Cleanup ()
----------	---------------------

2 Data Structure Description

2.1 Client Data Structure Description

```
//PTZ control type
typedef enum PTZ_ControlType
{
    TILT_UP = 0,           //UP
    TILT_DOWN,           //DOWN
    PAN_LEFT,            //LEFT
    PAN_RIGTH,           //RIGTH
    PAN_LEFTTOP,         //LEFT TOP
    PAN_LEFTDOWN,        //LEFT DOWN
    PAN_RIGHTTOP,        //RIGTH TOP
    PAN_RIGTHDOWN,       //RIGTH DOWN
    ZOOM_IN,             //ZOOM IN
    ZOOM_OUT,            //ZOOM OUT
    FOCUS_FAR,           //FOCUS FAR
    FOCUS_NEAR,          //FOCUS NEAR
    IRIS_OPEN,           //IRIS OPEN
    IRIS_CLOSE,          //IRIS CLOSE

    EXTPTZ_OPERATION_ALARM, //ALARM
    EXTPTZ_LAMP_ON,         //LIGTH OPEN
    EXTPTZ_LAMP_OFF,        //LIGTH CLOSE
    EXTPTZ_POINT_SET_CONTROL, //SET PRESET POINT
    EXTPTZ_POINT_DEL_CONTROL, //CLEAR PRESET POINT
    EXTPTZ_POINT_MOVE_CONTROL, //GOTO PRESET POINT
    EXTPTZ_STARTPANCRUISE,  //START PAN CRUISE
    EXTPTZ_STOPPANCRUISE,   //STOP PAN CRUISE
    EXTPTZ_SETLEFTBORDER,   //SET LEFT BORDER
    EXTPTZ_SETRIGHTBORDER,  //SET RIGHT BORDER
    EXTPTZ_STARTLINESCAN,   //START AUTO SCAN
    EXTPTZ_CLOSELINESCAN,   //STOP AUTO SCAN
    EXTPTZ_ADDTOLOOP,        //ADD PRESET POINT TO CRUISE LINE
    EXTPTZ_DELFROMLOOP,      //DEL PRESET POINT FROM CRUISE LINE
    EXTPTZ_POINT_LOOP_CONTROL, //START CRUISE
    EXTPTZ_POINT_STOP_LOOP_CONTROL, //STOP CRUISE
    EXTPTZ_CLOSELOOP,        //CLEAR CRUISE LINE
    EXTPTZ_FASTGOTO,         //FAST GOTO

```

```

EXTPTZ_AUXIOPEN,          //AUX OPEN
EXTPTZ_OPERATION_MENU,   //OPERATION MENU
EXTPTZ_REVERSECOMM,     //REVER CAMERAL
EXTPTZ_OPERATION_RESET,  ///< PTZ RESET
EXTPTZ_TOTAL,
};

```

Error code, return by GetLastError

```

typedef enum SDK_RET_CODE
{
    H264_DVR_NOERROR          = 0,          //no error
    H264_DVR_SUCCESS         = 1,          //success
    H264_DVR_SDK_NOTVALID    = -10000,     //invalid request
    H264_DVR_NO_INIT        = -10001,     //SDK not initied
    H264_DVR_ILLEGAL_PARAM   = -10002,     //illegal user parameter
    H264_DVR_INVALID_HANDLE  = -10003,     //handle is null
    H264_DVR_SDK_UNINIT_ERROR = -10004,     //SDK clear error
    H264_DVR_SDK_TIMEOUT     = -10005,     //timeout
    H264_DVR_SDK_MEMORY_ERROR = -10006,     //memory error
    H264_DVR_SDK_NET_ERROR   = -10007,     //network error
    H264_DVR_SDK_OPEN_FILE_ERROR = -10008, //open file fail
    H264_DVR_SDK_UNKNOWNERROR = -10009,     //unknown error
    H264_DVR_DEV_VER_NOMATCH = -11000,     //version mismatch
    H264_DVR_ERROR_GET_DATA  = -11001,     //get data fail (including
configure, user information and etc)

    H264_DVR_OPEN_CHANNEL_ERROR = -11200, //open channel fail
    H264_DVR_CLOSE_CHANNEL_ERROR = -11201, //close channel fail
    H264_DVR_SUB_CONNECT_ERROR = -11202, //open media connet fail
    H264_DVR_SUB_CONNECT_SEND_ERROR = -11203, //media connet send data
fail

    /// error code of user management
    H264_DVR_NOPOWER          = -11300, //no power
    H264_DVR_PASSWORD_NOT_VALID = -11301, // password not valid
    H264_DVR_LOGIN_USER_NOEXIST = -11302, // user not exist
    H264_DVR_USER_LOCKED      = -11303, // user is locked
    H264_DVR_USER_IN_BLACKLIST = -11304, // user is in blacklist
    H264_DVR_USER_HAS_USED     = -11305, // user have loggedin
    H264_DVR_USER_NOT_LOGIN    = -11305, // no login
    H264_DVR_CONNECT_DEVICE_ERROR = -11306, // maybe device no exist

```

```

    /// error code of configure management
    H264_DVR_OPT_RESTART          = -11400, // need to restart
application  H264_DVR_OPT_REBOOT      = -11401, // need to reboot
system H264_DVR_OPT_FILE_ERROR      = -11402, // write file fail
    H264_DVR_OPT_CAPS_ERROR       = -11403, // not support
    H264_DVR_OPT_VALIDATE_ERROR   = -11404, // validate fail
    H264_DVR_OPT_CONFIG_NOT_EXIST = -11405, // config not exist
    H264_DVR_CTRL_PAUSE_ERROR     = -11500, // pause fail
};

```

Alarm Event Type

enum SDK_EventCodeTypes

```

{
    SDK_EVENT_CODE_INIT = 0,
    SDK_EVENT_CODE_LOCAL_ALARM = 1, // local alarm
    SDK_EVENT_CODE_NET_ALARM,      // network alarm
    SDK_EVENT_CODE_MANUAL_ALARM,   // manual alarm
    SDK_EVENT_CODE_VIDEO_MOTION,   // motion detect
    SDK_EVENT_CODE_VIDEO_LOSS,     // loss detect
    SDK_EVENT_CODE_VIDEO_BLIND,    // blind detect
    SDK_EVENT_CODE_VIDEO_TITLE,
    SDK_EVENT_CODE_VIDEO_SPLIT,
    SDK_EVENT_CODE_VIDEO_TOUR,
    SDK_EVENT_CODE_STORAGE_NOT_EXIST,
    SDK_EVENT_CODE_STORAGE_FAILURE,
    SDK_EVENT_CODE_LOW_SPACE,
    SDK_EVENT_CODE_NET_ABORT,
    SDK_EVENT_CODE_COMM,
    SDK_EVENT_CODE_STORAGE_READ_ERROR,
    SDK_EVENT_CODE_STORAGE_WRITE_ERROR,
    SDK_EVENT_CODE_NET_IPCONFLICT,
    SDK_EVENT_CODE_ALARM_EMERGENCY,
    SDK_EVENT_CODE_DEC_CONNECT,
    SDK_EVENT_CODE_NR,
};
//alarm information
typedef struct SDK_ALARM_INFO
{
    int nChannel;
    int iEvent; ///< refer to SDK_EventCodeTypes
    int iStatus; ///< 0: start 1: stop

```

```

    SDK_SYSTEM_TIME SysTime;
}SDK_AlarmInfo;

