

Network Client SDK Programming Guide For Network Layer

Directory

Directory	1
1 Brief	5
2 Declare	5
3 Version update records	6
4 Data Structure Description	7
4.1 General structure	7
4.1.1 Frame Type	7
4.1.2 Time information	7
4.1.3 Absolute time information	8
4.1.4 Buffer Information	8
4.1.5 The error code enumeration	8
4.1.6 Query status	9
4.1.7 Resolution Index	9
4.1.8 Resolution Capacity Mask	9
4.2 System SDK Properties	10
4.3 User Login Logout management	10
4.3.1 User login parameters	10
4.3.2 Forced to delete the logged in user callbacks	10
4.4 Event notification subscription	11
4.4.1 Remote Event List Type	11
4.4.2 Users event callback	12
4.4.3 Alarm event type	14
4.5 Device configuration information	14
4.5.1 Remote configuration information	14
4.5.2 Configure Message Type list	15
4.6 Live Stream	19
4.6.1 Streaming mode	19
4.6.2 Streaming media types	19
4.6.3 Live stream connection parameters	19
4.6.4 Network Frame Header	20
4.6.5 Streaming data callback	20
4.7 History Flow	20
4.7.1 History stream video type	20
4.7.2 History query stream	21
4.7.3 History stream query result	21

4.7.4	History stream connection parameters	21
4.7.5	Historical flow data callback	21
4.7.6	History positioning operation flow direction enumeration	22
4.8	Voice Intercom	22
4.8.1	Voice stream connection parameters	22
4.8.2	Voice stream data callback	22
4.9	Log Query	23
4.9.1	Log query	23
4.9.2	History query results log	23
4.10	Clear Channel	23
4.10.1	Serial Type	23
4.10.2	Clear Channel Parameters	24
4.10.3	Transparent channel data callback	24
4.11	PTZ control	24
4.11.1	PTZ PTZ control code type	24
4.12	File Query	27
4.12.1	Document types of queries	27
4.12.2	File query	28
4.12.3	File search results	28
4.13	File upload and download	28
4.13.1	Transfer the file type	28
4.13.2	File Transfer Control	28
4.13.3	File upload parameters	28
4.13.4	File upload status	29
4.13.5	File upload data callback	29
4.13.6	Download parameters	29
4.13.7	Download data callback	30
4.14	Remote Control	30
4.14.1	Remote device control	30
4.14.2	Disk group type	30
4.14.3	Storage Management	31
4.14.4	File lock operation	31
4.14.5	Grab	31
4.15	Device Registration	32
4.15.1	Device Registration Information	32
4.15.2	Device Registration callback	32
4.16	Streaming media control	32
4.16.1	Mission Control	32
4.16.2	Media Function Type	32
4.16.3	Query results streaming connection ID	33
4.16.4	Streaming data callback	33
4.17	Font information parameters	33
4.18	Remote panel status information	34
4.19	Remote panel control parameters	34

4.20	Frame Header.....	35
4.21	Probe Device (Search Device)Parameter.....	35
4.21.1	Probe Setting Parameter.....	36
4.21.2	Probe device configuration parameters.....	36
4.21.3	Probe callback.....	36
4.22	Retrieve Device List(NVR/IPD Probed not PC).....	36
4.22.1	Probed device list.....	36
4.22.2	Probe callback.....	37
4.23	Preview Cruise for IPD.....	37
4.23.1	Preview Cruise List	37
4.23.2	Start and stop cruise.....	39
4.23.3	fullscreen or resume.....	39
5	SDK API Description	40
5.1	System initialization and anti-initialization.....	40
5.2	System SDK Properties.....	40
5.3	User login, exit, user event management.....	41
5.4	Get and set the device configuration information	44
5.5	Real-time streaming operations.....	45
5.6	History query stream.....	45
5.7	History flow operation	46
5.8	Voice talk	49
5.9	Log Query.....	50
5.10	Transparent Channel	51
5.11	PTZ control	52
5.12	File Query.....	52
5.13	File upload and download	53
5.14	Remote control operation.....	56
5.15	Data Query.....	59
5.16	Device Registration	60
5.17	Convert string lattice	60
5.18	Streaming media control	61
5.19	Remote Control Panel	62
5.20	Device Probe.....	63
	int VideoNetClient_DeviceProbeSetDeviceConfig(IN const BYTE *pMac, IN WORD wProbePort, IN const LPDeviceProbeConfig cConfig);.....	64
	int VideoNetClient_DeviceProbeStartV2(IN const LPDeviceProbeParameter cParameter, IN CB_DeviceProbeV2 cbDeviceProbeV2).....	64
	int VideoNetClient_DeviceProbeRefreshV2(IN const LPDeviceProbeParameter cParameter).....	64
	int VideoNetClient_DeviceProbeStopV2 ().....	65
5.21	Device Probe (From NVR)	65
	Return 0 for success, otherwise an error code.....	65

	Return 0 for success, otherwise an error code.....	66
	Return 0 for success, otherwise an error code.	66
	Return 0 for success, otherwise an error code.	66
6	SDK call order and the samples code	67
6.1	call order as shown below:	67
6.2	Examples.	67
6.2.1	login, logout and subscribe events.	67
6.2.2	Device configuration information operations	69
6.2.4	Voice talk	73
6.2.7	File Query	79
6.2.8	Transparent Channel Operation.....	81
6.2.9	File download	82
6.2.10	File upload operation	84
6.2.11	Remote command control.....	86

1 Brief

Network client SDK provides two kinds of network interfaces to the developer of application, the network layer SDK (VideoNetClient.dll) and Client Unit SDK interfaces (VNetCIU.dll), the main achievement of the network layer is configure devices, request media data and implement the interaction of control commands, **NO** playback; Client Unit SDK package is a wrapped interfaces based on network layer SDK and support playback media data, so the client unit SDK depends on the network layer SDK libraries. If you have not playback needs you can use the network layer SDK independently.

Network layer network client SDK (referred VideoNet Client) provides a set of interfaces for the developers to access and control remote device(DVR,NVR,IPC etc.). We are fully consider the security of device access, SDK version compatibility, as well as the compatibility issues of device. The main features include: real-time video streaming preview, history streaming playback by time (synchronous), video streaming download by time, voice talk (supports G711A, G711U,G726, PCM-IMA encoding format etc.), query and download files, log, users event notification and remote control device.

VideoNetClient SDK include VideoNetClient.h VideoNetClient_Common.h,VideoNetClient_Configure.h, VideoNetClient.dll, VideoNetClient. Lib.

Details as follows:

VideoNetClient.h	Definition of network layer interface
VideoNetClient_Common.h	Definition of data structures network layer
VideoNetClient_Configure.h	Data structure definition for settings
VideoNetClient.dll	DLL
VideoNetClient.lib	Symbol file

- Client SDK maximum support **512** users online and **512** media session.
- After login, you can create a maximum **64** edia sessions (including real-time streaming media sessions, history streams, voice talk, transparent channel, file uploading and downloading, history streaming download, etc.).
- After login, you can create a maximum **32** eal-time streams.
- login, you can create a maximum **32** history streaming.
- After login, you can create a maximum **8** history streaming **query**.
- After login, you can create a maximum **8** log **query**.
- After login, you can create a maximum **8** file **queries**.
- After login, you can create a maximum **8** file upload.
- After login, you can create a maximum **8** file download.
- After login, you can create a maximum **8** transparent channels.
- After login, you can create **a** voice intercom, as the same time the device supports only **one** is speaking states.

2 Declare

- 1, Channel number begins 0. 0 indicates that the 1st channel, the 2rd channel is 1, and so on.
- 2, All interfaces will return the error code.
- 3, If no special description channel number means the audio or video channels.
- 4, Does not support RTP.

3 Version update records

V1.0.6(20150526)

- 1, Support to control IPD's other monitor. More help information added that it help developer works well. If you want control our IPD device than you must be read it.
- 2, Regular some interface so that other program lanuage(VB, Dephi etc.) could be use it.

Version	Date	AM D	Description
V1.0.0	2013/07/11	A	New
V1.0.1	2014/02/14	A	NVR's ISP settings added
V1.0.2	2014/10/16	A	<ol style="list-style-type: none"> 1, Add new interface named VideoNetClient_HistoryStreamMultiTypeQueryConnectEx, device will send query result by downloading file, speed up query process. 2, At the same time SDK support 64 real stream connection. 3, Add 2 command control: recover password of admin user, and check black point(only support IPC). 4, Add ISP and ROI setting.
V1.0.3	2015/02/12	A	<ol style="list-style-type: none"> 1, Add a configuration of remote channel management. 2, Fixed VideoNetClient_GetDeviceConfigFromHttp issue when the ActiveX is running in 360 browser. 3, Add a configuration of the device extension capability sets. 4, Fixed issue when upload file failure. 5, Add a configuration that restore ISP to default. 6, Add a sense mode of ISP configuration.
V1.0.4	2015/03/20	MA	<ol style="list-style-type: none"> 1, Add some interface that get devices from NVR, such as VideoNetClient_DetectDeviceListStart, VideoNetClient_DetectDeviceListStop, VideoNetClient_SetDetectDeviceListFromNVR, VideoNetClient_DetectDeviceListRefresh. 2, Add a Configuration that zoom in or zoom out preview screen for IPD. 3, Add a Configuration of the preview cruise list.
V1.0.5	2015/05/26	A	<ol style="list-style-type: none"> 1, Support to control IPD's other monitor. More help information added that it help developer works well. If you want control our IPD device than you must be read it. 2, Regular some interface so that other program lanuage(VB, Dephi etc.) could be use it.
V1.0.6	2016/03/21	A	<ol style="list-style-type: none"> 1. Add another get and set configuration interface supported the more than 32 channels:

			<p>VideoNetClient_SetConfigV3, VideoNetClient_GetConfigV3</p> <p>2. Add the second version probe interface: VideoNetClient_DeviceProbeStartV2, VideoNetClient_DeviceProbeRefreshV2 VideoNetClient_DeviceProbeStopV2</p> <p>3. Add some Command: PICCFG_MOTION_ACTION_CONF PICCFG_VLOSTS_ACTION_CONF PICCFG_SHELTER_ACTION_CONF PICCFG_ADVANCED_CONF COMPRESSCFG_RESOLUTION_AND_FRAME_TABLE COMPRESSCFG_RESOLUTION_AND_BITRATE_TABLE COMPRESSCFG_VIDEO_INPUT_INFO ALARMCFG_ACTION_MULTICHANNEL_CONF</p>
--	--	--	--

4 Data Structure Description

4.1 General structure

4.1.1 Frame Type

```
eFrameType
typedef enum eFrameType
{
    eFrameError                = 0x0000,
    eFrameIFrames              = 0x0001,
    eFramePFrames              = 0x0002,
    eFrameBPFrames             = 0x0020,
    eFrameBBPFrames            = 0x0004,
    eFrameAudioFrames          = 0x0008,
    eFrameQCIFIFrames          = 0x0010,
    eFrameQCIFPFrames          = 0x0040,
    eFrameBIFrames             = 0x0090,
    eFrameBBIFrames            = 0x00C0,
    eFrameSysHeader            = 0x0080,
    eFrameSFrames              = 0x0200,
    eFrameDspStatus            = 0x0100,
    eFrameAimDetection          = 0x0400,
    eFrameOrigImage            = 0x0800,
    eFrameMotionDetection      = 0x1000
} EFrameType;
```

4.1.2 Time information

```
TimeInfo
typedef struct tagTimeInfo
{
    WORD    wYear;        /*! <Years          */
    WORD    wMonth;       /*! <Month          */
    WORD    wDay;         /*! <Day            */
}
```

```

WORD        wHour;        /*! <Time          */
WORD        wMinute;      /*! <Points        */
WORD        wSecond;      /*! <Seconds       */
} TimeInfo, * LPTimeInfo;

```

Note:

wYear = Absolute Time Year - 1900

wMonth = absolute time in March - 1

4.1.3 Absolute time information

AbsoluteTime

typedef struct tagAbsoluteTime

```

{
    WORD        wYear;        /*! <Years          */
    WORD        wMonth;      /*! <Month          */
    WORD        wWeek;        /*! <Week           */
    WORD        wDay;         /*! <Day            */
    WORD        wHour;        /*! <Hour           */
    WORD        wMinute;      /*! <Minute         */
    WORD        wSecond;      /*! <Second         */
    WORD        wMillisecond; /*! <Millis Seconds */
} AbsoluteTime, * LPAbsoluteTime;

```

4.1.4 Buffer Information

Buffer

typedef struct tagBuffer

```

{
    BYTE* PBuffer; /*! <Media data buffer */
    DWORD dwBufLen; /*! <Media data length */
} Buffer, * LPBuffer;

```

4.1.5 The error code enumeration

eVideoNetClientError

typedef enum eVideoNetClientError

```

{
    eErrorNone    = 0, /*! <Error          */
    eErrorFailed, /*! <Failure        */
    eErrorNoInitializeSDK, /*! <Uninitialized SDK */
    eErrorHandle, /*! <Handle error    */
    eErrorParameter, /*! <Parameter error */
    eErrorBufferNoEnough, /*! <Buffer too small */
    eErrorMemory, /*! <Memory error    */
    eErrorSystemFailed, /*! <Operating system returns an error */
    eErrorNoIdleResource, /*! <No available resources */
    eErrorTimeOut, /*! <Timeout error   */
    eErrorFunctionVersionLow, /*! <Function version is less than the server */
    eErrorFunctionVersionHigh, /*! <Function is later than the server */
    eErrorTaskNoRun, /*! <Task is not started */
    eErrorAlreadyRun, /*! <Repeated start  */

```



```
eErrorConnectFailed,          /*! <Connection failed */
eErrorSessionDisconnect,      /*! <Disconnected */
eErrorCommandSendFailed,     /*! <Command failed to send */
eErrorServerReject,          /*! <Server rejected */
eErrorInvalidUser,           /*! <User handle illegal */
eErrorCallFail,              /*! <Remote call fails */
eErrorCallReplyError,        /*! <Remote call answering error */
eErrorUserName,              /*! <User name error */
eErrorUserPass,              /*! <User password error */
eErrorIPLimited,              /*! <User IP restrictions      */
eErrorMACLimited,            /*! <Restriction by MAC        */
eErrorUserNumOverflow,       /*! <LOGIN too */
eErrorUserHeartBeat,         /*! <User heartbeat Subscribe failure */
eErrorGetConfigurationPort,  /*!<Obtain landing port configuration
information failed */
eErrorServiceReseting,       /*! <Restart the network services */
eErrorNoSupportCommand,      /*! <Unsupported commands */
eErrorNotImplement,          /*! <Unrealized */
} EVideoNetClientError;
```

4.1.6 Query status

```
eQueryStatus
typedef enum eQueryStatus
{
    eQueryOK = 0,              /*! <Inquiries successful */
    eQueryFailed,              /*! <Query failed */
    eQueryBusy,                /*! <Inquiries busy */
    eQueryFinished,            /*! <Inquiries end */
} EQueryStatus;
```

4.1.7 Resolution Index

```
typedef enum eResolutionIndex
{
    eResolutionIndex_CIF,
    eResolutionIndex_D1,
    eResolutionIndex_SD,
    eResolutionIndex_QuadVGA,
    eResolutionIndex_SXGA,
    eResolutionIndex_UXGA,
    eResolutionIndex_720P,
    eResolutionIndex_1080P,
    eResolutionIndex_VGA,
    eResolutionIndex_QXGA, // 2048x1536
    eResolutionIndex_640x360,
    eResolutionIndex_QVGA, //
} eResolutionIndex;
```

4.1.8 Resolution Capacity Mask

```
#define DVR_MAX_STREAM_TYPE    2
#define DVR_CAP_NORMAL_H264    0x01
#define DVR_CAP_ADVANCED_H264  0x02
#define DVR_CAP_RESOLUTION_QCIF 0x01
```

```
#define DVR_CAP_RESOLUTION_CIF 0x02
#define DVR_CAP_RESOLUTION_2CIF 0x04 //real-time CIF
#define DVR_CAP_RESOLUTION_D1 0x08 //real-time D1
#define DVR_CAP_RESOLUTION_4CIF_SIM 0x10 //not real-time D1
#define DVR_CAP_RESOLUTION_2CIF_SIM 0x20 //not real-time CIF
#define DVR_CAP_RESOLUTION_QVGA 0x40 //320*240
#define DVR_CAP_RESOLUTION_VGA 0x80 //640*480
#define DVR_CAP_RESOLUTION_SVGA 0x100 //800*600
#define DVR_CAP_RESOLUTION_XVGA 0x200 //1024*768
#define DVR_CAP_RESOLUTION_HD720 0x400 //1280*720
#define DVR_CAP_RESOLUTION_QuadVGA 0x800 //1280*960
#define DVR_CAP_RESOLUTION_SXGA 0x1000 //1280*1024
#define DVR_CAP_RESOLUTION_UXGA 0x2000 //1600*1200
#define DVR_CAP_RESOLUTION_HD1080 0x4000 //1920*1080
#define DVR_CAP_RESOLUTION_SD1 0x8000 //960*576
#define DVR_CAP_RESOLUTION_QXGA 0x10000 // 2048x1536
#define DVR_CAP_RESOLUTION_CR_640x360 0x20000 //640*360
```

4.2 System SDK Properties

eClientSDKAttributeType

typedef enum eClientSDKAttributeType

```
{
    eClientSDKAttributeTypeBegin = 0 //! <Starting value */
    eAttributeConnectTimeOut, //! <Connection timeout */
    eAttributeCommandTimeOut, //! <Command control timeout */
    eAttributeMediaTimeOut, //! <Media control timeout */
    eAttributeCaptureDump, //! <Client SDK exception caught */
    eClientSDKAttributeTypeEnd, //! <End value */
} EClientSDKAttributeType;
```

4.3 User Login Logout management

4.3.1 User login parameters

UserLoginPara

typedef struct tagUserLoginPara

```
{
    char sServerIP [MAX_ADDRESS_LEN]; //! <Server IP address */
    DWORD dwCommandPort; //! <Login signaling connection port */
    char sUName [USERNAME_LEN]; //! <Logged-on user name */
    char sUPass [USERPASS_LEN]; //! <Logged-on user password */
    DWORD dwReserve [DEF_RESERVE_NUM]; //! <Reserved */
} UserLoginPara, * LPUserLoginPara;
```

4.3.2 Forced to delete the logged in user callbacks:

CB_DeleteUserForce

typedef int (CALLBACK * CB_DeleteUserForce) (IN HUSER hUser, IN DWORD dwUserData);

Parameters:

hUser Users handle user login to get a handle

dwUserData User data

Comment:

Interface Type: Blocking

4.4 Event notification subscription

4.4.1 Remote Event List Type

Event Description	Event	Number
Alarm events	USEREVENT_ALARM_NOTICE	0x00000001
Heartbeat is lost, disconnected from the network	USEREVENT_HEARTBEAT_LOST	0x00000002
Successful network reconnection	USEREVENT_NET_RECOVER	0x00000004
Remote user disconnects	USEREVENT_USER_DISCONNECT	0x00000008
Remote streaming off	USEREVENT_STREAM_DISCONNECT	0x00000010
Hard disk group management event	USEREVENT_DISKGROUP_MANAGE	0x00000020
History stream event notification	USEREVENT_HISTORystream_NOTICE	0x00000040
Live stream start the connection ID notification	USEREVENT_REALSTREAM_STARTLINK	0x00000080
Real-time flow stops connection ID notification	USEREVENT_REALSTREAM_STOPLINK	0x00000100
Voice stream to initiate a connection ID notification	USEREVENT_VOICESTREAM_STARTLINK	0x00000200
Voice stream stops connection ID notification	USEREVENT_VOICESTREAM_STOPLINK	0x00000400
History stream destruction event notification	USEREVENT_HISTORystream_DESTROYLINK	0x00000800
History flows start event notification	USEREVENT_HISTORystream_STARTLINK	0x00001000
Stop the flow of history event notification	USEREVENT_HISTORystream_STOPLINK	0x00002000
History flows create an event notification	USEREVENT_HISTORystream_CREATELINK	0x00004000
AI event	USEREVENT_AI_EVENT_PUSH	0x00010000

4.4.2 Users event callback:

CB_UserEvent

typedef int (CALLBACK * CB_UserEvent) (IN HUSER hUser, IN DWORD dwEventType, DWORD dwParam1, DWORD dwParam2, DWORD dwParam3, IN DWORD dwUserData);

Parameters:

hUser Users handle user login to get a handle

eEvent Event Type

dwParam1 - dwParam3 See the following list of parameters:

Event Type	dwParam1	dwParam 2	dwParam 3
USEREVENT_REALSTREAM_STARTLINK	Connection ID	0	0
USEREVENT_REALSTREAM_STOPLINK	Connection ID	0	0
USEREVENT_ALARM_NOTICE	Alarm Type	Alarm channel	Alarm status (When video loss, 0 means no signal, a signal indicated)
USEREVENT_DISKGROUP_MANAGE	eDiskGroupOperation	Operating results (0 - success, other failures)	Hard disk to use when operating the verification code obtained by the client hard disk management commands
USEREVENT_HISTORYSTREAM_NOTICE	Indicates the type of operation: 0 - History flow break	History stream handle	Create a user data stream of history
	1 - To obtain historical stream backup data size.	Operating results (0 - success, other failures)	datasize backup data size (KB)
USEREVENT_HEARTBEAT_LOST	0	0	0
USEREVENT_NET_RECOVER	0	0	0

dwUserData User data, subscribe to events when the incoming user data

Returns:

0 - Successful

Comment:

Since this callback function receives the event thread calls,

To ensure accuracy, please minimize blocking work in the callback, in order to avoid hang.

Interface Type: Blocking

CB_UserEventEx

```
typedef int(CALLBACK *CB_UserEventEx)(IN HUSER hUser, IN DWORD
dwEventType, IN DWORD dwSubEventType, IN DWORD dwParam1, IN DWORD
dwParam2, IN DWORD dwParam3, IN const LPBuffer cBuffer, IN DWORD
dwUserData);
```

Parameters:

hUser Users handle user login to get a handle

dwEventType Main Event Type

dwSubEventType Sub Event Type

dwParam1 - dwParam3 See the following list of parameters:

Event Type	dwParam1	dwParam 2	dwParam 3
USEREVENT_REALSTREAM_STARTLINK	Connection ID	0	0
USEREVENT_REALSTREAM_STOPLINK	Connection ID	0	0
USEREVENT_ALARM_NOTICE	Alarm Type	Alarm channel	Alarm status (When video loss, 0 means no signal, a signal indicated)
USEREVENT_DISKGROUP_MANAGE	eDiskGroupOperation	Operating results (0 - success, other failures)	Hard disk to use when operating the verification code obtained by the client hard disk management commands
USEREVENT_HISTORYSTREAM_NOTICE	Indicates the type of operation: 0 - History flow break	History stream handle	Create a user data stream of history
	1 - To obtain historical stream backup data size.	Operating results (0 - success, other failures)	datasize backup data size (KB)
USEREVENT_HEARTBEAT_LOST	0	0	0
USEREVENT_NET_RECOVER	0	0	0
USEREVENT_AI_EVENT_PUSH			

dwUserData User data, subscribe to events when the incoming user data

dwEventType is USEREVENT_AI_EVENT_PUSH, dwSubEventType is eEventAIType, dwSubEventType is EVENT_AI_FACE_DETECT or EVENT_AI_FACE_RECOGNITION, buffer is EventFaceRecognition.

Returns:

0 - Successful

Comment:

Since this callback function receives the event thread calls,

To ensure accuracy, please minimize blocking work in the callback, in order to avoid hang.

Interface Type: Blocking

4.4.3 Alarm event type

typedef enum eAlertType

```
{
    ALERT_ALARMIN,           /*! <Alarm input */
    ALERT_MOTION,            /*! <Motion detection */
    ALERT_ENCODEERROR,       /*! <Encoding exception */
    ALERT_DISKERROR,         /*! <Hard disk error */
    ALERT_DISKFULL,          /*! <Hard disk is full */
    ALERT_IPCONFLICT,        /*! <IP conflict */
    ALERT_ILLEGEACCESS,      /*! <Unauthorized access */
    ALERT_RETICLEDISCONNECT, /*! <Network cable off */
    ALERT_VIDEOLOST,         /*! <Video loss */
    ALERT_VIDEOEXCEPTION,    /*! <Video exception */
    ALERT_DISKGROUP_ERROR,   /*! <Abnormal disk group */
};
```

4.5 Device configuration information

4.5.1 Remote configuration information

Configuration Information

typedef struct tagConfigInformation

```
{
    DWORD      dwMainCommand; /*! <Configuration master command word */
    DWORD      dwAssistCommand; /*! <Configurations auxiliary command word */
    char       sConfig [MAX_CFG_LEN]; /*! <Configuration information buffer */
    DWORD      dwConfigLen; /*! <Configuration information length */
    DWORD      dwReserve [DEF_RESERVE_NUM]; /*! <Reserved */
} ConfigInformation, * LPConfigInformation;
```

ConfigInformationV3

typedef struct tagConfigInformationV3

```
{
    DWORD      dwMainCommand; /*!< Configuration master command word */
    DWORD      dwAssistCommand; /*!< Configurations auxiliary command word */
    DWORD      dwConfigLen; /*!< Configuration information length */
    DWORD      dwBeginChannel; /*!< Start Channel*/
    DWORD      dwReserve[DEF_RESERVE_NUM-1]; /*!< Reserved */
};
```

```
char    sConfig[MAX_CFG_LEN_V2]; /*!< Configuration information buffer */
}ConfigInformationV3, *LPConfigInformationV3;
```

4.5.2 Configure Message Type list

Main Type	Subtype	Structure
Network configuration parameters VIDEONETCLIENT_GET_NETCFG VIDEONETCLIENT_SET_NETCFG	NETCFG_ALL	VIDEONETCLIENT_NET_CFG
	NETCFG_ETH_IF	VIDEONETCLIENT_ETH_IF
	NETCFG_DHCP_CONF	VIDEONETCLIENT_DHCP_CONF
	NETCFG_DHCP_STATE	VIDEONETCLIENT_DHCP_STATE
	NETCFG_PPPOE_CONF	VIDEONETCLIENT_PPPOE_CONF
	NETCFG_PPPOE_IF	VIDEONETCLIENT_PPPOE_IF
	NETCFG_DNS_CONF	VIDEONETCLIENT_DNS_CONF
	NETCFG_DDNS_CONF	VIDEONETCLIENT_DDNS_CONF
	NETCFG_HTTP_CONF	VIDEONETCLIENT_HTTP_CONF
	NETCFG_LISTENPORT_CONF	VIDEONETCLIENT_LISTENPORT_CONF
Server configuration parameters VIDEONETCLIENT_GET_NETSERVERCFG VIDEONETCLIENT_SET_NETSERVERCFG	NETSERVERCFG_ALL	VIDEONETCLIENT_NET_MANAGER
	NETSERVERCFG_CMS_CONF	VIDEONETCLIENT_NET_CMS
	NETSERVERCFG_CMS_STATE	VIDEONETCLIENT_CMS_STATE
	NETSERVERCFG_AMS_CONF	VIDEONETCLIENT_NET_AMS
	NETSERVERCFG_NTP_CONF	VIDEONETCLIENT_NET_NTP
	NETSERVERCFG_EML_CONF	VIDEONETCLIENT_NET_EML
Image parameters VIDEONETCLIENT_GET_PICCFG VIDEONETCLIENT_SET_PICCFG	PICCFG_ALL	VIDEONETCLIENT_PIC_CFG
	PICCFG_VIDEOSTANDARD_CONF	VIDEONETCLIENT_STANDARD
	PICCFG_OSD_CONF	VIDEONETCLIENT_OSD_CFG
	PICCFG_CHNAME_CONF	VIDEONETCLIENT_ALIAS_CFG
	PICCFG_TIMESEC_CONF	VIDEONETCLIENT_CHROMA
	PICCFG_VIDEOLOST_CONF	VIDEONETCLIENT_VILOST
	PICCFG_MOTION_CONF	VIDEONETCLIENT_MOTION
	PICCFG_MOSAIC_CONF	VIDEONETCLIENT_MOSAIC
	PICCFG_MOTION_ACTION_CONF	VIDEONETCLIENT_EVENT_ACTION
	PICCFG_VLOSTS_ACTION_CONF	VIDEONETCLIENT_EVENT_ACTION
	PICCFG_ADVANCED_CONF	VIDEONETCLIENT_ADVANCED_CHANNEL_CFG
	PICCFG_SHELTER_ACTION_CONF	VIDEONETCLIENT_EVENT_ACTION
Encoding configuration parameters	COMPRESSCFG_ALL	VIDEONETCLIENT_COMPRESSION_CFG

Network Client SDK Programming Guide Network Layer

VIDEONETCLIENT_GET_COMPRESSCFG G VIDEONETCLIENT_SET_COMPRESSCFG G	COMPRESSCFG_WORKMODE_CONF	VIDEONETCLIENT_WORKMODE
	COMPRESSCFG_COMPRESS_CAP	VIDEONETCLIENT_COMPRESS_CAP
	COMPRESSCFG_COMPRESS_CONF	VIDEONETCLIENT_COMPRESSION_CHANNEL
	VIDEONETCLIENT_SET_COMPRESSCFG	VIDEONETCLIENT_RESOLUTION_AND_FRAME_CAP_TABLE
	COMPRESSCFG_RESOLUTION_AND_BITRATE_TABLE	VIDEONETCLIENT_RESOLUTION_AND_BITRATE_CAP_TABLE
	COMPRESSCFG_VIDEO_INPUT_INFO	VIDEONETCLIENT_VI_CFG
Local recording parameters VIDEONETCLIENT_GET_RECORDCFG VIDEONETCLIENT_SET_RECORDCFG	RECORDCFG_ALL	VIDEONETCLIENT_RECORD_SCHEDULE
System time parameters VIDEONETCLIENT_GET_SYSTIME VIDEONETCLIENT_SET_SYSTIME	SYSTIME_ALL	VIDEONETCLIENT_TIME
PTZ parameters VIDEONETCLIENT_GET_PTZCFG VIDEONETCLIENT_SET_PTZCFG	PTZCFG_ALL	VIDEONETCLIENT_PTZPRO_CFG
Serial configuration parameters VIDEONETCLIENT_GET_SERIALCFG VIDEONETCLIENT_SET_SERIALCFG	SERIALCFG_ALL	VIDEONETCLIENT_DECODER_CFG
	SERIALCFG_PTZ_ELCTRONIC_CFG	VIDEONETCLIENT_PTZ_ELCTRONIC
Alarm Configuration Parameters VIDEONETCLIENT_GET_ALARMCFG VIDEONETCLIENT_SET_ALARMCFG	ALARMCFG_ALL	VIDEONETCLIENT_ALARM_CFG
	ALARMCFG_WORKMODE_CONF	VIDEONETCLIENT_WORKMODE
	ALARMCFG_IMGCAPTURE_CONF	VIDEONETCLIENT_ALARM_CAPTURE_PICTURE
	ALARMCFG_ALARMIN_CONF	VIDEONETCLIENT_ALARMIN_CHANNEL
	ALARMCFG_ALARMOUT_CONF	VIDEONETCLIENT_ALARMOUT_CHANNEL
	ALARMCFG_EXCEPTION_CONF	VIDEONETCLIENT_SYSTEM_EXCEPTION
	VIDEONETCLIENT_GET_ALARMCFG	VIDEONETCLIENT_EVENT_ACTION
Display output configuration parameters VIDEONETCLIENT_GET_VIDEOOUTCFG G VIDEONETCLIENT_SET_VIDEOOUTCFG G	VIDEOOUTCFG_ALL	VIDEONETCLIENT_VO_CFG
User Configuration VIDEONETCLIENT_GET_USERCFG	USERCFG_ALL	VIDEONETCLIENT_USER_CFG