```

2.1.1 Structure of device information

 Structure of device define as below

```

typedef struct _H264_DVR_DEVICEINFO
{
    char sSoftWareVersion[64]; ///< software version
    char sHardWareVersion[64]; ///< hardware version
    char sEncryptVersion[64];  ///< encrypt version
    SDK_SYSTEM_TIME tmBuildTime;///< build time
    char sSerialNumber[64];    ///< device serial number
    int byChanNum;             ///< channel number of video in
    int iVideoOutChannel;      ///< channel number of video out
    int byAlarmInPortNum;      ///< channel number of alarm in
    int byAlarmOutPortNum;     ///< channel number of alarm out
    int iTalkInChannel;        ///< channel number of talk in
    int iTalkOutChannel;       ///< channel number of talk out
    int iExtraChannel;         ///< channel number of extra
    int iAudioInChannel;       ///< channel number of audio in
    int iCombineSwitch;        ///< channel number of combine
}H264_DVR_DEVICEINFO,*LPH264_DVR_DEVICEINFO;

```

2.1.2 Date Information

```

typedef struct SDK_SYSTEM_TIME{
    int year;///< year
    int month;///< month, January = 1, February = 2, and so on.
    int day;///< day
    int wday;///< week, Sunday = 0, Monday = 1, and so on
    int hour;///< hour
    int minute;///< minute
    int second;///< second
    int isdst;///< DST(daylight saving time) flag, Yes = 1, No = 0
}SDK_SYSTEM_TIME;

```

2.1.3 Record File Information

```

//search condition structure

typedef struct
{
    int nChannelNO;           ///

```

```

    int nFileType;           //record type
    H264_DVR_TIME startTime; //start time
    H264_DVR_TIME endTime;  //end time
    char szCard[32];        //card number
}H264_DVR_FINDINFO;

//the return of record information structure
typedef struct
{
    int ch;                  ///< channel NO, start with 0
    int size;                ///< record size(BYTE)
    char sFileName[108];    ///< record file name
    SDK_SYSTEM_TIME stBeginTime; ///< start time of record
    SDK_SYSTEM_TIME stEndTime;  ///< end time of record
}H264_DVR_FILE_DATA;

```

Protocol Of Serial Information

```

struct SDK_COMMATTRI
{
    int iDataBits;    // data bit: [5,8]
    int iStopBits;   // stop bit: [0,2]
    int iParity;     // parity: 0: None 1: odd 2: even 3: mark 4: space
    int iBaudRate;   // baudrate: 115200,57600,38400,9600,4800,2400 and so
on
};
// serial configure
struct SDK_CONFIG_COMM_X
{
    char iProtocolName[32]; // Protocol: "Console"
    int iPortNo;           // Port No.
    SDK_COMMATTRI aCommAttri; // attribute of serial
};

```

云台协议

```

struct SDK_STR_CONFIG_PTZ
{
    char sProtocolName[NET_MAX_PTZ_PROTOCOL_LENGTH]; // Protocol
    int ideviceNo; // PTZ device NO.
}

```

```

    int iNumberInMatrixs;        // NO. in matrixs
    int iPortNo;                // serial port NO.: [1, 4]
    SDK_COMMATTRI dstComm;      // attribute of serial
};

//all channel of PTZ protocol
struct SDK_STR_PTZCONFIG_ALL
{
    SDK_STR_CONFIG_PTZ ptzAll[NET_MAX_CHANNUM];
};

```

User Manager Data Structure

Right List

```

typedef struct _OPR_RIGHT
{
    string    name;
}OPR_RIGHT;

typedef struct _USER_INFO
{
    int       righthNum;
    string    rights[NET_MAX_RIGTH_NUM];
    string    strGroupname;
    string    strmemo;
    string    strname;
    string    strpassWord;
    bool      reserved;    //is reserved user
    bool      shareable;   //is shareable
}USER_INFO;

typedef struct _USER_GROUP_INFO
{
    int       righthNum;
    string    memo;
    string    name;
    string    rights[NET_MAX_RIGTH_NUM]; //right list
}USER_GROUP_INFO;

//all of user and group information structure
typedef struct _USER_MANAGE_INFO
{
    int       righthNum;
    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;

```

```

//modify user
typedef struct _CONF_MODIFYUSER
{
    std::string sUserName;
    USER_INFO User;
}CONF_MODIFYUSER;

```

```

//modify group
typedef struct _CONF_MODIFYGROUP
{
    std::string sGroupName;
    USER_GROUP_INFO Group;
}CONF_MODIFYGROUP;

```

```

/// modify password
struct _CONF_MODIFY_PSW
{
    std::string sUserName;
    std::string sPassword;
    std::string sNewPassword;
};

```

Log Information

```

#define NET_MAX_RETURNED_LOGLIST 1024    //the max item of Log
/// log search condition
struct SDK_LogSearchCondition
{
    int nType;    ///< log type: 0: all 1: system 2: configure 3: storage 4:
alarm 5: record 6: account 7: file
    int iLogPosition;    ///< return of last log item position in the
whole logs
    SDK_SYSTEM_TIME stBeginTime;    ///< begin time
    SDK_SYSTEM_TIME stEndTime;    ///< end time
};
//return of Log search

```

```

struct SDK_LogList
{
    int iNumLog;
    struct LogItem
    {
        char sType[24]; ///< log type
        char sUser[32]; ///< Operator of log
        char sData[68]; ///< log data
        SDK_SYSTEM_TIME stLogTime; ///< the time of log happened
    } Logs[NET_MAX_RETURNED_LOGLIST];
};

```

Storage information structure

```

struct SDK_STORAGEDISK
{
    int iPhysicalNo;
    int iPartNumber;
    SDK_DriverInformation diPartitions[SDK_MAX_DRIVER_PER_DISK];
};

struct SDK_StorageDeviceInformationAll
{
    int iDiskNumber;
    SDK_STORAGEDISK vStorageDeviceInfoAll[SDK_MAX_DISK_PER_MACHINE];
};

```

Real-Time Monitor

```

typedef struct{
    int nChannel; //Channel NO.
    int nStream; //0: main stream, 1: extra stream
    int nMode; //0: TCP, 1: UDP
}H264_DVR_CLIENTINFO, *LPH264_DVR_CLIENTINFO;

```

2.1.4 Structure of Configuration

Commands of H264_DVR_GetDevConfig、H264_DVR_SetDevConfig

```

typedef enum _SDK_CONFIG_TYPE
{
    E_SDK_CONFIG_NOTHING = 0,
    //User Managerment
    E_SDK_CONFIG_USER, //User information, including power list, user
                        //list and group list
};