Network Client SDK Programming Guide Network Layer

VIDEONETCLIENT_SET_USERCFG		
	USERCFG_ONE	ZW_USER_BASIC_CFG
	USERCFG_CREATE	ZW_USER_BASIC_CFG
	USERCFG_REMOVE	ZW_USER_BASIC_CFG
Device Configuration VIDEONETCLIENT_GET_DEVICEINFO VIDEONETCLIENT_SET_DEVICEINFO (Read only not used)	DEVICEINFO_ALL	VIDEONETCLIENT_DEVICE_INFO
Device configuration parameters VIDEONETCLIENT_GET_DEVICECFG VIDEONETCLIENT_SET_DEVICECFG	DEVICECFG_ALL	VIDEONETCLIENT_DEVICE_CFG
	DEVICECFG_LOCKSCREENTIME_CON	---
	DEVICECFG_LANGUAGE_CONF	---
	DEVICECFG_DATETIME_CONF	---
	DEVICECFG_DST_CONF	VIDEONETCLIENT_DST
	DEVICECFG_REMOTECONTROL_ID	---
	DEVICECFG_TIMEZONE_CONF	VIDEONETCLIENT_TIMEZONE
	DEVICECFG_VIDEOSTANDARD_CONF	VIDEONETCLIENT_STANDARD
Store information VIDEONETCLIENT_GET_STORAGEINFO VIDEONETCLIENT_SET_STORAGEINFO (Read only not used)	STORAGEINFO_ALL	VIDEONETCLIENT_STORAGE_CFG
	STORAGEINFO_WORKMODE_CON	VIDEONETCLIENT_WORKMODE
	STORAGEINFO_DISK_INFO	VIDEONETCLIENT_DISK
	STORAGEINFO_DISKGROUP_IN	VIDEONETCLIENT_DISK_GROUP
	STORAGEINFO_DISK_MARK	VIDEONETCLIENT_DISK_MARK
Device status information VIDEONETCLIENT_GET_DEVICESTAT	DEVICESTATE_ALL	VIDEONETCLIENT_DEVICE_STATE

Network Client SDK Programming Guide Network Layer

E VIDEONETCLIENT_SET_DEVICESTAT E		
Equipment maintenance configuration VIDEONETCLIENT_GET_DEVICEMAIN TENANCE VIDEONETCLIENT_SET_DEVICEMAIN TENANCE	DEVICEMAINTENANCE_ALL	VIDEONETCLIENT_DEVICE_MAINTENANCE
Equipment personality VIDEONETCLIENT_GET_DEVICECUSTOM OM VIDEONETCLIENT_SET_DEVICECUSTOM OM	DEVICECUSTOM_ALL	VIDEONETCLIENT_DEVICE_CUSTOM
NXP8850 ISP parameters VIDEONETCLIENT_GET_NXP8850ISP VIDEONETCLIENT_SET_NXP8850ISP	NXP8850ISP_ALL	VIDEONETCLIENT_NXP8850ISP_CONFIG
Nvr ISP parameters HY_DVR_NVR_GET_ISPCFG HY_DVR_NVR_SET_ISPCFG	NVR_ISPCFG_COLOR	ZW_NVR_COLOR
	NVR_ISPCFG_EXPOSURE	ZW_NVR_EXPOSURE
	NVR_ISPCFG_WHITEBLANCE	ZW_NVR_WHITEBLANCE
	NVR_ISPCFG_DAY_NIGHT	ZW_NVR_DAY_NIGHT
	NVR_ISPCFG_BASIC	ZW_NVR_BASIC
	NVR_ISPCFG_DEFAULT	ZW_NVR_ISP_CONFIG
	NVR_ISPCFG_ALL	ZW_NVR_ISP_CONFIG
List parameters of connected equipment (NVR) VIDEONETCLIENT_GET_DEVICES_CONNECT_LIST VIDEONETCLIENT_SET_DEVICES_CONNECT_LIST	DEVICES_CONNECT_LIST_ALL	VIDEONETCLIENT_FRONT_DEVICE_LIST
Network configuration of connected equipment (NVR) VIDEONETCLIENT_GET_DEVICE_NETWORK VIDEONETCLIENT_SET_DEVICE_NETWORK	DEVICE_NETWORK_ALL	VIDEONETCLIENT_FRONT_DEVICE_NETWORK
Network state of connected equipment (NVR) VIDEONETCLIENT_GET_DEVICE_STATE VIDEONETCLIENT_SET_DEVICE_STATE	DEVICE_STATE_ALL	VIDEONETCLIENT_FRONT_DEVICE_STATE_LIST
Extensional configuration of equipment capacity VIDEONETCLIENT_GET_DEVICEINFO_EX VIDEONETCLIENT_SET_DEVICEINFO_EX	DEVICEINFO_EX_ALL	VIDEONETCLIENT_DEVICE_INFO_EX

Extensional configuration of ISP VIDEONETCLIENT_COM_GET_ISPCFG_EX VIDEONETCLIENT_COM_SET_ISPCFG_EX	COM_ISPCFG_EX_ALL	ZW_COM_ISP_EX
Preview List Configure (IPD) VIDEONETCLIENT_GET_PREVIEW_LIST VIDEONETCLIENT_SET_PREVIEW_LIST	PREVIEW_LIST_CFG_ALL	VIDEONETCLIENT_PREVIEW_LIST
Zoom in or zoom out Preview screen VIDEONETCLIENT_FULLSCREEN_WINDOW VIDEONETCLIENT_RETURN_FROM_FULLSCREEN	CONTROL_PREVIEW_ALL	VIDEONETCLIENT_FULLSCREEN_CHANNEL

4.6 Live Stream

4.6.1 Streaming mode

```
eStreamTransferMode
typedef enum eStreamTransferMode
{
    eStreamTransferModeBegin    =    0    /*! <Starting value */
    eGeneralTCP,                /*! <Ordinary TCP */
    eGeneralRTP,                /*! <Ordinary RTP, not supported */
    eStreamTransferModeEnd,      /*! <End value */
} EStreamTransferMode;
```

4.6.2 Streaming media types

```
eStreamMediaType
typedef enum eStreamMediaType
{
    eStreamMediaTypeBegin    =    0    /*! <Starting value */
    eMainVideoAndSound,      /*! <Main stream audio and video */
    eMainVideo,              /*! <Main stream video */
    eMainSound,              /*! <Main stream audio */
    eAssistVideo,            /*! <Sub-stream video */
    eStreamMediaTypeEnd,
    /*! <End value */
} EStreamMediaType;
```

4.6.3 Live stream connection parameters

```
RealStreamPara
typedef struct tagRealStreamPara
{
    DWORD        dwChannel;    /*! <Streaming channel number */
    eStreamTransferMode    eTransferMode;    /*! <Transfer mode */
    eStreamMediaType    eMediaType;    /*! <Streaming media types */
    DWORD        dwReserve [DEF_RESERVE_NUM];    /*! <Reserved */
} RealStreamPara, *LPRealStreamPara;
```

4.6.4 Network Frame Header

StreamMediaFrame

```
typedef struct tagStreamMediaFrame
```

```
{
    DWORD    dwChannel;           /*!< Frame number          */
    Buffer    cFrameBuffer;       /*!< Frame buffer          */
    AbsoluteTime    cFrameTime;   /*!< Absolute timestamp   */
    DWORD    dwFrameType;        /*!< Frame Type( refer to eFrameType)*/
    DWORD    dwReserve[DEF_RESERVE_NUM]; /*!< Reserve*/
}StreamMediaFrame, *LPStreamMediaFrame;
```

4.6.5 Streaming data callback

CB_StreamMedia

```
typedef int (CALLBACK * CB_StreamMedia) (IN HSTREAM hStream, IN const
StreamMediaFrame * cFrame, IN DWORD dwUserData);
```

Parameters:

hStream Streaming handle the flow to get through the connection, which means that streaming data sources

cFrame Streaming media data, including a frame buffer, the frame time, the frame type information

dwUserData User data set when the media data stream callback incoming user data

Returns:

0 - Successful

Comment:

Since this callback function is called streaming data reception thread,

In order to ensure real-time, please minimize blocking work in the callback to avoid dropped frames.

Interface Type: Blocking

4.7 History Flow

4.7.1 History stream video type

```
/** @ Brief common video types          */
#define GENERAL_RECORD                    0 x 00000001
/** @ Brief manual recording type */
#define MANUAL_RECORD                    0 x 00000002
/** @ Brief mobile video type          */
#define ALARM_RECORD                    0 x 00000004
/** @ Brief alarm recording type        */
#define MOTION_RECORD                    0 x 00000008
/** @ Brief all video types            */
#define ALL_STREAM_MEDIA (GENERAL_RECORD | MANUAL_RECORD | MOTION_RECORD |
ALARM_RECORD)
```

4.7.2 History query stream

HistoryStreamQueryFactor

typedef struct tagHistoryStreamQueryFactor

```
{
    DWORD      dwChannel;          /*! <Channel number          */
    DWORD      dwDiskGroup;        /*! <Disk group              */
    DWORD      eStreamType;        /* <Historical stream type, or use multiple types! ()
Combination */
    char       cBeginTime [15];    /*! <Start time */
    char       cEndTime [15];     /*! <End Time */
} HistoryStreamQueryFactor, * LPHistoryStreamQueryFactor;
```

4.7.3 History stream query result

HistoryStreamQueryResult

typedef struct tagHistoryStreamQueryResult

```
{
    DWORD      dwChannel;          /*! <Channel number          */
    DWORD      eStreamType;        /* <Historical stream type, or use multiple types! ()
Combination */
    char       cBeginTime [DATE_TIME_LEN]; /*! <Start time          */
    char       cEndTime [DATE_TIME_LEN];   /*! <End time */
    DWORD      dwStreamSize;        /*! <Stream data length      */
} HistoryStreamQueryResult, * LPHistoryStreamQueryResult;
/*!
```

4.7.4 History stream connection parameters

HistoryStreamPara

typedef struct tagHistoryStreamPara

```
{
    DWORD      dwDiskGroup;        /*! <Disc set */
    DWORD      dwChannel;          /*! <Streaming channel number */
    DWORD      dwEnableEndTime;    /*! <End time is valid */
    TimeInfo   cBeginTime;         /*! <Historical stream start time */
    TimeInfo   cEndTime;          /*! <Historical stream end time */
    DWORD      eStreamType;        /* <Historical stream type, or use multiple types! ()
Combination */
    eStreamTransferMode eTransferMode; /*! <Transfer mode */
    DWORD      dwReserve [DEF_RESERVE_NUM]; /*! <Reserved */
} HistoryStreamPara, * LPHistoryStreamPara;
```

4.7.5 Historical flow data callback

CB_StreamMedia

typedef int (CALLBACK * CB_StreamMedia) (IN HSTREAM hStream, IN const StreamMediaFrame * cFrame, IN DWORD dwUserData);

Parameters:

hStream Streaming handle the flow to get through the connection, which means that streaming data sources

cFrame Streaming media data, including the frame buffer, frame time, frame type information; on frame type, need special attention or treatment of the following type:

Frame Type	Number	Description
------------	--------	-------------

HISTORY_STREAM_EXCEPTION	0x01000000	The end of history flow anomalies
HISTORY_STREAM_SWITCH_FRAME	0x02000000	History stream switching frames
HISTORY_STREAM_TIME_JUMP	0x03000000	Historical flow jumps to the new time slot
HISTORY_STREAM_TIME_END	0x04000000	The end of history flow frame
HISTORY_STREAM_TIME_CURR	0x05000000	Historical flow data query is completed

dwUserData User data set when the media data stream callback incoming user data

Returns:

0 - Successful

Comment:

Since this callback function is called streaming data reception thread,

In order to ensure real-time, please minimize blocking work in the callback to avoid dropped frames.

Interface Type: Blocking

4.7.6 History positioning operation flow direction enumeration

eOperationType

```
typedef enum eOperationType
```

```
{
    eOperationTypeBegin      =      0      /*! <Starting value */
    eOperationGet,           /*! <Get operation */
    eOperationSet,           /*! <Set operation */
    eOperationTypeEnd,       /*! <End value */
} EOperationType;
```

4.8 Voice Intercom

4.8.1 Voice stream connection parameters

VoiceStreamPara

```
typedef struct tagVoiceStreamPara
```

```
{
    DWORD          dwChannel;          /*! <Streaming channel number */
    eVoiceMode eMode;                  /*! <Speech codec mode */
    eStreamTransferMode eTransferMode; /*! <Transfer mode */
    DWORD dwReserve [DEF_RESERVE_NUM]; /*! <Reserved */
} VoiceStreamPara, * LPVoiceStreamPara;
```

4.8.2 Voice stream data callback

CB_StreamVoice

```
typedef int (CALLBACK * CB_StreamVoice) (IN HSTREAM hStream, IN eVoiceStreamSource
eSource, IN const StreamVoiceFrame * cFrame, IN DWORD dwUserData);
```

Parameters:

hStream Handle voice traffic, by connecting the flow capture, said streaming media data sources

eSource Voice stream data sources, including capturing or receiving data
cFrame Voice data stream, including buffer, the frame information sources
dwUserData User data, voice stream data set when the incoming user data callback

Returns:

0 - Successful

Comment:

Since this callback function receives streaming data and voice capture thread calls,

In order to ensure real-time, please minimize blocking work in the callback to avoid dropped frames.

Interface Type: Blocking

4.9 Log Query

4.9.1 Log query

LogQueryFactor

typedef struct tagLogQueryFactor

```
{
    DWORD          m_dwQueryMode;           /*! <Query */
    eHistoryLogMajorType m_eMajorType;       /*! <Main type */
    eHistoryLogMinorType m_eMinorType;       /*! <Subtype */
    DWORD          m_dwChannel;             /*! <Channel number */
    char           m_sStartTime [DATE_TIME_LEN]; /*! <start time */
    char           m_sStopTime [DATE_TIME_LEN]; /*! <end time */
} LogQueryFactor, * LPLogQueryFactor;
```

4.9.2 History query results log

LogQueryResult

typedef struct tagLogQueryResult

```
{
    eHistoryLogMajorType m_eMajorType; /*! <Main type */
    eHistoryLogMinorType m_eMinorType; /*! <Subtype */
    DWORD               m_dwDetailInfo; /*! <Details */
    char                m_sUserName [USERNAME_LEN]; /*! <Operation of the user */
    char                m_sUserIP [IP_ADDRESS_LEN]; /*! <User IP address */
    char                m_sLogTime [DATE_TIME_LEN]; /*! <Operating time */
} LogQueryResult, * LPLogQueryResult;
```

4.10 Clear Channel

4.10.1 Serial Type

eSerialType

typedef enum eSerialType

```
{
    eTTY232= 0, /*! <RS232 serial port */
    eTTY485, /*! <RS485 serial port */
    eTTY422, /*! <RS422 serial port */
} ESerialType;
```

4.10.2 Clear Channel Parameters

```
TransparentChannelPara
typedef struct tagTransparentChannelPara
{
    DWORD   dwMajorType;          /* <Main types of information -! ESerialType */
    DWORD   dwMinorType; /* <Subtype information -! Video channel number (e.g., 0, 1,
2 ...), corresponding to the configuration of the channel using the baud rate */
} TransparentChannelPara, * LPTransparentChannelPara;
```

4.10.3 Transparent channel data callback

```
CB_TransparentChannel
typedef int (CALLBACK * CB_TransparentChannel) (IN HTRANSPARENT hTransparent, IN
const LPBuffer pBuffer, IN DWORD dwUserData);
```

Parameters:

hTransparent Transparent channel handle, transparent channel connection to get through,
which means that the source of the data

pBuffer Transparent channel data, including data buffer length

dwUserData User data, channel data is set transparent user data callback,

Returns:

0 - Successful

Comment:

Since this callback function receives data in a transparent channel thread calls,

In order to ensure real-time, please minimize blocking work in the callback.