```



```

                                USER_MANAGE_INFO
E_SDK_CONFIG_ADD_USER,          //add user USER_INFO
E_SDK_CONFIG_MODIFY_USER,       //modify user CONF_MODIFYUSER
E_SDK_CONFIG_DELETE_USER,       //del user USER_INFO
E_SDK_CONFIG_ADD_GROUP,         //add group USER_GROUP_INFO
E_SDK_CONFIG_MODIFY_GROUP,      //modify group CONF_MODIFYGROUP
E_SDK_CONFIG_DELETE_GROUP,      //del group USER_GROUP_INFO
E_SDK_CONFIG_MODIFY_PSW,        //modify password CONF_MODIFY_PSW

//device ability
E_SDK_CONFIG_ABILITY_SYSFUNC = 9, //support network services

SDK SystemFuncio
E_SDK_CONFIG_ABILITY_ENCODE,     //encode ability CONFIG EncodeAbility
E_SDK_CONFIG_ABILITY_PTZPRO,     //protocols of ptz support

SDK PTZPROTOCOLFUNC
E_SDK_CONFIG_ABILITY_COMMPRO,    // protocols of 232 support

SDK COMMFUNC
E_SDK_CONFIG_ABILITY_MOTION_FUNC, //Motion detect SDK MotionDetectFunction
E_SDK_CONFIG_ABILITY_BLIND_FUNC, //Blind detect SDK BlindDetectFunction
E_SDK_CONFIG_ABILITY_DDNS_SERVER, // type of DDNS services support

SDK DDNSServiceFunction
E_SDK_CONFIG_ABILITY_TALK,       // encode type of audio talk support

//Device configuration
E_SDK_CONFIG_SYSINFO = 17,       //system information H264 DVR_DEVICEINFO
E_SDK_CONFIG_SYSNORMAL,          //general SDK CONFIG NORMAL
E_SDK_CONFIG_SYSENCODE,          //encode SDK EncodeConfigAll
E_SDK_CONFIG_SYSNET,             //network SDK CONFIG NET COMMON
E_SDK_CONFIG_PTZ,                //ptz SDK STR PTZCONFIG ALL
E_SDK_CONFIG_COMM,               //232 SDK CommConfigAll
E_SDK_CONFIG_RECORD,            //record SDK RECORDCONFIG ALL
E_SDK_CONFIG_MOTION,             //motion detect SDK MOTIONCONFIG
E_SDK_CONFIG_SHELTER,            //blind detect SDK BLINDETECTCONFIG ALL
E_SDK_CONFIG_VIDEO_LOSS,         //loss detect SDK VIDEOLOSSCONFIG ALL
E_SDK_CONFIG_ALARM_IN,          //alarm in SDK ALARM INPUTCONFIG ALL
E_SDK_CONFIG_ALARM_OUT,         //alarm out
E_SDK_CONFIG_DISK_MANAGER        //disk management
E_SDK_CONFIG_OUT_MODE,           //out mode
E_SDK_CONFIG_AUTO,               //auto maintain SDK AutoMaintainConfig
E_SDK_CONFIG_DEFAULT,           //set default
E_SDK_CONFIG_DISK_INFO,         //disk info SDK StorageDeviceInformationAll
E_SDK_CONFIG_LOG_INFO,          //get log SDK_LogList

```

```

E_SDK_CONFIG_NET_IPFILTER,    //network services: black/white list
    SDK\_NetIPFilterConfig
E_SDK_CONFIG_NET_DHCP,       //network services: DHCP
E_SDK_CONFIG_NET_DDNS,       //network services: DDNS
    SDK\_NetDDNSConfigALL
E_SDK_CONFIG_NET_EMAIL,      //network services: EMAIL           SDK\_NetEmailConfig
E_SDK_CONFIG_NET_MULTICAST,  //network services:Multicast
    SDK\_NetMultiCastConfig
E_SDK_CONFIG_NET_NTP,        //network services: NTP           SDK\_NetNTPConfig
E_SDK_CONFIG_NET_PPPOE,     //network services: PPPOE         SDK\_NetPPPoEConfig
E_SDK_CONFIG_NET_DNS,       //network services: DNS           SDK\_NetDNSConfig
E_SDK_CONFIG_NET_FTPSERVER, //network services: FTP
    SDK\_FtpServerConfig
E_SDK_CONFIG_SYS_TIME,      //system time
E_SDK_CONFIG_CLEAR_LOG,    //clear log
E_SDK_REBOOT_DEV,          //reboot device

E_SDK_CONFIG_ABILITY_LANG,  //languages support
E_SDK_CONFIG_VIDEO_FORMAT,  //Video format
E_SDK_CONFIG_COMBINEENCODE, //combine-encode
E_SDK_CONFIG_EXPORT,        //config export
E_SDK_CONFIG_IMPORT,        //config import
E_SDK_LOG_EXPORT,           //log export
E_SDK_CONFIG_COMBINEENCODEMODE, //mode of combine-encode
E_SDK_WORK_STATE,          //work status
}SDK_CONFIG_TYPE;

/// type of DDNS support
struct SDK_DDNSServiceFunction
{
    int nTypeNum;
    char vDDNSType[NET_MAX_DDNS_TYPE][64];
};

/// blind detect support
struct SDK_BlindDetectFunction
{
    int iBlindCoverNum;    ///< the number of cover area support
};

/// motion detect
struct SDK_MotionDetectFunction
{
    int iGridRow;    ///< the number of row