Interface Type: Blocking

4.11 PTZ control

4.11.1 PTZ PTZ control code type

ePTZControlCmdCode

Edit Number	Control Code Type and Description	Meaning corresponding parameters			
		Parameter 1	Parameter 2	Parameter 3	Parameter 4
0	ePTZControlCodeAllOff (Off (or disconnect) all the switches)	Invalid	Invalid	Invalid	Invalid
1	ePTZControlCodeCameraPower (Turn on the camcorder)	1 - Start 0 - Stop	Invalid	Invalid	Invalid
2	ePTZControlCodeLightPower (Turning lights Power)	1 - Start 0 - Stop	Invalid	Invalid	Invalid
3	ePTZControlCodeWiper (Wiper switch is turned on)	1 - Start 0 - Stop	Invalid	Invalid	Invalid
4	ePTZControlCodeFans	1 -	Invalid	Invalid	Invalid

	(Turn the fan switch)	Start 0 - Stop			
5	ePTZControlCodeHeater (Heater switch is turned on)	1 Start 0 - Stop	Invalid	Invalid	Invalid
6	ePTZControlCodeAuxEquipment (Auxiliary equipment switch is turned on)	1 Start 0 - Stop	Invalid	Invalid	Invalid
10	ePTZControlCodeStopContinue (Stop all continuous (lenses, PTZ) action)	Invalid	Invalid	Invalid	Invalid
11	ePTZControlCodeZoomIn (Focal length becomes large (larger magnification))	1 Start 0 - Stop	Speed	Invalid	Invalid
12	ePTZControlCodeZoomOut (Focal length becomes smaller (smaller magnification))	1 Start 0 - Stop	Speed	Invalid	Invalid
13	ePTZControlCodeFocusNear (Focus before the transfer)	1 Start 0 - Stop	Speed	Invalid	Invalid
14	ePTZControlCodeFocusFar (Back focus adjustment)	1 Start 0 - Stop	Speed	Invalid	Invalid
15	ePTZControlCodeApertureUp (Aperture expand)	1 Start 0 - Stop	Speed	Invalid	Invalid
16	ePTZControlCodeApertureDown (Narrow aperture)	1 Start 0 - Stop	Speed	Invalid	Invalid
17	ePTZControlCodeAutoZoom (Open Auto Focus (automatic magnification))	1 Start 0 - Stop	Invalid	Invalid	Invalid
18	ePTZControlCodeAutoFocus (On Auto Focus)	1 Start 0 - Stop	Invalid	Invalid	Invalid
19	ePTZControlCodeAutoAperture (Open auto iris)	1 Start 0 - Stop	Invalid	Invalid	Invalid

21	ePTZControlCodeUp (PTZ pitching)	1 - Start 0 - Stop	Speed	Invalid	Invalid
22	ePTZControlCodeDown (PTZ next stoop)	1 - Start 0 - Stop	Speed	Invalid	Invalid
23	ePTZControlCodeLeft (PTZ turn left)	1 - Start 0 - Stop	Speed	Invalid	Invalid
24	ePTZControlCodeRight (PTZ turn right)	1 - Start 0 - Stop	Speed	Invalid	Invalid
25	ePTZControlCodeUpLeft (Yang and turn left on the pan)	1 - Start 0 - Stop	Speed	Invalid	Invalid
26	ePTZControlCodeUpRight (Yang and turn right on the head)	1 - Start 0 - Stop	Speed	Invalid	Invalid
27	ePTZControlCodeDownLeft (PTZ stoop and turn left)	1 - Start 0 - Stop	Speed	Invalid	Invalid
28	ePTZControlCodeDownRight (PTZ stoop and turn right)	1 - Start 0 - Stop	Speed	Invalid	Invalid
29	ePTZControlCodeAutoLeftRight (About PTZ auto scan)	1 - Start 0 - Stop	Speed	Invalid	Invalid
40	ePTZControlCodePresetSet (Set preset)	Preset Point No. [> = 0]	Invalid	Invalid	Invalid
41	ePTZControlCodePresetClear (Clear Preset)	Preset Point No. [> = 0]	Invalid	Invalid	Invalid
42	ePTZControlCodePresetCall (Go to the preset point)	Preset Point No. [> = 0]	Invalid	Invalid	Invalid
51	ePTZControlCodeCuriserSetStart (Start Cruising memory)	Cruise No.	Invalid	Invalid	Invalid

		[> = 0]			
52	ePTZControlCodeCuriserSetStop (Close cruise memories)	Cruise No. [> = 0]	Invalid	Invalid	Invalid
53	ePTZControlCodeCuriserAddPreset (The preset join cruise sequence)	Cruise No. [> = 0]	Preset Point No. [> = 0]	Pause Time [S> = 0]	Cruise Speed [1-10]
54	ePTZControlCodeCuriserRunStart (Start cruising)	Cruise No. [> = 0]	Invalid	Invalid	Invalid
55	ePTZControlCodeCuriserRunStop (Stop Cruise)	Cruise No. [> = 0]	Invalid	Invalid	Invalid
61	ePTZControlCodeTraceSetStart (Start trace memory)	Track number [> = 0]	Invalid	Invalid	Invalid
62	ePTZControlCodeTraceSetStop (Off track memory)	Track number [> = 0]	Invalid	Invalid	Invalid
63	ePTZControlCodeTraceRunStart (Start track)	Track number [> = 0]	Invalid	Invalid	Invalid
64	ePTZControlCodeTraceRunStop (Stop track)	Track number [> = 0]	Invalid	Invalid	Invalid
99	ePTZControlCodeSystemReset (System reset)	Invalid	Invalid	Invalid	Invalid

4.12 File Query

4.12.1 Document types of queries

eFileQueryType

```
typedef enum eFileQueryType
```

```
{
```

```
    eFileTypeBegin      =      0
```

```
    eUpgrade,
```

```
    eImage,
```

```
    ePtzProtocolFile,
```

```
    eFileTypeEnd,
```

```
} EFileQueryType;
```

```
    /*! <Starting value          */
```

```
    /*! <Upgrade package        */
```

```
    /*! <Image file */
```

```
    /*! <PTZ protocol */
```

```
    /*! <End value              */
```

4.12.2 File query

```
FileQueryFactor
typedef struct tagFileQueryFactor
{
    DWORD    dwChannel;           /*! <Channel number */
    DWORD    dwFileType;         /*! <File type */
    char cBeginTime [DATE_TIME_LEN]; /*! <Start time */
    char cEndTime [DATE_TIME_LEN]; /*! <End time */
} FileQueryFactor, * LPFileQueryFactor;
```

4.12.3 File search results

```
FileQueryResult
typedef struct tagFileQueryResult
{
    DWORD    dwChannel;           /*! <Channel number */
    DWORD    dwLock;             /*! <Whether to lock */
    DWORD    dwDataSize;         /*! <Data size */
    DWORD    dwFileType;         /*! <Image type */
    char      cCreateTime [DATE_TIME_LEN]; /*! <Picture time */
    char      cFileName [MAX_FILENAME]; /*! <Pictures device-side path */
} FileQueryResult, * LPFileQueryResult;
```

4.13 File upload and download

4.13.1 Transfer the file type

```
eTransferFileType
typedef enum eTransferFileType
{
    eTransferFileTypeBegin    =    0    /*! <Starting value */
    eFileSystemUpdate,        /*! <Upgrade package file */
    ePtzProtocol,             /*! <PTZ protocol file */
    eDeviceConfig,           /*! <Device configuration file */
    eTransferFileTypeEnd,     /*! <End value */
} ETransferFileType;
```

4.13.2 File Transfer Control

```
eFileTransferControl
typedef enum eFileTransferControl
{
    eFileTransferControlBegin    =    0    /*! <Starting value */
    eFileTransferStart,         /*! <Transfusion startup file */
    eFileTransferStop,          /*! <Stop file transfer */
    eFileTransferControlend,     /*! <Starting value */
} EFileTransferControl;
```

4.13.3 File upload parameters

```
FileUploadPara
typedef struct tagFileUploadPara
```

```
{
    eTransferFileType    eFileType;           /*! <Uploaded file type */
    char  strRemoteFilePath [MAX_FILE_PATH_LEN]; /*! <Uploaded file path */
    char  strLocalFilePath [MAX_FILE_PATH_LEN]; /*! <Uploaded file path */
} FileUploadPara, * LPFileUploadPara;
```

4.13.4 File upload status

```
FileUploadState
typedef struct tagFileUploadState
{
    DWORD          dwUploadSize; /*! <Uploading data length */
    DWORD          dwStatus;     /*! <Upload status,
                                0 is being sent,
                                1 to cancel,
                                2 full,
                                3 is the wrong version or file errors,
                                4 indicating a write failure,
                                5 indicates successful completion,
                                6 said transmission fails,
                                7 represents Error reading from file */
    DWORD          dwReserve [DEF_RESERVE_NUM]; /*! <Reserved */
} FileUploadState, * LPFileUploadState;
```

4.13.5 File upload data callback

```
CB_FileUpload
typedef int (CALLBACK * CB_FileUpload) (IN HFILE_TRANSFER hFileTransfer, IN const
FileUploadState cState, IN DWORD dwUserData);
```

Parameters:

hFileTransfer Handles file upload, file upload to get through the connection
cState File upload status, including data transmission length
dwUserData User data set when the data callback incoming user data

Returns:

0 - Successful

Comment:

Since this callback function in the file upload data transmission thread calls,
In order to ensure real-time, please minimize blocking work in the callback.
Interface Type: Blocking

4.13.6 Download parameters

```
FileDownloadPara
typedef struct tagFileDownloadPara
{
    eTransferFileType    eFileType;           /*! "Download file types */
    char strRemoteFilePath [MAX_FILE_PATH_LEN]; /*! <Source file storage path */
    char  strLocalFilePath [MAX_FILE_PATH_LEN]; /*! <Target file storage path */
} FileDownloadPara, * LPFileDownloadPara;
```

4.13.7 Download data callback

CB_FileDownload

typedef int (CALLBACK * CB_FileDownload) (IN HFILE_TRANSFER hFileTransfer, IN const Buffer cBuffer, IN DWORD dwUserData);

Parameters:

hFileTransfer Download handle, download files obtained through connections

cBuffer Download file data, including data buffer length

dwUserData User data set when the data callback incoming user data

Returns:

0 - Successful

Comment:

Since this callback function receives the data in the file download thread calls,

In order to ensure real-time, please minimize blocking work in the callback.

When cBuffer length is 0xFFFFFFFF, the buffer pointer is NULL, the download error, should be disconnected deal

Interface Type: Blocking

4.14 Remote Control

4.14.1 Remote device control

eRemoteDeviceControl

typedef enum eRemoteDeviceControl

```
{
    eRemoteDeviceControlBegin = 0,    /*! <Starting value */
    eDeviceReboot,                    /*! <Restart */
    eDeviceHalt,                      /*! <Shutdown */
    eDeviceStandby,                   /*! <Standby */
    eDeviceSetDefault,                /*! <Restore the system default */
    eDeviceSendTestEmail,             /*!< send a test E-mail */
    eDeviceSetDefense,                 /*!< Set Defense */
    eDeviceUnsetDefense,               /*!< Unset defense */
    eDeviceDefaultISPCfg,             /*!< store default ISP configuration */
    eDeviceAdminDefaultPassword,      /*!< store default password of admin*/
    eDeviceDetectBadPoint,            /*!< check bad point */
    eDeviceStartDetect,               /*!< start to detect*/
    eRemotePreviewCruiseStart = 20,  /*!< start preview cruise */
    eRemotePreviewCruiseStop,         /*!< stop preview cruise*/
    eRemoteDeviceControlEnd = 30,     /*! <End value */
} ERemoteDeviceControl;
```

4.14.2 Disk group type

eDiskGroupType

typedef enum eDiskGroupType

```
{
    eDiskGroupTypeBegin = 0,          /*! <Starting value */
    eDiskGroupNormal,                 /*! <Ordinary disk group */
    eDiskGroupAlarm,                  /*! <Alarm disc set */
    eDiskGroupRedundance,             /*! <Redundancy disk group */
    eDiskGroupBackup,                 /*! <Backup disk group */
}
```

```
        eDiskGroupTypeEnd,                /*! <End value          */
    } EDiskGroupType;
```

4.14.3 Storage Management

eDiskGroupOperation

typedef enum eDiskGroupOperation

```
{
    eDiskGroupOperationBegin = 0,        /*! <Starting value */
    eUnmountDisk,                        /*! <Unmount the partition */
    eCreateNewPartition,                 /*! "Create a new partition */
    eDeletePartition,                    /*! <Deleted partition */
    eFormatPartition,                    /*! <Formatted partition */
    eDiskGroupKeepTime,                  /*! <Time to keep the data packet */
    eDiskGroupBindChannel,                /*! <Grouping bound channel */
    eDiskGroupAddPartition,               /*! <Add a partition to the group */
    eDiskGroupDelPartition,               /*! <Delete partitions from the packet */
    eDiskGroupOperationEnd,
    /*! <End value */
} EDiskGroupOperation;
```

4.14.4 File lock operation

eFileLockOperationCode

typedef enum eFileLockOperationCode

```
{
    eUnlockFile    =    0    /*! <Unlock          */
    eLockFile,      /*! <Unlock          */
} EFileLockOperationCode;
```

4.14.5 Grab

eImageFormat (file format)

typedef enum eImageFormat

```
{
    eImageFormatBegin=0,                /*! <Starting value          */
    eBmpFormat,                          /*! <Bitmap format          */
    eJpgFormat,                          /*! <Jpg format            */
    eImageFormatEnd,                     /*! <End value              */
} EImageFormat;
```

ImageFileInfo (file info)

typedef struct tagImageFileInfo

```
{
    DWORD  m_dwFileSize;                  /*! <File size */
    char    m_sFileName [MAX_FILENAME];   /*! <Filename */
    char    m_sCreateTime [DATE_TIME_LEN]; /*! <File creation time */
} ImageFileInfo, * LPImageFileInfo;
```

4.15 Device Registration

4.15.1 Device Registration Information

DeviceRegisterInfo

typedef struct tagDeviceRegisterInfo

```
{
    DWORD    dwDeviceID;                /*! <Product serial number */
    char cDeviceIP [IP_ADDRESS_LEN];    /*! <Device IP address */
    char cDeviceMAC [MAC_ADDRESS_LEN];  /*! <Device MAC address */
    DWORD    dwDeviceCmdPort;           /*! <Device command port */
    DWORD    dwHTTPPort;                /*! <HTTP port */
    DWORD    dwDeviceType;              /*! <Device type */
    char cDeviceVersion [VERSION_INFO_LEN]; /*! <Device software version number */
    DWORD    dwDeviceMaxConnect;        /*! <Video maximum number of connections */
} DeviceRegisterInfo, * LPDeviceRegisterInfo;
```

4.15.2 Device Registration callback

CB_DeviceRegister

typedef int (CALLBACK * CB_DeviceRegister) (IN LPDeviceRegisterInfo pDeviceRegister);

Parameters:

pDeviceRegister Device Registration Information

Comment:

Interface Type: Blocking

4.16 Streaming media control

4.16.1 Mission Control

eTaskControl

typedef enum eTaskControl

```
{
    eTaskControlBegin    =    0    /*! <Starting value */
    eTaskStart,          /*! <Task starts */
    eTaskStop,           /*! <Task to stop */
    eTaskSetSpeed,       /*! <Quick release when the flow of history, set the
fast-motion */
    eTaskControlEnd,     /*! <End value */
} ETaskControl;
```

4.16.2 Media Function Type

eMediaFunctionType

typedef enum eMediaFunctionType

```
{
    eMediaFunctionTypeBegin    =    0    /*! <Starting value */
    eMediaFunctionRealStream,  /*! <Live Stream */
    eMediaFunctionVoiceStream, /*! <Voice stream */
    eMediaFunctionHistoryStream, /*! <History stream */
    eMediaFunctionTransparent, /*! <Transparent channel */
}
```



```
eMediaFunctionFileUpload,      /*! <File upload      */
eMediaFunctionFileDownload,    /*! <File download */
eMediaFunctionTypeEnd,         /*! <End value      */
} EMediaFunctionType;
```

4.16.3 Query results streaming connection ID

```
MediaLinkIDQueryResult
typedef struct tagMediaLinkIDQueryResult
{
    DWORD dwNumber; /*! <Valid number */
    DWORD dwLinkID [MAX_LINK_ID_QUERY_NUM]; /*! <Connection ID */
} MediaLinkIDQueryResult, * LPMediaLinkIDQueryResult;
```

4.16.4 Streaming data callback

```
CB_StreamMedia
typedef int (CALLBACK * CB_StreamMedia) (IN HSTREAM hStream, IN const
StreamMediaFrame * cFrame, IN DWORD dwUserData);
```

Parameters:

hStream Streaming handle the flow to get through the connection, which means that streaming data sources

cFrame Streaming media data, including a frame buffer, the frame time, the frame type information

dwUserData User data set when the media data stream callback incoming user data

Returns:

0 - Successful

Comment:

Since this callback function is called streaming data reception thread,

In order to ensure real-time, please minimize blocking work in the callback to avoid dropped frames.

Interface Type: Blocking

4.17 Font information parameters

The contents of the operating system parameters similar font parameters

FontPara

```
typedef struct tagFontPara
{
    LONG lfHeight; /*! <Character logical unit height */
    LONG lfWidth; /*! <Character logical unit width */
    LONG lfEscapement; /*! <When each line of text output angle relative to the bottom
of the page */
    LONG lfOrientation; /*! <Character baseline angle relative to the bottom of the
page */
    LONG lfWeight; /*! <Font weight, font-generation refers to the degree of thickness*/
    BYTE lfItalic; /*! <Whether to use italics */
    BYTE lfUnderline; /*! <Whether to use an underscore */
    BYTE lfStrikeOut; /*! <Whether to use strikethrough */
    BYTE lfCharSet; /*! <Character set */
    BYTE lfOutPrecision; /*! <Output accuracy */
    BYTE lfClipPrecision; /*! <Clip accuracy */
    BYTE lfQuality; /*! <Output quality */
}
```

```

        BYTE lfPitchAndFamily;           / *! <Font character spacing and
family                                */
        char lfFaceName [MAX_FONT_NAME_LEN];           / *! <Font
name                                */
    } FontPara, * LPFontPara;

```

4.18 Remote panel status information

```

PanelStatusInfo
typedef struct tagPanelStatusInfo
{
    DWORD dwStatus;           / *! <Panel status, 0 is not locked, 1 is locked */
    DWORD dwReserve [DEF_RESERVE_NUM]; / *! <Reserved */
} PanelStatusInfo, * LPPanelStatusInfo;

```

4.19 Remote panel control parameters

```

PanelControlParameter
typedef struct tagPanelControlParameter
{
    DWORD dwControl;           / *! <Panel controls, see ePanelControlType value */
    DWORD dwKeyBoardCode;       / *! <Panel keys, see ePanelKeyBoardCodeType value */
    DWORD dwReserve [DEF_RESERVE_NUM]; / *! <Reserved */
} PanelControlParameter, * LPPanelControlParameter;

```

```

typedef enum ePanelControlType
{
    ePanelControlBegin           = 0,           / *! <Starting value */
    ePanelControlKeyDown, / *! <Press */
    ePanelControlKeyUp,         / *! <Bounce */
    ePanelControlClick,         / *! <Click (press and pop) */
    ePanelControlEnd,           / *! <End value */
} EPanelControlType;

```

```

typedef enum ePanelKeyBoardCodeType
{
    ePanelKeyBoardTypeBegin      = 0x0,         / *! <Starting value */
    ePanelKeyBoardCode_0         = 0x1,         / *! <0 */
    ePanelKeyBoardCode_1         = 0x2,         / *! <1 */
    ePanelKeyBoardCode_2         = 0x3,         / *! <2 */
    ePanelKeyBoardCode_3         = 0x4,         / *! <3 */
    ePanelKeyBoardCode_4         = 0x5,         / *! <4 */
    ePanelKeyBoardCode_5         = 0x6,         / *! <5 */
    ePanelKeyBoardCode_6         = 0x7,         / *! <6 */
    ePanelKeyBoardCode_7         = 0x8,         / *! <7 */
    ePanelKeyBoardCode_8         = 0x9,         / *! <8 */
    ePanelKeyBoardCode_9         = 0xA,         / *! <9 */
    ePanelKeyBoardCode_10        = 0xB,         / *! <10 */
    ePanelKeyBoardCode_11        = 0xC,         / *! <11 */
    ePanelKeyBoardCode_12        = 0xD,         / *! <12 */
    ePanelKeyBoardCode_13        = 0xE,         / *! <13 */
    ePanelKeyBoardCode_14        = 0xF,         / *! <14 */
    ePanelKeyBoardCode_15        = 0x10,        / *! <15 */
    ePanelKeyBoardCode_16        = 0x11,        / *! <16 */
    ePanelKeyBoardCode_PTZ       = 0x40,        / *! <PTZ */
    ePanelKeyBoardCode_Copy      = 0x41,        / *! <Backup */
}

```

```

ePanelKeyBoardCode_Multi = 0x42,
ePanelKeyBoardCode_Switch = 0x43,
ePanelKeyBoardCode_Function= 0x44,
ePanelKeyBoardCode_Play = 0x45,
ePanelKeyBoardCode_Backward= 0x46,
ePanelKeyBoardCode_Record=0x47,
ePanelKeyBoardCode_ESC = 0x48,
ePanelKeyBoardCode_Left = 0x49,
ePanelKeyBoardCode_Right = 0x4A,
ePanelKeyBoardCode_Up = 0x4B,
ePanelKeyBoardCode_Down = 0x4C,
ePanelKeyBoardCode_Enter = 0x4D,
ePanelKeyBoardCode_Shutdown = 0x4E,
ePanelKeyBoardCode_TV_VGA = 0x4F,
ePanelKeyBoardCode_Edit = 0x50,
ePanelKeyBoardCode_Language = 0x51,
ePanelKeyBoardCode_Mute = 0x52,
ePanelKeyBoardCode_Pause = 0x53,
ePanelKeyBoardCode_Stop = 0x54,
ePanelKeyBoardCode_AlarmClear = 0x55,
ePanelKeyBoardCode_Defence = 0x56,
ePanelKeyBoardCode_Capture = 0x57,
ePanelKeyBoardCode_LightAdd = 0x58,
ePanelKeyBoardCode_LightSub = 0x59,
ePanelKeyBoardCode_ContrastAdd = 0x5A,
ePanelKeyBoardCode_ContrastSub = 0x5B,
ePanelKeyBoardCode_SpeedAdd = 0x5C,
ePanelKeyBoardCode_SpeedSub = 0x5D,
ePanelKeyBoardCode_Set = 0x5E,
ePanelKeyBoardCode_F1 = 0x5F,
ePanelKeyBoardTypeEnd,
} EPanelKeyBoardCodeType;

!/* < multi-screen */
!/* < Switch */
!/* < Auxiliary */
!/* < Play */
!/* < rewind */
!/* < Video */
!/* < exit */
!/* < Left */
!/* < Right */
!/* <Up */
!/* <Down */
!/* < Confirm */
!/* < Shutdown */
!/* < TV / VGA */
!/* < edit */
!/* < Languages */
!/* < Mute */
!/* < Pause */
!/* < Stop */
!/* <! Eliminate the police */
!/* < arm and disarm */
!/* < capture */
!/* < increase brightness*/
!/* < decrease brightness */
!/* <! Contrast increase */
!/* <! Contrast decrease */
!/* < increase speed */
!/* < reduce speed */
!/* < Set */
!/* < F1*/
!/* <End value */

```

4.20 Frame Header

typedef struct

```

{
    UINT DataPacketStartCode;    /*!< the start code of frame */
    UINT FrameNo;                /*!< the number of frame */
    BYTE FrameType;              /*!< the type of frame 1.I key frame , 2. P
                                frame, 8.audio frame */
    BYTE FrameInfo;              /*!< the codec type of frame */
    UINT ShowTime;               /*!< Relative timestamp (current Showtime -
                                last showTime)/90 = current frame duration(unit:ms) */
    UINT Sec;                    /*!< second of absolute timestamp */
    UINT uSec;                   /*!< microseconds of absolute timestamp */
    unsigned short reserved;     /*!< Reserve */
    UINT nOverloadLen;           /*!< the length of frame date */
} _zwFrameHeader;

```

note: Structure must be single-byte aligned.

4.21 Probe Device (Search Device)Parameter

4.21.1 Probe Setting Parameter

```
typedef struct tagDeviceProbeParameter
{
    WORD m_wBeginPort; /*!< Start Port*/
    Network Client SDK Programming Guide Network Layer
    WORD m_wEndPort; /*!< End Port*/
    DWORD m_dwReserve[DEF_RESERVE_NUM]; /*!<Reservd*/
}DeviceProbeParameter, *LPDeviceProbeParameter;
```

4.21.2 Probe device configuration parameters

```
typedef struct tagDeviceProbeConfig
{
    DWORD m_bytDevType; /*!< DeviceType(Readonly) */
    BYTE m_bytDevChan; /*!< Channel num(Readonly)*/
    BYTE m_bytDevAutoReg; /*!< if Auto registe*/
    BYTE m_bytDevRegInterval; /*!< register time*/
    BYTE m_bytDevMac[MAC_BINARY_ADDRESS_LEN]; /*!<MACaddress(Read)*/
    BYTE m_bytDevIP[IP_ADDRESS_LEN]; /*!< IP*/
    BYTE m_bytDevMask[IP_ADDRESS_LEN]; /*!< MASK */
    BYTE m_bytDevGateway[IP_ADDRESS_LEN]; /*!< Gate*/
    BYTE m_bytDevManHost[IP_ADDRESS_LEN]; /*!< Manage Host*/
    WORD m_bytDevDetectPort; /*!< Listen Port*/
    WORD m_wManHostPort; /*!< Manage Host Port*/
    WORD m_wCmdPort; /*!< CMD Port*/
    WORD m_wMediaPort; /*!< Media Port*/
    DWORD m_dwReserveFirst; /*!< Reservd*/
    DWORD m_dwReserveSecond; /*!< Reservd*/
    DWORD m_dwValidMask; /*!< Valid bitMask*/
}DeviceProbeConfig, *LPDeviceProbeConfig;
```

4.21.3 Probe callback

```
typedef void ( *CB_DeviceProbe)(IN LPDeviceProbeConfig cConfig);
Parameter:
```

cConfig Detect Parameter

return:

0-success

Note:

Interface Type: Blocking

4.22 Retrieve Device List(NVR/IPD Probed not PC)

4.22.1 Probed device list

NVR or IPD will probe(search) devices[other IPC(include others provider' s IPC),DVR or NVR] in LAN and client(your application) can retrieve and handle it .

```
typedef struct tagVIDEONETCLIENT_FRONT_PROBE_DEVICE_INFO
{

```

```
char strDeviceName[DVR_MAX_DEVICE_NAME_LENGTH];    //Device Name
char strDevIp[IP_ADDRESS_LEN];                    //Device IP
char strDevSubmask[IP_ADDRESS_LEN];                //Device Submask
char strDevGateway[IP_ADDRESS_LEN];                //Device Gateway
char strDevMac[MAC_ADDRESS_LEN];                  //Device MAC
char cReserve1[2];                                  //Reserve
DWORD dwDevType;                                    //Device Type[Readonly, keep
it unchanged when set front device connection list]
DWORD dwDevProtocolType;                           //Protocol Type[3:P6S
4:P1, 5.:P2, 6, ONVIF 8, P5,]
DWORD dwDevChannelCount;                           //Channel
DWORD dwDevPort;                                    //Command Port
char cDevUuid[64] ;                                //Device UUID
char dwReserve[256];                                //Reserve
}VIDEONETCLIENT_FRONT_PROBE_DEVICE_INFO;
```

4.22.2 Probe callback

```
typedef void (CALLBACK *CB_DeviceProbeFromNVR)(IN HUSER hUser,IN
VIDEONETCLIENT_FRONT_PROBE_DEVICE_INFO * cConfig,IN DWORD
dwUserData);
```

Parameter:

hUser User ID.
cConfig Device Configuration..
dwUserData User Data。

return:

no

note:

Interface Type: Blocking

4.23 Preview Cruise for IPD

Client can configure IPD to display appropriate video channel to appropriate split-mode, appropriate monitor, appropriate stay time. IPD could works fine independtly if you give it corrected preview cruise list. Another way, client change current spit-mode setting everytime, it' s like preview cruise.

The first way, IPD is independent, it works fine when client crash. and switching speed of video faster than than the second way. the first method is recommended.

The second method may be easier than first one. Client could harder control IPD.

4.23.1 Preview Cruise List

```
// Preview Curist List
//!\brief the maxium amound of cruise node
#define DEV_MAX_POLL_NODE_NUM      (64)
#define DEV_MAX_SUPPORT_CHANNEL    (64)

typedef enum
{
    SINGLE      = 1,
    THREE       = 3,
    FOUR        = 4,
    SIX         = 6,
    EIGHT       = 8,
    NINE        = 9,
    TEN         = 10,
    TWELVE      = 12,
    THIRTEEN    = 13,
    FIFTEEN     = 15,
    SIXTEEN     = 16,
    TWENTYFIVE  = 25,
    THIRTYSIX   = 36,
    SIXTYFOURTH = 64
}enumScreenType;

//!\brief each preview screen has its own channel and split-mode
//! example: THREE mode, the 3 elements of filed channel array will
be fill appropriate channel No. And others elements must be fill to
0xFF indicate this element it invalid.

typedef struct
{
    int bValid;

    //!\brief indicate this struture is valid or invalid.
    int bEnable;

    //!\brief split-mode
    int enScreenType;           // enumScreenType
    int Page;                   // if split-mode more than
64 channels and we can change Page value to setting next 64
channel' s details.
    int dwDuration;             // stay time
    int dwMonitorIndex;         // (abandoned).
    //!\brief video channel index
    unsigned int channel[DEV_MAX_SUPPORT_CHANNEL];
```

```
} VIDEONETCLIENT_PREVIEW_NODE, *LP_VIDEONETCLIENT_PREVIEW_NODE;

// Main Comannd is VIDEONETCLIENT_GET_PREVIEW_LIST and
VIDEONETCLIENT_SET_PREVIEW_LIST.
typedef struct
{
    int bValid;

    unsigned short BeginNodeIndex;          // more split-screen
setting could be implemment using this field if you need.
    unsigned short MonitorIndex;           // priview cruise list
will be displayed On MonitorIndex monitor(retrieve device info and
then to know how much monitor(VGA channel Number))
    //!\brief split-screen node
    VIDEONETCLIENT_PREVIEW_NODE Node[DEV_MAX_POLL_NODE_NUM];
} VIDEONETCLIENT_PREVIEW_LIST, *LP_VIDEONETCLIENT_PREVIEW_LIST;

// Main command of retriving priview cruise list(only support by
IPD),appropriate structure is VIDEONETCLIENT_PREVIEW_LIST
#define VIDEONETCLIENT_GET_PREVIEW_LIST    (500)
// Main command of setting priview cruiselist(only support by IPD)
appropriate structure is VIDEONETCLIENT_PREVIEW_LIST.
#define VIDEONETCLIENT_SET_PREVIEW_LIST    (501)
// SUB command
#define PREVIEW_LIST_CFG_ALL                (0xFFFFFFFF)
```

4.23.2 Start and stop cruise

Reference the define of eRemotePreviewCruiseStart and eRemotePreviewCruiseStop.