```

```
    int iGridColumn; ///< the number of column
};
/// protocols of 232 support
struct SDK_COMMFUNC
{
    int nProNum;      ///< the numbers of protocol
    char vCommProtocol[SDK_COM_TYPES][32]; ///< the name of protocol
};
/// protocols of PTZ
struct SDK_PTZPROTOCOLFUNC
{
    int nProNum;
    char vPTZProtocol[100][NET_MAX_PTZ_PROTOCOL_LENGTH];
};
/// encode information
struct SDK_EncodeInfo
{
    bool bEnable;      ///< enable
    int iStreamType;   ///< stream type see refer to capture_channel_t
    bool bHaveAudio;   ///< is support audio
    unsigned int uiCompression;    ///< mask of capture_comp_t
    unsigned int uiResolution;     ///< mask of capture_size_t
};
/// encode power
struct CONFIG_EncodeAbility
{
    int iMaxEncodePower;      ///< max encode power
    SDK_EncodeInfo vEncodeInfo[SDK_CHL_FUNCTION_NUM]; ///< encode information
    SDK_EncodeInfo vCombEncInfo[SDK_CHL_FUNCTION_NUM]; ///< combine-encode information
};

///system function
struct SDK_SystemFunction
{
    bool vEncodeFunction[SDK_ENCODE_FUNCTION_TYPE_NR]; ///< Encode Functions
    bool vAlarmFunction[SDK_ALARM_FUNCTION_TYPE_NR];  ///< Alarm Fucntions
    bool vNetServerFunction[SDK_NET_SERVER_TYPES_NR]; ///< Net Server Functions
    bool vPreviewFunction[SDK_PREVIEW_TYPES_NR];      ///< Preview Functions
};

///< Auto-Maintain setting
struct SDK_AutoMaintainConfig
{
```

```
    int iAutoRebootDay;           ///< interval of Auto-Reboot days
    int iAutoRebootHour;         ///< time to reboot [0, 23]
    int iAutoDeleteFilesDays;    ///< interval of Auto-Del record file [0, 30]
};
//Disk info
struct SDK_STORAGEDISK
{
    int iPhysicalNo;             // Physical No.
    int iPartNumber;            // Partition numbers
    SDK_DriverInformation diPartitions[SDK_MAX_DRIVER_PER_DISK];
};
struct SDK_StorageDeviceInformationAll
{
    int iDiskNumber;
    SDK_STORAGEDISK vStorageDeviceInfoAll[SDK_MAX_DISK_PER_MACHINE];
};

// Type of PTZ link
enum PtzLinkTypes
{
    PTZ_LINK_NONE,              // NONE
    PTZ_LINK_PRESET,           // GOTO PRESET
    PTZ_LINK_TOUR,              // TOUR
    PTZ_LINK_PATTERN           // PATTERN
};

// PTZ Link Config
struct SDK_PtzLinkConfig
{
    int iType;                  // see refer to PtzLinkTypes
    int iValue;                 // value of link type
};

// handler of event
struct SDK_EventHandler
{
    unsigned int dwRecord;      // bitmask of record. Bit per channel
    unsigned int iRecordLatch; // record latch: 10~300 sec.
    unsigned int dwTour;        // bitmask of tour. Bit per channel
    unsigned int dwSnapshot;    // bitmask of snapshot. Bit per channel
    unsigned int dwAlarmOut;    // bitmask of alarm out. Bit per channel
    unsigned int dwMatrix;      // bitmask of matrix. Bit per channel
    int iEventLatch;           // interval of event(unit:sec.)
};
```

```
int      iAOLatch;           // Alarm out latch: 10~300 sec
SDK_PtzLinkConfig PtzLink[NET_MAX_CHANNUM]; // PTZ link activation
SDK_CONFIG_WORKSHEET schedule; // weeksheet of record
bool    bRecordEn;         // enable flag of record
bool    bTourEn;          // enable flag of tour
bool    bSnapEn;          // enable flag of snapshot
bool    bAlarmOutEn;      // enable flag of alarm out
bool    bPtzEn;           // enable flag of PTZ link
bool    bTip;             // enable flag of screen tip
bool    bMail;            // enable flag of sending email
bool    bMessage;        // enable flag of sending message to alarm center
bool    bBeep;           // enable flag of buzzer beep
bool    bVoice;          // enable flag of voice tip
bool    bFTP;            // enable flag of FTP unload
bool    bMatrixEn;       // no used
bool    bLog;            // enable flag of log
bool    bMessagetoNet;   // no used
};

///< Blind detect
struct SDK_BLINDETECTCONFIG
{
    bool bEnable;           ///< enable
    int  iLevel;           ///< sensitivity: 1~6
    SDK_EventHandler hEvent; ///< handler of blind detect event
};

///< All channel of blind detect configuration
struct SDK_BLINDETECTCONFIG_ALL
{
    SDK_BLINDETECTCONFIG vBlindDetectAll[NET_MAX_CHANNUM];
};

///< Alarm in
struct SDK_ALARM_INPUTCONFIG
{
    bool bEnable;           ///< enable
    int  iSensorType;      ///< Sensor Type: Normal Open or Normal Close
    SDK_EventHandler hEvent; ///< handler of alarm in
};

///< All channel of alarm in configuration
struct SDK_ALARM_INPUTCONFIG_ALL
{
```

```
    SDK_ALARM_INPUTCONFIG vAlarmConfigAll[NET_MAX_CHANNUM];
};

///< Motion detect
struct SDK_MOTIONCONFIG
{
    bool bEnable;           ///< enable
    int iLevel;            ///< sensitivity: [1,6]
    unsigned int mRegion[NET_MD_REGION_ROW];  ///< regions of motion detect, one bit per column, Max
    region: 18*22
    SDK_EventHandler hEvent;  ///< handler of motion detect
};

/// All channel of video motion configuration
struct SDK_MOTIONCONFIG_ALL
{
    SDK_MOTIONCONFIG vMotionDetectAll[NET_MAX_CHANNUM];
};

///< video loss detect
struct SDK_VIDEOLOSSCONFIG
{
    bool bEnable;          ///< enable
    SDK_EventHandler hEvent;  ///< event handler
};

/// All channel of video loss configuration
struct SDK_VIDEOLOSSCONFIG_ALL
{
    SDK_VIDEOLOSSCONFIG vGenericEventConfig[NET_MAX_CHANNUM];
};

/// record mode type
enum SDK_RecordModeTypes
{
    SDK_RECORD_MODE_CLOSED,    ///< Closed
    SDK_RECORD_MODE_MANUAL,    ///< Manual: record all the time
    SDK_RECORD_MODE_CONFIG,    ///< Configuration: according to SDK_RECORDCONFIG
    SDK_RECORD_MODE_NR,
};

///< record setting
struct SDK_RECORDCONFIG
{
```

```

int iPreRecord;           ///< pre-record time (Unit:sec.)
bool bRedundancy;        ///< redundancy record
bool bSnapShot;          ///< no used
int iPacketLength;       ///< record length (unit:minute) [1, 255]
int iRecordMode;         ///< record mode, refer to SDK\_RecordModeTypes
SDK_CONFIG_WORKSHEET wcWorkSheet;           ///< worksheet
unsigned int typeMask[NET_N_WEEKS][NET_N_TSECT];  ///< mask of record type, corresponding to
worksheet
};

// All channel of record configuration
struct SDK_RECORDCONFIG_ALL
{
    SDK_RECORDCONFIG vRecordConfigAll[NET_MAX_CHANNUM];
};

// General Configuration
typedef struct _SDK_CONFIG_NORMAL
{
    NEW_NET_TIME sysTime;           ///< system time
    int iLocalNo;                   ///< device No.:[0, 998]
    int iOverWrite;                 ///< when disk full, 0: OverWrite, 1: StopRecord
    int iSnapInterval;              ///< no used
    char sMachineName[64];          ///< device name
    int iVideoStartOutPut;          ///< no used
    int iAutoLogout;                ///< auto logout [0, 120], 0 means never
    int iVideoFormat;               ///< video format: 0:PAL, 1:NTSC, 2:SECAM
    int iLanguage;                  ///< language: 0:English, 1: SimpChinese, 2:TradChinese, 3: Italian,
4:Spanish, 5:Japanese, 6:Russian, 7:French, 8:German
    int iDateFormat;               ///< date format: 0:YYMMDD, 1:MMDDYY, 2:DDMMYY
    int iDateSeparator;            ///< Date separator: 0: ., 1: -, 2: /
    int iTimeFormat;               ///< Time format: 0: 12, 1: 24
    int iDSTRule;                  ///< DST rule: 0: OFF, 1: ON
    int iWorkDay;                  ///< work day
    DSTPoint dDSTStart;
    DSTPoint dDSTEnd;
}SDK_CONFIG_NORMAL;

// encode configuration
struct SDK_CONFIG_ENCODE
{
    SDK_MEDIA_FORMAT dstMainFmt[SDK_ENCODE_TYPE_NUM];           // main stream
    SDK_MEDIA_FORMAT dstExtraFmt[SDK_EXTRATYPES];               // Extra stream
};

```

```
        SDK_MEDIA_FORMAT dstSnapFmt[SDK_ENCODE_TYPE_NUM];        // Snapshot
};

// all of channel encode configuration
struct SDK_EncodeConfigAll
{
    SDK_CONFIG_ENCODE vEncodeConfigAll[NET_MAX_CHANNUM];
};

// combine-encode
struct SDK_CombineEncodeConfigAll
{
    SDK_CONFIG_ENCODE vEncodeConfigAll[NET_MAX_COMBINE_NUM];
};

// network configuration
struct SDK_CONFIG_NET_COMMON
{
    char HostName[NET_NAME_PASSWORD_LEN]; ///< hostname
    CONFIG_IPAddress HostIP;           ///< IP
    CONFIG_IPAddress Submask;          ///< Netmask
    CONFIG_IPAddress Gateway;          ///< NetGateway
    int HttpPort;                       ///< HTTP port
    int TCPPort;                         ///< TCP port
    int SSLPort;                         ///< no used
    int UDPPort;                         ///< no used
    int MaxConn;                         ///< max connect
    int MonMode;                         ///< translation protocol: 0:TCP, 1:UDP, 2: MCAST, only support
    TCP now
    int MaxBps;                          ///< no used
    int TransferPlan;                    ///< Translation policy: 0: AUTO 1: Quality first 2: fluency first
    bool bUseHSDownload;                 ///< flag of high speed download
};

// PTZ configuration
struct SDK_STR_CONFIG_PTZ
{
    char sProtocolName[NET_MAX_PTZ_PROTOCOL_LENGTH]; ///< Protocol name
    int  ideviceNo;           ///< device No.
    int  iNumberInMatrixs;    ///< No. in Matrixs
    int  iPortNo;             ///< Port No. [1, 4]
    SDK_COMMATTRI dstComm;    ///< comm attribute
};
```



```
};  
// all channel of PTZ configuration  
struct SDK_STR_PTZCONFIG_ALL  
{  
    SDK_STR_CONFIG_PTZ ptzAll[NET_MAX_CHANNUM];  
};  
// 232 configuration  
struct SDK_CONFIG_COMM_X  
{  
    char iProtocolName[32]; // Protocol name: "Console"  
    int iPortNo;           // Port No.  
    SDK_COMMATTRI aCommAttri; // comm attribute  
};  
  
// all channel of 232 configuration  
struct SDK_CommConfigAll  
{  
    SDK_CONFIG_COMM_X vCommConfig[SDK_COM_TYPES];  
};  
  
///< IP Fliter  
struct SDK_NetIPFilterConfig  
{  
    bool Enable; // enable  
    CONFIG_IPAddress BannedList[NET_MAX_FILTERIP_NUM]; // black list  
    CONFIG_IPAddress TrustList[NET_MAX_FILTERIP_NUM]; // white list  
};  
  
///< multicast  
struct SDK_NetMultiCastConfig  
{  
    bool Enable; // enable  
    SDK_RemoteServerConfig Server; // multicast server  
};  
  
///< pppoe  
struct SDK_NetPPPoEConfig  
{  
    bool Enable; // enable  
    SDK_RemoteServerConfig Server; // PPPoE server  
    CONFIG_IPAddress addr; // ip get from PPPoE dial
```

```
};

///< DDNS
struct SDK_NetDDNSConfig
{
    bool Enable; ///< enable
    char DDNSKey[NET_NAME_PASSWORD_LEN]; ///< the type of DDNS name
    char HostName[NET_NAME_PASSWORD_LEN]; ///< hostname
    SDK_RemoteServerConfig Server; ///< DDNS server
};

///< DDNS
struct SDK_NetDDNSConfigALL
{
    SDK_NetDDNSConfig ddnsConfig[5];
};

///< FTP
struct SDK_FtpServerConfig {
    ///< enable
    bool Enable;
    ///< FTP server
    SDK_RemoteServerConfig Server;
    ///< spare server IP
    CONFIG_IPAddress SpareIP;
    ///< path name in FTP server
    char RemotePathName[NET_MAX_PATH_LENGTH];
    ///< max file lenght
    int FileMaxLen;
    ///< upload periods
    SDK_TIMESECTION UpLoadPeriod[NET_N_MIN_TSECT];
};

///< NTP
struct SDK_NetNTPConfig
{
    ///< enable
    bool Enable;
    ///< NTP server
    SDK_RemoteServerConfig Server;
    ///< update period
    int UpdatePeriod;
};
```

```
    ///< time zone
    int TimeZone;
};
#define NET_MAX_EMAIL_TITLE_LEN 64
#define NET_MAX_EMAIL_RECIEVERS 5
#define NET_EMAIL_ADDR_LEN 32

///< EMAIL
struct SDK_NetEmailConfig
{
    ///< enalbe
    bool Enable;
    ///< smtp server
    SDK_RemoteServerConfig Server;
    ///< is need SSL ?
    bool bUseSSL;
    ///< sender address
    char SendAddr[NET_EMAIL_ADDR_LEN];
    ///< receiver
    char Recievers[NET_MAX_EMAIL_RECIEVERS][NET_EMAIL_ADDR_LEN];
    ///< email title
    char Title[NET_MAX_EMAIL_TITLE_LEN];
    ///< time section
    SDK_TIMESECTION Schedule[NET_N_MIN_TSECT];
};

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

///< audio format for audio talk
struct SDK_AudioInFormatConfig
{
    int iBitRate;    ///< bitrate, (unit:kbps)
    int iSampleRate; ///< sample rate(unit:Hz)
    int iSampleBit;  ///< sample bit
    int iEncodeType; ///< type of encode, see refer to AudioEncodeTypes
};
```

```

/// alarm status
struct SDK_DVR_ALARMSTATE
{
    int iVideoMotion; ///< motion detect status, bit mask for channel, bit0 means channel 1, and so on, 1:
    alarming 0: normal
    int iVideoBlind; ///< blind detect status, bit mask for channel, bit0 means channel 1, and so on, 1:
    alarming 0: normal
    int iVideoLoss;    ///< loss detect status, bit mask for channel, bit0 means channel 1, and so on, 1:
    alarming 0: normal
    int iAlarmIn;     ///< alarm in status, bit mask for channel, bit0 means channel 1, and so on, 1: alarming
    0: normal
    int iAlarmOut;    ///< alarm out status, bit mask for channel, bit0 means channel 1, and so on, 1:
    alarming 0: normal
};

// channel status
struct SDK_DVR_CHANNELSTATE
{
    bool bRecord; ///< is recording
    int iBitrate; ///< bitrate
};

// device work status
struct SDK_DVR_WORKSTATE
{
    SDK\_DVR\_CHANNELSTATE vChnState[NET_MAX_CHANNUM];
    SDK\_DVR\_ALARMSTATE vAlarmState;
};

```

2.1.5 Network keyboard define

```

/// KEY VALUE
enum SDK_NetKeyBoardValue
{
    SDK_NET_KEY_0, SDK_NET_KEY_1, SDK_NET_KEY_2, SDK_NET_KEY_3, SDK_NET_KEY_4, SDK_NET_KEY_5,
    SDK_NET_KEY_6, SDK_NET_KEY_7, SDK_NET_KEY_8, SDK_NET_KEY_9,
    SDK_NET_KEY_10, SDK_NET_KEY_11, SDK_NET_KEY_12, SDK_NET_KEY_13, SDK_NET_KEY_14, SDK_NET_KEY_15,
    SDK_NET_KEY_16, SDK_NET_KEY_10PLUS,
    SDK_NET_KEY_UP = 20,    // UP
    SDK_NET_KEY_DOWN,      // DOWN
    SDK_NET_KEY_LEFT,      // LEFT

```

```
SDK_NET_KEY_RIGHT, // RIGHT
SDK_NET_KEY_SHIFT,
SDK_NET_KEY_PGUP, // PAGE UP
SDK_NET_KEY_PGDN, // PAGE DOWN
SDK_NET_KEY_RET, // ENTER
SDK_NET_KEY_ESC, // ESC
SDK_NET_KEY_FUNC, // FUNC
SDK_NET_KEY_PLAY, // PLAY/PAUSE
SDK_NET_KEY_BACK, // BACK
SDK_NET_KEY_STOP, // STOP
SDK_NET_KEY_FAST, // FAST
SDK_NET_KEY_SLOW, // SLOW
SDK_NET_KEY_NEXT, // NEXT FILE
SDK_NET_KEY_PREV, // PREV FILE
SDK_NET_KEY_REC = 40, // ENTER RECORD SETTING PAGE
SDK_NET_KEY_SEARCH, // ENTER RECORD SEARCH PAGE
SDK_NET_KEY_INFO, // ENTER SYSTEM INFO PAGE
SDK_NET_KEY_ALARM, // ENTER ALARM OUT PAGE
SDK_NET_KEY_ADDR, // ENTER REMOTE ADDRESS SETTING PAGE
SDK_NET_KEY_BACKUP, // ENTER BACKUP PAGE
SDK_NET_KEY_SPLIT, // NEXT SPLIT MODE
SDK_NET_KEY_SPLIT1, // SLPIT MODE 1
SDK_NET_KEY_SPLIT4, // SLPIT MODE 4
SDK_NET_KEY_SPLIT8, // SLPIT MODE 8
SDK_NET_KEY_SPLIT9, // SLPIT MODE 9
SDK_NET_KEY_SPLIT16, // SLPIT MODE 16
SDK_NET_KEY_SHUT, // SHUTDOWN
SDK_NET_KEY_MENU, // MENU
SDK_NET_KEY_PTZ = 60, // ENTER PTZ CONTROL PAGE
SDK_NET_KEY_TELE, // ZOOM -
SDK_NET_KEY_WIDE, // ZOOM +
SDK_NET_KEY_IRIS_SMALL, // APERTURE -
SDK_NET_KEY_IRIS_LARGE, // APERTURE +
SDK_NET_KEY_FOCUS_NEAR, // FOCUS -
SDK_NET_KEY_FOCUS_FAR, // FOCUS +
SDK_NET_KEY_BRUSH, // BRUSH
SDK_NET_KEY_LIGHT, // LIGHT
SDK_NET_KEY_SPRESET, // SET PRESET POINT
SDK_NET_KEY_GPRESET, // GOTO PRESET POINT
SDK_NET_KEY_DPRESET, // CLEAR PRESET POINT
SDK_NET_KEY_PATTERN, // PATTERN
SDK_NET_KEY_AUTOSCAN, // AUTO-SCAN ON/OFF
SDK_NET_KEY_AUTOTOUR, // AUTO-TOUR ON/OFF
```

```
    SDK_NET_KEY_AUTOPAN,    // AUTO-PAN ON/OFF
};

///  
///  
enum SDK_NetKeyBoardState  
{  
    SDK_NET_KEYBOARD_KEYDOWN,    // key down  
    SDK_NET_KEYBOARD_KEYUP,    // key up  
};  
  
struct SDK_NetKeyBoardData  
{  
    int iValue;    // see refer to SDK_NetKeyBoardValue  
    int iState;    // see refer to SDK_NetKeyBoardState  
};
```