4.23.3 fullscreen or resume

```
// fullscreen some channel(Main command is
VIDEONETCLIENT_FULLSCREEN_WINDOW)
typedef struct
{
    int bValid;

    int nChannel;          // which channel need
fullscreen or resume.
```

```
    int dwMonitorIndex;                                // which monitor to
display.
    char cResvered[255];
}VIDEONETCLIENT_FULLSCREEN_CHANNEL;

// Main command of fullscreen some channel for IPD
#define VIDEONETCLIENT_FULLSCREEN_WINDOW      (502)

// Main command of resume some channel for IPD
#define VIDEONETCLIENT_RETURN_FROM_FULLSCREEN (503)

#define CONTROL_PREVIEW_ALL                    (0xFFFFFFFF)
```

5 SDK API Description

5.1 System initialization and anti-initialization

int VideoNetClient_Start ()

Start the service (allocate memory, create threads and prepare other resources)

Returns:

Return 0 for success, or that fails, it returns an error code

Comment:

After starting the service, the service can be invoked when no longer needed to stop the service interface VideoNetClient_Stop

Interface Type: Blocking

int VideoNetClient_Stop ()

Stop Service (release service uses memory, threads and other resources)

Returns:

Return 0 for success, or that fails, it returns an error code

Comment:

Interface Type: Blocking

5.2 System SDK Properties

int VideoNetClient_GetAttribute (IN eClientSDKAttributeType eType, OUT DWORD * dwAttribute)

Get Client SDK Properties (SDK interfaces are used to control the behavior of the client)

Parameters:

eType Client SDK property types, see [eClientSDKAttributeType](#) .

dwAttribute Client SDK property values

Returns:

Return 0 for success, or that fails, it returns an error code

Comment:

Interface Type: Blocking

int VideoNetClient_SetAttribute (IN eClientSDKAttributeType eType, IN DWORD dwAttribute)

Setting Client SDK Properties (SDK interfaces are used to control the behavior of the client)

Parameters:

eType Client SDK property types, see [eClientSDKAttributeType](#) .

dwAttribute Client SDK property values

Returns:

Return 0 for success, or that fails, it returns an error code

Comment:

Interface Type: Blocking

5.3 User login, exit, user event management

int VideoNetClient_UserLogin (OUT HUSER * hUser, IN const LPUserLoginPara cUserLoginPara)

Users log on, all the operations of the equipment required to login first

Parameters:

hUser User handle addresses, interfaces fill the handle value after a successful login

cUserLoginPara Type the user login parameter information can be found in their specific cUserLoginPara structure [UserLoginPara](#) Definition Description

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_UserLogout (IN HUSER hUser)

The user logs off, the value of this handle will be invalid after a successful logout

Parameters:

hUser Users handle value through user login to get a handle

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_DeleteUserForceCB (IN CB_DeleteUserForce cbDeleteUserForce, IN DWORD dwUserData)

Device side can be forced to delete the user has logged in, the user will be notified via a callback when removed.

This interface settings force the removal of the logged in user callback interface.

Application layer can use this interface to set mandatory delete the logged in user callback, which inform the user interfaces have been forced to delete the device side, due to the need to respond to this callback function devices, in order to ensure real-time, it is recommended not to be blocked in the callback operation.

Parameters:

cbDeleteUserForce Forced removal notification callback interface, see

[CB_DeleteUserForce](#) .

dwUserData User data passed to the application layer to send commands.

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_SubscribeEvent (IN HUSER *hUser*, IN DWORD *dwEventTypes*, IN CB_UserEvent *cbEvent*, IN DWORD *dwUserData*)

Subscribers events

When specifying the type of event server occurs, notification is sent to a callback request processing.

Parameters:

hUser Users handle user login to get a handle .

dwEventTypes remote event type, by or (!) operator can subscribe to a variety of events, share the same callback and user data, it is desirable types are as follows (see [remote event type list](#)):

	USEREVENT_ALARM_NOTICE	Alarm events
	USEREVENT_HEARTBEAT_LOST	Heartbeat is lost, disconnected from the
network	USEREVENT_NET_RECOVER	Successful network reconnection
	USEREVENT_USER_DISCONNECT	Remote user disconnects
	USEREVENT_STREAM_DISCONNECT	Remote streaming off
	USEREVENT_DISKGROUP_MANAGE	Hard disk group management event
	USEREVENT_HISTORYSTREAM_NOTICE	History stream event notification
	USEREVENT_REALSTREAM_STARTLINK	Live stream start the connection ID
notification	USEREVENT_REALSTREAM_STOPLINK	Real-time flow stops connection ID
notification	USEREVENT_VOICESTREAM_STARTLINK	Voice stream to initiate a connection ID
notification	USEREVENT_VOICESTREAM_STOPLINK	Voice stream stops connection ID
notification	USEREVENT_HISTORYSTREAM_DESTROY LINK	History stream destruction event
notification	USEREVENT_HISTORYSTREAM_CREATELINK	History flows create an event
notification	USEREVENT_HISTORYSTREAM_START LINK	History flows start event
notification	USEREVENT_HISTORYSTREAM_STOP LINK	Stop the flow of history event

cbEvent callback function, see [CB_UserEvent](#) definition.

dwUserData User data back to the application layer event callbacks.

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_SubscribeEventEx(IN HUSER hUser, IN DWORD dwEventTypes, IN CB_UserEventEx cbEventEx, IN DWORD dwUserData);

Subscribers events

When specifying the type of event server occurs, notification is sent to a callback request processing.

Parameters:

hUser Users handle user login to get a handle .

dwEventTypes remote event type, by or (|) operator can subscribe to a variety of events, share the same callback and user data, it is desirable types are as follows (see [remote event type list](#)):

	USEREVENT_ALARM_NOTICE	Alarm events
	USEREVENT_HEARTBEAT_LOST	Heartbeat is lost, disconnected from the
network	USEREVENT_NET_RECOVER	Successful network reconnection
	USEREVENT_USER_DISCONNECT	Remote user disconnects
	USEREVENT_STREAM_DISCONNECT	Remote streaming off
	USEREVENT_DISKGROUP_MANAGE	Hard disk group management event
	USEREVENT_HISTORYSTREAM_NOTICE	History stream event notification
	USEREVENT_REALSTREAM_STARTLINK	Live stream start the connection ID
notification	USEREVENT_REALSTREAM_STOPLINK	Real-time flow stops connection ID
notification	USEREVENT_VOICESTREAM_STARTLINK	Voice stream to initiate a connection ID
notification	USEREVENT_VOICESTREAM_STOPLINK	Voice stream stops connection ID
notification	USEREVENT_HISTORYSTREAM_DESTROY LINK	History stream destruction event
notification	USEREVENT_HISTORYSTREAM_CREATELINK	History flows create an event
notification	USEREVENT_HISTORYSTREAM_START LINK	History flows start event
notification	USEREVENT_HISTORYSTREAM_STOP LINK	Stop the flow of history event
notification		
	<i>cbEvent</i> callback function, see CB_UserEvent definition.	
	<i>dwUserData</i> User data back to the application layer event callbacks.	

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_UnSubscribeEvent (IN HUSER hUser, IN DWORD dwEventTypes)

Unsubscribe server event notifications

When specifying the type of event occurs the device side, no longer inform the client.

Parameters:

hUser Users handle user login to get a handle .

dwEventTypes remote event type, by or (|) operator a variety of events can unsubscribe, see the specific VideoNetClient_SubscribeEvent instructions.

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

5.4 Get and set the device configuration information

int VideoNetClient_SetConfig (IN HUSER *hUser*, IN const LPConfigInformation *cConfig*)

Set configuration information

Parameters:

hUser Users handle user login to get a handle

cConfig Type of configuration information, can be found in their specific cConfig structure

[ConfigInformation](#) Definition Description

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_GetConfig(IN HUSER *hUser*, IN LPConfigInformation *cConfig*)

Obtain configuration information

Parameters:

hUser Users handle user login to get a handle

cConfig Type of configuration information, can be found in their specific cConfig structure

[ConfigInformation](#) Definition Description

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_SetConfigV3(IN HUSER *hUser*, IN const LPConfigInformationV3 *cConfig*)

Set configuration information (Support more than 32 channels)

Parameters:

hUser Users handle user login to get a handle

cConfig Type of configuration information, can be found in their specific cConfig structure

[ConfigInformationV3](#) Definition Description

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_GetConfigV3 (IN HUSER *hUser*, IN LPConfigInformationV3 *cConfig*)

Obtain configuration information (Support more than 32 channels)

Parameters:

hUser Users handle user login to get a handle

cConfig Type of configuration information, can be found in their specific cConfig structure

[ConfigInformationV3](#) Definition Description

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

5.5 Real-time streaming operations

int VideoNetClient_RealStreamConnect (OUT HSTREAM * *hStream*, IN HUSER *hUser*, IN const LPRealStreamPara *cStreamPara*)

Live stream connection

When the connection is live stream, after recovery network if the network is disconnected, the live stream will automatically recover their connections.

Parameters:

hStream Streaming media handler address, the interface will fill the handle value after a successful connection

hUser Users handle user login to get a handle

cStreamPara Live stream connection information, cStreamPara specific information, see [RealStreamPara](#) defined

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_RealStreamDisconnect (IN HSTREAM *hStream*)

Disconnect the live stream, and will close its corresponding broadcast channel

Parameters:

hStream Handle streaming, live streaming available via connection

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

5.6 History query stream

int VideoNetClient_HistoryStreamQueryConnect (OUT HSTREAM_QUERY * *hStreamQuery*, IN HUSER *hUser*, IN LPHistoryStreamQueryFactor *cStreamQueryFactor*)

Establish historical stream query join operations

History query is mainly used for streaming video capture device-side data distribution in time, the query results in the form of fragments given time, the video data used to establish the historical timeline, a clear description of a certain time period video channel type information.

When a query is completed or terminated midway, need to call off the operation to release the resources.

Parameters:

hStreamQuery History stream query handler address, the interface to fill the handle value after a successful connection

hUser Users handle user login to get a handle

cStreamQueryFactor Historical flow query, reference [HistoryStreamQueryFactor](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryStreamQueryDisconnect (IN HSTREAM_QUERY hStreamQuery)

History stream query disconnected operation.

When a query is completed or terminated midway, need to call this interface to release resources.

Parameters:

hStreamQuery History query handle the flow by connecting the historical stream query to get

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryStreamQueryNext (IN HSTREAM_QUERY hStreamQuery, OUT LPHistoryStreamQueryResult pStreamQueryResult)

Under a stream query history

The result is stored in *pStreamQueryResult* in time and video clip type given in the form.

Parameters:

hStreamQuery History query handle the flow by connecting the historical stream query to get

pStreamQueryResult History of a stream query record, reference [HistoryStreamQueryResult](#) .

Returns:

Return 0 for success, eQueryBusy (2) indicates that the query is busy, eQueryFinished (3) indicates that the query completes, the other indicates an error code, as defined [eQueryStatus](#) .

Comment:

Interface Type: Blocking

5.7 History flow operation

int VideoNetClient_HistoryStreamCreate (OUT HSTREAM * hHistoryStream, IN HUSER hUser, IN const LPHistoryStreamPara cStreamPara)

Create a flow channel history

Parameters:

hHistory Interface fill the handle value after flow channel handle address history, creating successful

hUser Users handle user login to get a handle

cStreamPara History channel parameter information, the type specific cStreamPara can be found in its structure [HistoryStreamPara](#) Definition Description

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryStreamDestroy (IN HSTREAM hHistoryStream)

Destruction of historical flow channel, and will close its corresponding broadcast channel

Parameters:

hHistory History channel handle, get the channel by creating history

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryStreamPosition (IN HSTREAM hHistoryStream, IN eOperationType eOperation, IN_OUT TimeInfo * cTime)

Gets or sets the historical flow channel time position

Parameters:

hHistoryStream History flow channel handle, get the channel by creating history

eOperation History flow channel operation type, with eOperationSet set with eOperationGet get. History streaming download progress also obtain this interface, see [eOperationType](#) .

cTime History channel time information, see [TimeInfo](#) :

When *eOperation* == eOperationSet, CTime positioning time;

When *eOperation* == eOperationGet, CTime has been received for the current time stream.

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryStreamSetSpeed (IN HSTREAM hHistory, IN DWORD dwSpeed)

Setting history flow fast-motion (1X, 2X, 4X, 8X, 16X)

Parameters:

hHistory Stream handle, get the flow by connecting history.

dwSpeed Speed, value 1/2/4/8/16 1 indicates normal speed, 16 said 16x.

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryStreamSync (IN HSTREAM hHistory, IN HSTREAM hDestHistory)

Multiple simultaneous playback streams of history

Parameters:

h History Stream handle, get the flow by connecting history.

hDestHistory purpose stream handle to get by connecting historical stream. stream data indicates that this path with h History synchronous playback, Basis *h History*, will make the same operation on this road stream flow when operating benchmarks

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryStreamUnsync (IN HSTREAM *h History*, IN HSTREAM *hDestHistory*)

Excluding have multi-channel synchronous playback of a stream flow channel, and then on the basis of current operations do not affect the synchronized flow is removed

Parameters:

h History Stream handle, get the flow by connecting history.

hDestHistory Stream handle, get the flow by connecting history.

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryStreamCB (IN HSTREAM *hStream*, IN CB_StreamMedia *cbStreamMedia*, IN DWORD *dwUserData*)

Setting history stream callback interface

Parameters:

hStream Stream handle, get the flow by connecting history.

cbStreamMedia History stream callback interface. You can set this time to cancel the callback is empty (NULL), defined [CB_StreamMedia](#) .

dwUserData Passed to the application layer user data when the callback

Returns:

Return 0 for success, otherwise an error code

Comment:

In order to ensure real-time, it is recommended not to do blocking operations *cbStreamMedia* callback to avoid dropped frames.

In the history of the stream callback interface need to be addressed in the frame type, with particular attention to the following types:

Frame Type	Number	Description
HISTORY_STREAM_EXCEPTION	0x01000000	The end of history flow anomalies
HISTORY_STREAM_SWITCH_FRAME	0x02000000	History stream switching frames
HISTORY_STREAM_TIME_JUMP	0x03000000	Historical flow jumps to the new time slot
HISTORY_STREAM_TIME_END	0x04000000	The end of history flow frame
HISTORY_STREAM_TIME_CURR		Historical flow data query is completed

	0x05000000	
--	------------	--

Interface Type: Blocking

5.8 Voice talk

int VideoNetClient_VoiceStreamConnect (OUT HSTREAM * *hStream*, IN HUSER *hUser*, IN const LPVoiceStreamPara *cStreamPara*)

Connecting voice stream

Parameters:

hStream Voice stream handler address, the interface connected to populate the handle value
hUser Users handle user login to get a handle
cStreamPara Voice stream connection information, cStreamPara specific information, see [VoiceStreamPara](#) defined

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_VoiceStreamDisconnect (IN HSTREAM *hStream*)

Disconnect the voice stream

Parameters:

hStream Handle voice traffic, voice traffic VideoNetClient_VoiceStreamConnect get through the connection

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_VoiceStreamCB (IN HSTREAM *hStream*, IN CB_StreamVoice *cbStreamVoice*, IN DWORD *dwUserData*)

Set voice stream callback interface

Parameters:

hStream Handle voice traffic flow to get through the connection of voice
cbStreamVoice Voice stream callback interface, if the value is empty (NULL), that is no longer needed correction, defined [CB_StreamVoice](#) .
dwUserData Back to the caller when the voice stream callback

Returns:

Return 0 for success, otherwise an error code

Comment:

Since this callback function to receive and capture thread calls the voice stream data, in order to ensure real-time, please minimize blocking work in the callback to avoid dropped frames.
 Interface Type: Blocking

int VideoNetClient_VoiceStreamSend (IN HSTREAM *hStream*, IN const LPBuffer *cFrame*)

Send voice stream data interface.