3 API Definition

3.1 SDK Initialization

1. H264_DVR_API `long` H264_DVR_GetLastError();

- API description: It is to return function failure code. when you failed to call the following interface, you can call this function to get error code.
- Parameter: none
- Return: Please see refer to [error code](#)
- Reference API:

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

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

- API description: Initialize SDK, calling before all SDK function
- Parameter:
 - cbDisConnect*
Disconnect callback function. It is to callback disconnect device excluding device logout successfully(call H264_DVR_Logout(), set it as 0 when forbid callbacking.
 - [in]dwUser*
User data

🔗 Callback function Parameters:

lLoginID

Login handle

pchDVRIP

Device IP

nDVRPort

Port

dwUser

User data, just the same with the above user data you have input.

- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: H264_DVR_Cleanup

3. CLIENT_API `void` H264_DVR_Cleanup ();

- API description: Clean up SDK and release occupied resource, calling after all SDK

function.

- **Parameter:** none
- **Return:** none
- **Reference API:** H264_DVR_Init

3.2 Get alarm status

```
typedef bool (__stdcall *fMessCallBack)(long lLoginID, char *pBuf,
                                         unsigned long dwBufLen, long dwUser);
H264_DVR_API bool H264_DVR_SetDVRMessCallBack(fMessCallBack
                                              cbAlarmcallback, unsigned long lUser);
```

- **API description:** Set device message callback function to get device current state. Callback order does not matter here. SDK default setting is not to callback. You need to call alarm message subscription interface H264_DVR_SetupAlarmChan(). It applies to current defined alarm status. Device state is callbacked every second.


- **Parameter:**

cbAlarmcallback

Message callback function. It is to callback device status (such as alarm status). When it is 0, system disables callback.

[in] lUser

user self-defined data

 Callback function parameters:

lLoginID

Return value of H264_DVR_Login

pBuf

Refer to see [SDK_AlarmInfo](#)

dwBufLen

pBuf length. Unit is byte.

dwUser

User self-defined data

- **Return:** Succeeded: TRUE, Fail: FALSE
- **Reference API:** H264_DVR_SetupAlarmChan、H264_DVR_CloseAlarmChan

4. H264_DVR_API long H264_DVR_SetupAlarmChan(long lLoginID);

- **API description:** Start listening device message. This function is to set callbacking device message or not. Message is callbacked from H264_DVR_SetDVRMessCallBack.

- **Parameter:**

[in]lLoginID

Return value of H264_DVR_Login

- Return: Succeeded: TRUE, Fail: FALSE
- Reference API : H264_DVR_SetDVRMessCallBack , H264_DVR_CloseAlarmChan

5. H264_DVR_API `bool` H264_DVR_CloseAlarmChan(`long` lLoginID);

- API description: Stop lisening one device
- Parameter:
[in]lLoginID
Return value of H264_DVR_Login
- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: H264_DVR_SetupAlarmChan

3.3 Device Registration

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

■ **API description:** Login. When device set the user as reuse(device default user such as admin can be reused.). this account can registered several time.

■ **Parameter:**

[in] sDVRIP

device IP

[in] wDVRPort

device port

[in] sUserName

user name

[in] sPassword

password

[out] lpDeviceInfo

device property. it is a output parameter.

[out] error

(when the function returned successfully, the parameter is null.Please refer to [error code](#).)

■ **Return:** Return 0 if failed. Return device ID if succeeded. Using this value(device handle)all operations after successfully log in can corresponding to the device.

■ **Reference API:** H264_DVR_Logout

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

■ **API description:** Register a user to device extension port. support one user specify device.

■ **Parameter:**

[in] sDVRIP

device IP

[in] wDVRPort

device port

[in] sUserName

user name

[in] sPassword

```

        password
[out] lpDeviceInfo
        device property. it is a output parameter.

```

[in] nType

the type as follows:

```

enum LoginType
{
    LOGIN_TYPE_GUI,           ///< Local GUI
    LOGIN_TYPE_CONSOLE,      ///< Console
    LOGIN_TYPE_WEB,          ///< WEB
    LOGIN_TYPE_SNS,          ///< SNS
    LOGIN_TYPE_MOBIL,        ///< Mobile terminal
    LOGIN_TYPE_NETKEYBOARD,  ///< Netkeyboard
    LOGIN_TYPE_SERVER,       ///< Center servers
    LOGIN_TYPE_AUTOSEARCH,   ///< IP search tool
    LOGIN_TYPE_UPGRADE,      ///< Upgrade tool
    LOGIN_TYPE_MEGAEEYE,     ///< Megaeye
    LOGIN_TYPE_NR,
};

```

[out] error

(when the function returned successfully, the parameter is null.Please refer to [error code](#).)

- **Return:** Return 0 if failed. Return device ID if succeeded.Using this value(device handle)all operations after successfully logged in can corresponding to the device.
- **Reference API:** H264_DVR_Logout

8. H264_DVR_API `long` H264_DVR_Logout(`long` lLoginID)

- **API description:** Logout user
- **Parameter:**
[in]lLoginID
Return value of H264_DVR_Login
- **Return:** Succeeded: TRUE, Fail: FALSE
- **Reference API:** H264_DVR_Login

3.4 Real-time Monitor

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

- **API description:** Start real-time monitor.
- **Parameter:**

[in]lLoginID
Return value of H264_DVR_Login

[in] [lpClientInfo](#)
Client information

- **Return:** Return real-time monitor handle if succeeded, return 0 if failed.
- **Reference API:** H264_DVR_StopRealPlay, H264_DVR_SetRealDataCallBack

10. H264_DVR_API `bool` H264_DVR_StopRealPlay(`long` lRealHandle);


- **API description:** Stop real-time monitor
- **Parameter:**
[in]lRealHandle
Return value of H264_DVR_RealPlay
- **Return:** Succeeded: TRUE, Fail: FALSE
- **Reference API:** H264_DVR_RealPlay

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

```
typedef int(__stdcall *fRealDataCallBack) (long lRealHandle, long
dwDataType, unsigned char *pBuffer, long lbufsize, long dwUser);
```

- **API description:** Set real-time monitor data callback and provides you with data from the device .When cbRealData is NULL, callback ends.

- **Parameter:**
[in]lRealHandle
Return value of H264_DVR_RealPlay
cbRealData
It is a callback function to output the current real-time data from the device.
[in]dwUser
User data

 Callback function parameters:

lRealHandle
Return value of H264_DVR_RealPlay

dwDataType

0: original data

1: Frame data

2: yuv data

3: pcm audio data

pBuffer

call-back data. Everytime calling back data of different lengths according to the different data types (except type

0). Other data types are based on frames, every time it calls back one frame.

dwBufSize

length of callback data.(Unit:byte).

dwUser

User self-defined

- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: H264_DVR_RealPlay、H264_DVR_StopRealPlay

3.5 Playback And Download

12. H264_DVR_API long H264_DVR_FindFile(long lLoginID, H264_DVR_FINDINFO* lpFindInfo, H264_DVR_FILE_DATA *lpFileData, int lMaxCount, int *findcount, int waittime = 2000);

- API description: Search record files

- Parameter:

[in]lLoginID

Return value of H264_DVR_Login

[in]lpFindInfo

Search condition [H264_DVR_FINDINFO](#)

[out]lpFileData

Returned record file information.It is a [H264_DVR_FILE_DATA](#) structure array.

[in]lMaxCount

The length of lpFileData (Unit: BYTE, The value shall between 100~200*sizeof([H264_DVR_FILE_DATA](#)))

[out]filecount

Returned file amount; It is an output max parameter. You can only search the video files before buffer is full

[in]waittime

Waitting time

- Return: Succeeded: TRUE, Fail: FALSE

- Reference API:

H264_DVR_Login,H264_DVR_PlayBackByName,H264_DVR_StopPlayBack, H264_DVR_PlayBackControl, H264_DVR_GetFileByName

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

13. H264_DVR_API long H264_DVR_PlayBackByName(long lLoginID,

```
H264_DVR_FILE_DATA *sPlayBackFile, fDownloadPosCallBack
cbDownloadPos, fRealDataCallBack fDownloadDataCallBack, long
dwDataUser);
```

- **API description:** Network playback. Please note, when you login one device, one channel can only play one record at one time, while multi-records of the same channel can't be opened simutaniously.

- **Parameter:**

[in]lLoginID

Return value of H264_DVR_Login

[in] sPlayBackFile

Recorded file information return by H264_DVR_FindFile

[in] cbDownloadPos

Progress call-back function

[in] fDownloadDataCallBack

Video data call-back function

[in]dwUserData

User self-defined data

🔗 CallBack function:

lPlayHandle

Return value of H264_DVR_PlayBackByName

dwTotalSize

Current total play size, unit is KB.

dwDownloadSize

Played size, unit is KB. When value is -1, it means cureent playback is over.

dwUser

User data, just the same with user data in the above.

- **Return:** Return network playbackID if succeeded, return 0 if failed.

- **Reference API:**

H264_DVR_Login,H264_DVR_FindFile,H264_DVR_StopPlayBack,
H264_DVR_PlayBackControl

14. H264_DVR_API `bool` H264_DVR_StopPlayBack(`long` lPlayHandle);

- **API description:** Stop playback

- **Parameter:**

[in]lPlayHandle

Playback handle, Such as the return value of H264_DVR_PlayBackByName

- **Return:** Succeeded: TRUE, Fail: FALSE

- **Reference API:** H264_DVR_PlayBackByName

15. H264_DVR_API `long` H264_DVR_GetFileByName(`long` lLoginID, H264_DVR_FILE_DATA *sPlayBackFile, `char` *sSavedFileName, fDownloadPosCallBack cbDownloadPos = NULL, `long` dwDataUser = NULL);

■ **API description:** Download recorded files via the searched information

■ **Parameter:**

[in]lLoginID

The return value of H264_DVR_Login

[in] sPlayBackFile

Recorded file information pointer.

[in]sSavedFileName

The name of file to save(full path).

[in]cbDownloadPos

Download process calls user self-defined data. For download process callback function, please refer to H264_DVR_GetDownloadPos

[in]dwUserData

Download process calls user self-defined data.

🔗 **CallBack function:**

lPlayHandle

Return value of H264_DVR_PlayBackByName

dwTotalSize

Current total play size, unit is KB.

dwDownloadSize

Played size, unit is KB. When value is -1, it means current playback is over.

dwUser

User data, just the same with user data in the above.

■ **Return:** Return download ID if succeeded. Return 0 if failed.

■ **Reference API:** H264_DVR_StopGetFile、H264_DVR_GetDownloadPos

16. H264_DVR_API `bool` H264_DVR_StopGetFile(`long` lFileHandle);

■ **API description:** Stop downloading files.

■ **Parameter:**

[in]lFileHandle

The return value of H264_DVR_GetFileByName

■ **Return:** Succeeded: TRUE, Fail: FALSE

■ **Reference API:** H264_DVR_GetFileByName、H264_DVR_GetDownloadPos

17. H264_DVR_API `int` H264_DVR_GetDownloadPos(`long` lFileHandle);

- **API description:** It is to get current downloading place for interfaces that do not need to show real-time download progress. It is similar to the download callback function.
- **Parameter:**
[in]lFileHandle
The return value of H264_DVR_GetFileByName
- **Return:** position(value between 0 to 100)
- **Reference API:** H264_DVR_GetFileByName、H264_DVR_StopGetFile

3.6 Playback Control

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

- **API description:** Pause , Resume, Seek network playback
- **Parameter:**
[in]lPlayHandle
Play handle, return by H264_DVR_GetFileByName
[in] lControlCode
`enum` SEDK_PlayBackAction
{
 SDK_PLAY_BACK_PAUSE, // Pause
 SDK_PLAY_BACK_CONTINUE, // Resume
 SDK_PLAY_BACK_SEEK, // Seek
};
- **Return:** Succeeded: TRUE, Fail: FALSE
- **Reference API:** H264_DVR_PlayBackByName、H264_DVR_StopPlayBack

3.7 PTZ Control

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

- **API description:** PTZ control
- **Parameter:**
[in]lLoginID
Return value of H264_DVR_Login
[in] nChannelNo
Channel NO. start with 0
[out] lPTZCommand

Commands, see refer to [PTZ_ControlType](#)

[in] bStop

Whether stop or not. It applies to PTZ direction and lens operation. When you operate other functions, input this parameter as FALSE.

[out] lSpeed

Step/Speed. The value ranges from 1 to 8. 8 has the highest control capability(4 by default). dwStep is the preset value when you use preset function.

- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: H264_DVR_Login, H264_DVR_RealPlay

3.8 System Configuration

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

- API description: Get device configuration information.

- Parameter:

[in]lLoginID

The return value of H264_DVR_Login

[in]dwCommand

Configuration type, Please refer to [SDK_CONFIG_TYPE](#)

[in]nChannelNO

Channel number. Set as -1 to configure all channels. The parameter is null if command does not need the channel number.

[out] lpOutBuffer

Receive data buffer pointer, buffer length depend on configuration structure size.

[in]dwOutBufferSize

Receive data buffer length(Unit:byte)

[out]lpBytesReturned

The data length actually received.

[in]waittime

Waiting time

- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: H264_DVR_SetDevConfig

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

■ API description: Set device configuration information

■ Parameter:

[in]lLoginID

The return value of H264_DVR_Login

[in]dwCommand

Configuration type, Please refer to [SDK_CONFIG_TYPE](#)

[in]nChannelNO

Channel number. Set as -1 to configure all channels. The parameter is null if command does not need the channel number.