Parameters:

hStream Handle voice traffic flow to get through the connection of voice

cFrame Voice frame data stream may be obtained from the callback voice. LPBuffer see [Buffer](#) definitions

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

5.9 Log Query

int VideoNetClient_HistoryLogQueryConnect (OUT HLOG_QUERY * hLogQuery, IN HUSER hUser, IN LPLogQueryFactor StreamDecodePara)

Log consultation to establish the connection operation

Log queries mainly for local and remote access to the system operation and performance information recording device side.

When a query is completed or terminated midway, need to call off the operation to release the resources.

Parameters:

hLogQuery History log query handler address, the interface to fill the handle value after a successful connection

hUser Users handle user login to get a handle

p LogQueryFactor History query log, see [LogQueryFactor](#) .

Returns:

Return 0 for success, eQueryBusy (2) indicates that the query is busy, eQueryFinished (3) indicates that the query, the other indicates an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryLogQueryDisconnect (IN HLOG_QUERY hLogQuery)

History Log Query disconnected operation

When a query is completed or terminated midway, need to call this interface to release resources.

Parameters:

hLogQuery History log query handle, by connecting history log query to get

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_HistoryLogQueryNext (IN HLOG_QUERY hLogQuery, OUT LPLogQueryResult pLogQueryResult)

Under a history log query

Parameters:

hLogQuery History log query handle, by connecting history log query to get
pLogQueryResult Queries a history log records, see [LogQueryResult](#) .

Returns:

Return 0 for success, eQueryBusy (2) indicates that the query is busy, eQueryFinished (3) indicates that the query completes, the other indicates an error code, see [eQueryStatus](#) .

Comment:

Interface Type: Blocking

5.10 Transparent Channel

int VideoNetClient_TransparentChannelConnect (OUT HTRANSPARENT * hTransparent, IN HUSER hUser, IN LPTransparentChannelPara cTransparentPara)

Transparent channel connection operation

Transparent channel for communication between the client and the client device to send and receive information, information format by the recipient and sender to agree.

Transparent channel when no longer in use, the need to call off the operation, in order to free up resources.

Parameters:

hTransparent Transparent channel handle, the handle value populate the interface connection
hUser Users handle user login to get a handle
cTransparentPara Transparent channel connection parameters, see [TransparentChannelPara](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_TransparentChannelDisconnect (IN HTRANSPARENT hTransparent)

Clear Channel disconnected operation

Transparent channel when no longer in use, you need to call this interface to release resources.

Parameters:

hTransparent Transparent channel handle, transparent channel to get through the connection

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_TransparentChannelWrite (IN HTRANSPARENT hTransparent, IN const LPBuffer p Buffer)

Transparent data transmission channel

Parameters:

hTransparent Transparent channel handle, transparent channel to get through the connection

p Buffer Transparent channel data buffer, see [Buffer](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Each time the maximum write 4K bytes, writes timeout is 30 seconds.
Interface Type: Blocking

**int VideoNetClient_TransparentChannelCB (IN HTRANSSPARENT hTransparent, IN
CB_TransparentChannel cbTransparent, IN DWORD dwUserData)**

Set clear channel data callback interface

Transparent channel device sends data received, the call *cbTransparent* callback processing.

Parameters:

hTransparent Transparent channel handle, transparent channel to get through the connection
cbTransparent Transparent channel data callback interface, if the value is empty (NULL),
that is no longer needed correction, see [CB_TransparentChannel](#) .
dwUserData Transparent channel data back to the application layer callback

Returns:

Return 0 for success, otherwise an error code.

Comment:

Since this callback function in a transparent channel data reception thread calls, in order to ensure real-time, please minimize blocking work in the callback to avoid dropped frames.
Interface Type: Blocking

5.11 PTZ control

**int VideoNetClient_PtzControl (IN HUSER hUser, IN DWORD dwChannel, IN
ePTZControlCmdCode eCommandCode, IN DWORD dwParam1, IN DWORD dwParam2, IN
DWORD dwParam3, DWORD dwParam4)**

Remote PTZ control

Parameters:

hUser User Login handle
dwChannel Channel number (starting from 0)
eCommandCode PTZ command word, see [ePTZControlCmdCode](#) .
dwParam1 PTZ command parameters 1
dwParam2 PTZ command parameter 2
dwParam3 PTZ command parameter 3
dwParam4 PTZ command parameter 4

Returns:

Return 0 for success, otherwise an error code

5.12 File Query

**int VideoNetClient_FileQueryConnect (OUT HFILE_QUERY * hFileQuery, IN HUSER hUser,
IN LPFileQueryFactor pFileQueryFactor)**

File query join operations

File details to get the file specified criteria for the query.

After the query is completed or terminated, the need to call off the operation, in order to free up resources.

Parameters:

hFileQuery Interface to populate the file handle value query handle, after connecting
hUser Users handle user login to get a handle
pFileQueryFactor File query, see [FileQueryFactor](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_FileQueryDisconnect (IN HFILE_QUERY hFileQuery)

File check off operation

After the query is completed or terminated, the need to call this interface to release resources.

Parameters:

hFileQuery File handle queries, the query to get through the connection file

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_FileQueryNext (IN HFILE_QUERY hFileQuery, OUT LPFileQueryResult pFileQueryResult)

Under a document query

Parameters:

hFileQuery File handle inquiries Query to get through the connection file.
pFileQueryResult File query a record, see [FileQueryResult](#) .

Returns:

Return 0 for success, eQueryBusy (2) indicates that the query is busy, eQueryFinished (3) indicates that the query completes, the other indicates an error code, see [equery S TATUS](#) .

Comment:

Interface Type: Blocking

5.13 File upload and download

int VideoNetClient_FileUploadConnect (OUT HFILE_TRANSFER * hFileTransfer, IN HUSER hUser, IN LPFileUploadPara cFileUploadPara)

Establish a connection file upload

After the connection is established file upload, file upload process begins.

Parameters:

hFileTransfer connection handle file uploads
hUser Users handle
cFileUploadPara File upload parameters, see [FileUploadPara](#) .

Returns:

Return 0 for success, the other indicates an error code.

Comment:

Interface Type: Blocking

int VideoNetClient_FileUploadDisconnect (IN HFILE_TRANSFER hFileTransfer)

Disconnect the file upload connection

When the file upload is completed or terminated, the need to call this interface to release resources.

Parameters:

hFileTransfer Connection handle file uploads

Returns:

Return 0 for success, the other indicates an error code.

Comment:

Interface Type: Blocking

int VideoNetClient_FileUploadCB (IN HFILE_TRANSFER hFileTransfer, IN CB_FileUpload cbFileUpload, IN DWORD dwUserData)

Set file upload callback interface

Upload process will upload progress and status reports via the interface

Parameters:

hFileTransfer connection handle file uploads

cbFileUpload Callback interface, see [CB_FileUpload](#) .

dwUserData User data passed to the callback when the callback user interfaces

Returns:

Return 0 for success, the other indicates an error code.

Comment:

Interface Type: Blocking

int VideoNetClient_FileUploadControl (IN HFILE_TRANSFER hFileTransfer, IN eFileTransferControl eControlCode)

File upload control, file transfer is started or stopped.

Parameters:

hFileTransfer connection handle file uploads

eControlCode Control operations, start or stop the file transfer, see [eFileTransferControl](#) .

Returns:

Return 0 for success, the other indicates an error code.

Comment:

Interface Type: Blocking

int VideoNetClient_FileDownloadConnect (OUT HFILE_TRANSFER * hFileTransfer, IN HUSER hUser, IN LPFileDownloadPara p FileDownloadPara)

Establish a file download link

After the file download link is established, the file download process begins.

Parameters:

hFileTransfer file download connection handle

hUser Users handle

p File DownloadPara Download parameters, see [FileDownloadPara](#) .

Returns:

Return 0 for success, the other indicates an error code.

Comment:

Interface Type: Blocking

int VideoNetClient_FileDownloadDisconnect (IN HFILE_TRANSFER hFileTransfer)

Disconnect the file download link

When the file download is completed or terminated, the need to call this interface to release resources.

Parameters:

hFileTransfer file download connection handle

Returns:

Return 0 for success, the other indicates an error code.

Comment:

Interface Type: Blocking

int VideoNetClient_FileDownloadCB (IN HFILE_TRANSFER hFileTransfer, IN CB_FileDownload cbFileDownload, IN DWORD dwUserData)

Download callback interface settings file

Download process will download progress and status reports via the interface

Parameters:

hFileTransfer file download connection handle

cbFile Download Callback interface when the callback, cBuffer length is greater than 0, that is being sent is equal to 0 indicates that the transfer is complete, equal to (DWORD) -1, it means that the transmission error, see [CB_FileDownload](#) .

d wUserData User data passed to the callback when the callback user interfaces

Returns:

Return 0 for success, the other indicates an error code.

Comment:

Interface Type: Blocking

int VideoNetClient_FileDownloadControl (IN HFILE_TRANSFER hFileTransfer, IN eFileTransferControl eControlCode)

File upload control, file transfer is started or stopped.

Parameters:

hFileTransfer connection handle file uploads

eControlCode Control operations, start or stop the file transfer, see [eFileTransferControl](#) .

Returns:

Return 0 for success, the other indicates an error code.

Comment:

Interface Type: Blocking

5.14 Remote control operation

int VideoNetClient_DeviceControl (IN HUSER *hUser*, eRemoteDeviceControl *eOperationCode*)

Remote device control

Remote control device to shutdown / restart / standby / resume system default

Parameters:

hUser Users handle user login to get a handle

eOperationCode Equipment operation code, see [eRemoteDeviceControl](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_ForceIFrame (IN HUSER *hUser*, IN DWORD *dwChannel*)

Remote forced I-frame

Parameters:

hUser Users handle user login to get a handle

dwChannel Channel number

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_AlarmClear (IN HUSER *hUser*, IN DWORD *dwChannel*, IN DWORD *dwAlarmType*)

Clear alarm remote

Parameters:

hUser Users handle user login to get a handle

dwChannel Channel number

dwAlarmType Alarm Type

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_ImageCapture (IN HUSER *hUser*, IN DWORD *dwChannel*, IN eImageFormat *eFormat*, OUT LPImageFileInfo *pImageFileInfo*)

Remote Capture

The captured image files stored on the device side.

Parameters:

hUser Users handle user login to get a handle

dwChannel Channel number

eFormat Image formats, see [eImageFormat](#) .

pImageFileInfo Picture information, see [ImageFileInfo](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_FileLock (IN HUSER *hUser*, IN char * *szFileName*, IN *eFileLockOperationCode* *eOperationCode*)

Remote file locking and unlocking

Locked files will not be deleted in the rotation process.

Parameters:

hUser Users handle user login to get a handle

szFileName File Name

eOperationCode Opcode (lock / unlock), see [eFileLockOperationCode](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_RecordControl (IN HUSER *hUser*, IN DWORD *dwChannel*, IN DWORD *dwRecordType*)

Remote video control Set a channel recording mode.

Parameters:

hUser Users handle user login to get a handle

dwChannel Channel number

dwRecordType Video type , All three recording modes: 0 - manual recording mode; 1 - Automatic video mode; 2 - stop recording

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_AlarmoutControl (IN HUSER *hUser*, IN DWORD *dwChannel*, IN DWORD *dwSwitch*)

Remote Alarm Output Control

Parameters:

hUser Users handle user login to get a handle

dwChannel Alarm output channel number, channel number rather than video coding

dwSwitch Switch status, 0 - off, 1 - Open

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_AlarmoutStateGet (IN HUSER *hUser*, OUT DWORD * *dwChanNum*, OUT DWORD * *dwStateBits*)

Remote Alarm output status to obtain

Parameters:

hUser Users handle user login to get a handle
dwChanNum Effective alarm output channel number
dwStateBits Bit 0-31 denote 0 - 31 channel alarm output status

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_DiskGroupManage (IN HUSER *hUser*, IN eDiskGroupOperation *eOperationCode*, IN DWORD *dwParam1*, IN DWORD *dwParam2*, IN DWORD *dwParam3*)

Remote storage management

Remote hard disk and hard disk group management of remote command interface.

Parameters:

hUser Users handle user login to get a handle
eOperationCode Opcode see [eDiskGroupOperation](#) .
dwParam1-dwParam 3 Parameter List

Operation type \ Parameters	dwParam1	dwParam2	dwParam3
eUnMountDisk	HDD index	0	Operation verification code used to identify the current operation is based on the correct configuration; may store information obtained.
eCreateNewPartition	HDD index	Partition size (MB)	verification code
eDeletePartition	HDD index	Partition index	verification code
eFormatPartition	HDD index	Partition index	verification code
eDiskGroupKeepTime	Disk group type: Normal disk group (0), alarm disc set (1), redundant disk group (2), the backup disk group (3)	Data hold time (S)	0
eDiskGroupBindChannel	Disk group type (ditto)	Video Type: No video (0), alarm recording (1), normal recording (2), manual recording (3), mobile video (4)	Channel mask, that is, each bit bit identifies each channel of support for 32 channels. Bit 0 indicates the delete, 1 add.
eDiskGroupAddPartition	Disk group type (ditto)	Partition mount point Four bytes, as sda6, Middle-byte character	0

		code stored sda, low byte is the number one digital (16 hex), which is 0x73646106.	
eDiskGroupDelPartition	Disk group type (ditto)	Partition mount point (ibid.)	0
eInitializeDisk	HDD index	0	verification code

Returns:

Return 0 for success, otherwise an error code

Comment:

Store operation is asynchronous, the result can be obtained by subscription
USEREVENT_DISKGROUP_MANAGE events.

The significance of the event parameters see [CB_UserEvent](#) definition.

Interface Type: Blocking

5.15 Data Query

int VideoNetClient_DataExistCheck (IN HUSER *hUser*, IN DWORD *dwChannel*, IN DWORD *dwMajorType*, IN DWORD *dwMinorType*, IN char * *szYearMonth*, OUT DWORD * *dwResult*)

Analyzing the data type specified day of the existence

Given a month, press the digit returns in *dwResult* each day of the month whether the specified type of data.

Parameters:

hUser User Login handle

dwChannel Channel number (starting from 0; When the query image files, this parameter indicates the channel number, when a query video files, this parameter indicates the disk group number)

dwMajorType Main types 0 - video files 1 - picture file

dwMinorType Sub-type (currently not used)

szYearMonth Date time string (eg 201003)

dwResult return results obtained, 0-30, 1-31, respectively, corresponding to the date data exists, the corresponding bit is 1, otherwise 0

Returns:

Return 0 for success, otherwise an error code

int VideoNetClient_GetDataSize (IN HUSER *hUser*, IN DWORD *dwChannelBits*, IN eDiskGroupType *eDskGroupType*, IN DWORD *eStreamType*, IN const char * *szStartTime*, IN const char * *szEndTime*, IN DWORD *dwUserData*, OUT DWORD * *dwDataSize*)

Get the video data size of a disk group to a particular channel class type within a certain period of time.

Parameters:

hUser User Login handle

dwChannelBits Channel mask, bit0-bit31 represent channel 1 - 32 data channels are included in the calculation.

eDskGroupType Specify a disk group, see [eDiskGroupType](#) .

eStreamType Specify the video type, see [the history of the video stream type](#) .

szStartTime Start time, date time string Such as "20091029103030."

sz End Time End time, date time string

dwUserData User data

dwDataSize Data size, the asynchronous case, this parameter is invalid, but can not be NULL.

Returns:

Return 0 for success, otherwise an error code

Comment:

This feature uses asynchronous implementation, access to the data size can be obtained by subscription `USEREVENT_HISTORYSTREAM_NOTICE` event. See `CB_UserEvent` significance of the event parameter definitions.