[in]lpInBuffer

Data buffer pointer

[in]dwInBufferSize

Data buffer length(unit is byte).

[in]waittime

Waiting time

■ Return: Succeeded: TRUE, Fail: FALSE

■ Reference API: H264_DVR_GetDevConfig

3.9 Log Management

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

■ API description: Log query

■ Parameter:

[in]lLoginID

The return value of H264_DVR_Login

[in] pFindParam

Search condition, Please refer to [SDK_LogSearchCondition](#)

[in] pRetBuffer

The return of log information, Please refer to [SDK_LogList](#)

[in] lBufSize

The return length of log information

[in] waittime

Waiting time

■ Return: Succeeded: TRUE, Fail: FALSE

- Reference API: None

3.10 Remote Control

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

- API description: Reboot device and clear log
- Parameter:
 - [in]lLoginID
The return value of H264_DVR_Login
 - [in] type
0: reboot device, 1: clear log
- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: None

3.11 Upgrade

```
typedef void(__stdcall *fUpgradeCallBack) (long lLoginID, long
    lUpgradechannel, int nTotalSize, int nSendSize, long dwUser);
24. H264_DVR_API long H264_DVR_Upgrade(long lLoginID, char *sFileName,
    int nType = 0, fUpgradeCallBack cbUpgrade = NULL, long dwUser =
    0);
```

- API description: Upgrade device
- Parameter:
 - [in]lLoginID
The return value of H264_DVR_Login
 - [in] sFileName
The name of upgrade file
 - [in] nType
The type of upgrade file
 - `enum` UpgradeTypes


```
{
            UPGRADE_TYPES_SYSTEM,    ///< System
            UPGRADE_TYPES_NR,
          };
```

🔗 CallBack function:

fUpgradeCallBack

Callback of upgrade progress, lUpgradechannel is the upgrade handle, return by H264_DVR_Upgrade

nTotalSize
Upgrade file length, (Unit: BYTE)

nSendSize
data length have been upgraded, (unit: BYTE)

[in]dwUser
User self-define data

- Return: Succeeded: return upgrade handle, Fail: FALSE
- Reference
API: H264_DVR_GetUpgradeState, H264_DVR_CloseUpgradeHandle

25. H264_DVR_API long H264_DVR_CloseUpgradeHandle(long lUpgradeHandle);

- API description: stop upgrade
- Parameter:
[in]lUpgradeID
The return value of H264_DVR_Upgrade
- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: H264_DVR_Upgrade

26. H264_DVR_API int H264_DVR_GetUpgradeState(long lUpgradeHandle)

- API description: get upgrade status
- Parameter:
[in] lUpgradeHandle
The return value of H264_DVR_Upgrade
- Return: 1: Succeeded 2: Upgrading, 3: FALSE
- Reference API: H264_DVR_Upgrade, H264_DVR_CloseUpgradeHand

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

- API description: Search device in LAN
- Parameter:
[in] szBuf
Buffer to receive structure [SDK_CONFIG_NET_COMMON](#), return the a structure as long as it search a device.
[in] nBufLen

szBuf buffer length

[in] pRetLen

Return the structure total length of [SDK_CONFIG_NET_COMMON](#)

[in] nSearchTime

Waiting time

- Return: Succeeded: TRUE, Fail: FALSE
- Reference API:

3.11 Audio Talk

```
typedef void (__stdcall *pfAudioDataCallBack)(long lVoiceHandle, char *pDataBuf,
                                             long dwBufSize, char byAudioFlag, long dwUser);
```

28. H264_DVR_API long H264_DVR_StartVoiceCom_MR (long lLoginID, pfAudioDataCallBack pVcb,
 Long dwDataUser);

- API description: Send audio talk request to device.
- Parameter:

[in] lLoginID

The return value of H264_DVR_Login

[in] pVcb

Self-defined data callback interface.

[in] dwDataUser

Self-defined data. Returned to you via callback function

- Return: >0 audio talk handle, <= FALSE
- Reference API:

H264_DVR_VoiceComSendData, H264_DVR_StopVoiceCom, H264_DVR_SetTalkMode

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

- API description: Send user audio data to device.
- Parameter:

[in] lVoiceHandle

Return value of H264_DVR_StartVoiceCom_MR

[in] pSendBuf

The audio data to be sent out.

[in] lBufSize

Audio data length to be sent out.(Unit:byte)

■ **Return:** Succeeded: TRUE, Fail: FALSE

■ **Reference API:**

H264_DVR_StartVoiceCom_MR H264_DVR_StopVoiceCom H264_DVR_SetTalkMode

30. H264_DVR_API **bool** H264_DVR_StopVoiceCom (**long** lVoiceHandle);

■ **API description:** Stop audio talk

■ **Parameter:**

[in] lVoiceHandle

The return value of H264_DVR_StartVoiceCom_MR

■ **Return:** Succeeded: TRUE, Fail: FALSE

■ **Reference API:**

H264_DVR_StartVoiceCom_MR H264_DVR_VoiceComSendData H264_DVR_SetTalkMode

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

■ **API description:** set audio talk mode

■ **Parameter:**

0- *[in]* lLoginID

The return value of H264_DVR_Login

1- *[in]* pTalkMode

Mode of audio talk ,see refer to [SDK_AudioInFormatConfig](#)

■ **Return:** Succeeded: TRUE, Fail: FALSE

■ **Reference API:**

H264_DVR_StartVoiceCom_MR H264_DVR_VoiceComSendData H264_DVR_StopVoiceCom

3.12 Record Mode

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

- **API description:** set record mode

- **Parameter:**

`lLoginID`

The return value of H264_DVR_Login

`[in]` `nChannelNo`

Channel No., -1 all of channel

`[in]` `lRecordType`

Record mode, see refer to [SDK_RecordModeTypes](#)

- **Return:** Succeeded: TRUE, Fail: FALSE

- **Reference API:** H264_DVR_StopDVRRecord

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

- **API description:** stop record

- **Parameter:**

`[in]` `lLoginID`

The return value of H264_DVR_Login

`[in]` `nChannelNo`

Channel No., -1: all of channel

- **Return:** Succeeded: TRUE, Fail: FALSE

- **Reference API:** H264_DVR_StartDVRRecord

3.13 Set System Time

34. H264_DVR_API `bool` H264_DVR_SetSystemDateTime (`long` lLoginID, [SDK_SYSTEM_TIME](#) *pSysTime);

- **API description:** set system time

- **Parameter:**

`[in]` `lLoginID`

The return value of H264_DVR_Login

`[in]` `pSysTime`

System time, see refer to [SDK SYSTEM TIME](#)

- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: None

3.14 Get Device status

35. H264_DVR_API `bool` H264_DVR_GetDVRWorkState(`long` lLoginID, [SDK DVR WORKSTATE](#) *pWorkState);

- API description: Get device working status
- Parameter:

[in] lLoginID

The return value of H264_DVR_Login

0- *[in]* pWorkState

Structure of work staus, see refer to [SDK DVR WORKSTATE](#)

- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: None

3.15 Net Keyboard

36. H264_DVR_API `bool` H264_DVR_ClickKey (`long` lLoginID, [SDK NetKeyboardData](#) *pKeyboardData);

- API description: send net keyboard message
- Parameter:

[in] lLoginID

The return value of H264_DVR_Login

0- *[in]* pKeyboardData

Key value, see refer to [SDK NetKeyboardData](#)

- Return: Succeeded: TRUE, Fail: FALSE
- Reference API: None