5.16 Device Registration

int VideoNetClient_DeviceRegisterListenStart (IN char * sListenIP, IN DWORD dwListenPort)

Registration request to start the listener, listening devices

Parameters:

IP addresses *sListenIP* PC machine, if it is NULL, SDK will automatically obtain IP address of the PC, if the PC has multiple IP addresses, you can specify an IP address for monitoring.

dwListenPort Local listening port number, set by the user;

Returns:

Return 0 for success, otherwise an error code

int VideoNetClient_DeviceRegisterListenStop ()

Stop the listener

Returns:

TRUE indicates success, FALSE indicates failure

int VideoNetClient_DeviceRegisterCB (IN CB_DeviceRegister cbDeviceRegister)

Set register callback functions

When the device sends the registration information to the machine, the callback function to handle the registration process.

Parameters:

cbDeviceRegister callback function pointer, see [CB_DeviceRegister](#) .

Returns:

Return 0 for success, otherwise an error code

5.17 Convert string lattice

int VideoNetClient_StringToLattice (IN const char * pString, IN const LPFontPara cFont, OUT void * buffer, IN_OUT DWORD * dwBuflen, OUT DWORD * dwWidth, OUT DWORD * dwHeight)

To convert a string into a lattice

Parameters:

pString Source string.

cFont Font information, see [FontPara](#) .
buffer Lattice data receive buffer
dwBuflen Lattice data buffer length, the conversion will be filled in as the length of the actual use
dwWidth The actual dot width
dwHeight The actual dot height

Returns:

Return 0 for success, otherwise an error code

5.18 Streaming media control

int VideoNetClient_StreamMediaControl (IN HSTREAM *hStream*, IN eTaskControl *eMediaControl*)

Streaming media control interface, control the start and stop streaming

Parameters:

hStream Streaming handle, get the flow through the connection.
eMediaControl Streaming media control commands, see [eTaskControl](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_StreamMediaCB (IN HSTREAM *hStream*, IN CB_StreamMedia *cbStreamMedia*, IN DWORD *dwUserData*)

Set Streaming callback interface

Parameters:

hStream Streaming handle, get the flow through the connection.
cbStreamMedia Streaming callback interface, if it is NULL, then cancel the callback, see [CB_StreamMedia](#) .
dwUserData Passed to the application layer user data when the callback

Returns:

Return 0 for success, otherwise an error code

Comment:

Application layer can use this interface to set a callback streaming data, because this callback function receives streaming data thread calls,
In order to ensure real-time, please minimize blocking work in the callback to avoid dropped frames.
Interface Type: Blocking

int VideoNetClient_SetStreamMediaLinkID (IN HSTREAM *hStream*, IN DWORD *dwLinkID*)

Set streaming connection ID

Parameters:

hStream Streaming handle, get the flow through the connection.
dwLinkID Streaming connection ID.

Returns:

Return 0 for success, otherwise an error code

Comment:

Media set up the connection ID will be saved to the server before the server restart has always existed, customers can check the connection ID device

Interface Type: Blocking

int VideoNetClient_QueryStreamMediaLinkID (IN HUSER *hUser*, IN eMediaFunctionType *eType*, IN LPMediaLinkIDQueryResult *cQueryResult*)

Set in the streaming media queries on the server connection ID

Parameters:

hUser Users handle

eType Streaming function type, see [eMediaFunctionType](#) .

cQueryResult Acceptance of streaming media query results, see [MediaLinkIDQueryResult](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

5.19 Remote Control Panel

int VideoNetClient_PanelGetStatus (IN HUSER *hUser*, OUT LPPanelStatusInfo *cStatus*)

Get panel status

Parameters:

hUser Users handle

cStatus Streaming function type, see [PanelStatusInfo](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_PanelSetStatus (IN HUSER *hUser*, IN LPPanelStatusInfo *cStatus*)

Settings panel status

Parameters:

hUser Users handle

cStatus Streaming function type, see [PanelStatusInfo](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

int VideoNetClient_PanelControl (IN HUSER *hUser*, IN LPPanelControlParameter *cControl*)

Panel Control

Parameters:

hUser Users handle

cControl Streaming function type, see [PanelControlParameter](#) .

Returns:

Return 0 for success, otherwise an error code

Comment:

Interface Type: Blocking

5.20 Device Probe

int VideoNetClient_DeviceProbeStart(IN const LPDeviceProbeParameter cParameter, IN CB_DeviceProbe cbDeviceProbe);

Start detection equipment, detection equipment to detect information within the specified range

Parameter:

cParameter Detect Setting Parameter, refer DeviceProbeParameter
cbDeviceProbe Detect Callback Function, refer CB_DeviceProbe

return:

Return 0 for success, otherwise an error code

Note :

Interface Type: Blocking

int VideoNetClient_DeviceProbeRefresh(IN const LPDeviceProbeParameter cParameter);

Re-detection equipment, the information in the detection range of the specified detection equipment

parameter:

cParameter refer DeviceProbe Parameter

return:

Return 0 for success, otherwise an error code
Interface Type: Blocking

int VideoNetClient_DeviceProbeStop();

Stop Detect

return:

Return 0 for success, otherwise an error code

Note:

Interface Type: Blocking

```
int VideoNetClient_DeviceProbeSetDeviceConfig(IN const BYTE
*pMac, IN WORD wProbePort, IN const LPDeviceProbeConfig
cConfig);
```

set the device config by detection method.

Parameter:

pMac the mac address you want to modified;
wProbePort detection port
cConfig Device Configuration, refer LPDeviceProbeConfig

return:

Return 0 for success, otherwise an error code

Note:

Interface Type: Blocking

```
int VideoNetClient_DeviceProbeStartV2(IN const
LPDeviceProbeParameter cParameter, IN
CB_DeviceProbeV2 cbDeviceProbeV2)
```

Start detection equipment, detection equipment to detect
information within the specified range

Parameter:

cParameter the second version Device configuration, refer
LPDeviceProbeParameterV2
cbDeviceProbeV2 the callback function, refer CB_DeviceProbeV2.

return:

Return 0 for success, otherwise an error code

Note:

Interface Type: Blocking

```
int VideoNetClient_DeviceProbeRefreshV2(IN const
LPDeviceProbeParameter cParameter)
```


Re-detection equipment, the information in the detection range of the specified detection equipment

Parameter:

cParameter the second version Device configuration, refer LPDeviceProbeParameterV2

return:

Return 0 for success, otherwise an error code

Note:

Interface Type: Blocking

int VideoNetClient_DeviceProbeStopV2()

Stop Detect

return:

Return 0 for success, otherwise an error code

Note:

Interface Type: Blocking

5.21 Device Probe (From NVR)

int CALLSTACK VideoNetClient_SetDetectDeviceListFromNVR
CB(IN HUSER hUser,CB_DeviceProbeFromNVR cbDeviceProbeFromNVR,DWORD dwUserData)
Set callback

parameter:

hUser User ID.
cbDeviceProbeFromNVR Callback function.
dwUserData User Data.

return:

Return 0 for success, otherwise an error code

Note:

Interface Type: Blocking

int CALLSTACK VideoNetClient_DetectDeviceListStart(IN HUSER hUser,IN char * strPathFileName,IN int nPathFileNameLength)

Start Detect

parameter:

hUser User ID.

strPathFileName Template filename(Path + filename).

nPathFileNameLength the length of filename string.

return:

Return 0 for success, otherwise an error code

Note:

Interface Type: Blocking

int CALLSTACK VideoNetClient_DetectDeviceListRefresh(IN HUSER hUser)

Re-detection equipment

parameter:

hUser User ID.

return:

Return 0 for success, otherwise an error code.

Note:

Interface Type: Blocking

int CALLSTACK VideoNetClient_DetectDeviceListStop(IN HUSER hUser)

Stop Detect

parameter:

hUser User ID.

return:

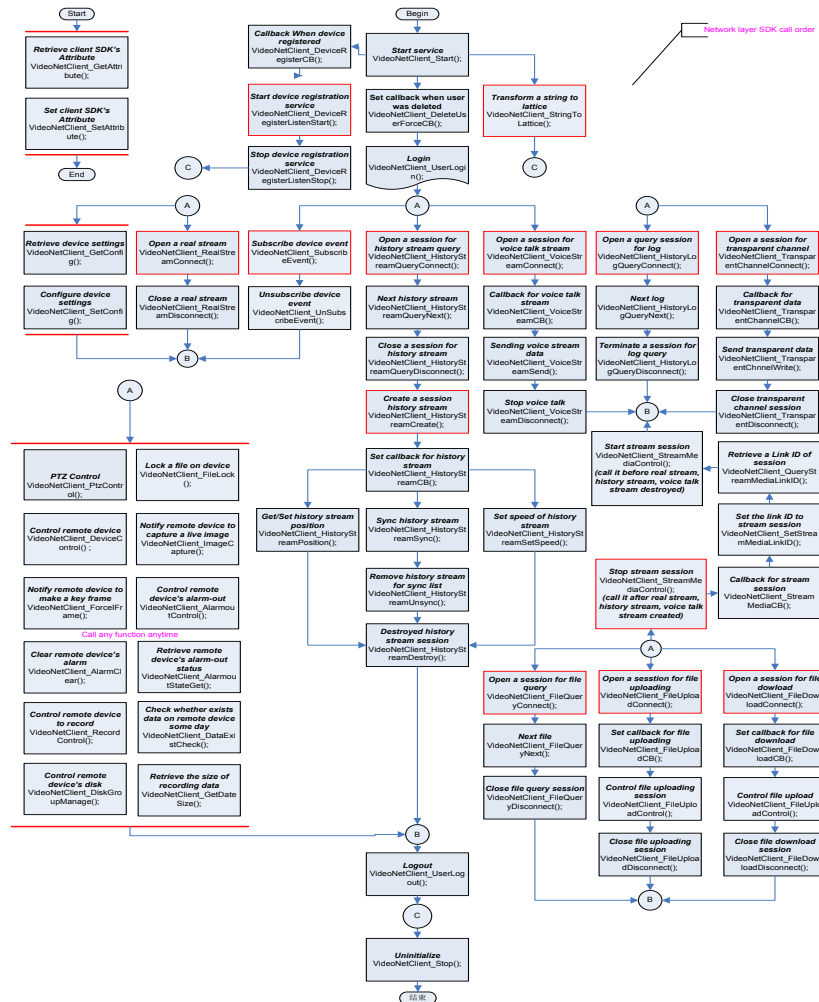
Return 0 for success, otherwise an error code.

Note:

Interface Type: Blocking

6 SDK call order and the samples code

6.1 call order as shown below:



6.2 Examples

6.2.1 login, logout and subscribe events

```
# Include <windows.h>
# Include "VideoNetClient.h"
// Callback function remote forcibly remove users
char m_strUserEvent [14] [50] = {"alarm event", "heartbeat loss", "network reconnection", "User disconnected",
"streaming off", "Disk Management", "history flow notice" "Real Time Streaming notice initiates the connection ID"
"Real-time flow stops the connection ID notification", "the voice stream initiates the connection ID notification",
"Stop the voice stream connection ID notification", "history flow Destruction", "history flow start", "stop the flow of history"};

static int CALLBACK CallBack_DeleteUserForce (IN HUSER hUser, IN DWORD dwUserData)
{
    printf ("\n [CallBack_DeleteUserForce] user is forced to kick the server, user =% d, userdata = 0x% x \n", hUser, dwUserData);
    return 0;
}
```

```
}
static int CALLBACK CallBack_UserEvent (IN HUSER hUser, IN DWORD dwEventType, DWORD
dwParam1, DWORD dwParam2, DWORD dwParam3, IN DWORD dwUserData)
{
    printf("\n [CallBack_UserEvent] [%s] Event callback hUser=%d, EventType=%d, dwParam1=%d,
dwParam2=%d, dwParam3=%d, dwUserData=%d\n",
        m_strUserEvent [dwEventType], hUser, dwEventType, dwParam1, dwParam2, dwParam3,
dwUserData);
    return 0;
}
HUSER UserMan_Login (char * strIp, DWORD port, char * strUserName, char * strPassword)
{
    UserLoginPara tUserlp;
    HUSER hUserlogin = -1;
    int iResult;
    DWORD dwDelUserData = 0x1234;
    int i;

    iResult = VideoNetClient_DeleteUserForceCB (CallBack_DeleteUserForce, dwDelUserData);
    if (iResult)
    {
        printf("[UserMan_Login] VideoNetClient_DeleteUserForceCB failed (%d)\n", iResult);
        return -1;
    }
    memset (& tUserlp, 0, sizeof (UserLoginPara));
    strcpy (tUserlp.sServerIP, strIp, MAX_ADDRESS_LEN);
    tUserlp.dwCommandPort = port;
    strcpy (tUserlp.sUName, strUserName, USERNAME_LEN);
    strcpy (tUserlp.sUPass, strPassword, USERPASS_LEN);

    iResult = VideoNetClient_UserLogin (& hUserlogin, & tUserlp);
    if (iResult)
    {
        printf("[UserMan_Login] Login Failed, return %d\n", iResult);
        return -1;
    }

    // Subscribe event
    for (i = 0; i < 14; i++)
    {
        printf("[UserMan_Login] Subscribe to events <%s>\n", m_strUserEvent [i]);
        iResult = VideoNetClient_SubscribeEvent (hUserlogin, 1 << i, CallBack_UserEvent, 0);
        if (iResult)
        {
            printf("[UserMan_Login] Subscribe to events <%s> failed (%d)\n", m_strUserEvent
[i], iResult);
        }
    }

    return hUserlogin;
}
int main ()
{
    int iResult;
    int RunTime = 100 ;// 100 秒
    HUSER hUser;
    // Start the network service
    printf("[MAIN] Sever Start ... \n");
    iResult = VideoNetClient_Start ();
    if (iResult)
    {
        // Failed to start
        printf("[MAIN] Start Sever failed \n");
        return -1;
    }
    // User Management - landing
    printf("[MAIN] Login ... \n");
    hUser = UserMan_Login ("172.1.12.247", 5050, "Admin", "");
}
```

```
if (hUser == -1)
{
    printf("[MAIN] Login Failed \n");
    goto GOTO_STOP;
}
printf("[MAIN] Login success \n");
////////////////////////////////////
// Has landed successfully, can support all operations.
// Do something
////////////////////////////////////
// Operation is complete, ready to cancel, exit
Exit // 100 seconds
printf("[MAIN] system is running \n");
printf("\n r [MAIN] Countdown to exit:% 3d", RunTime);
while (RunTime)
{
    Sleep (1000);
    RunTime--;
    printf("\n r [MAIN] Countdown to exit:% 3d", RunTime);
    if (RunTime < 0)
    {
        RunTime = 0;
    }
}
printf("\n [MAIN] system ready to exit \n");
GOGO_LOGOUT:
// User Management - Cancelled
printf("[MAIN] LogOut ... \n");
iResult = VideoNetClient_UserLogout (hUser);
if (iResult)
{
    printf("[MAIN] logout failed \n");
}
GOTO_STOP:
printf("[MAIN] Sever Stop ... \n");
iResult = VideoNetClient_Stop ();
if (iResult)
{
    // Stop failure
    printf("[MAIN] Sever Stop failed ... \n");
}
}
```

6.2.2 Device configuration information operations

Configuration operations, related structures and commands in the file header (VideoNetClient_Configure.h) there are defined

With operating time configuration example:

```
int TimeCfg (HUSER hUser)
{
    ConfigInformation cfgi;
    LPVIDEONETCLIENT_TIME pTime;
    int iResult;
    memset (& cfgi, 0, sizeof (ConfigInformation));
    cfgi.dwMainCommand = VIDEONETCLIENT_GET_SYSTIME;
    cfgi.dwAssistCommand = SYSTIME_ALL;
    pTime = (LPVIDEONETCLIENT_TIME) cfgi.sConfig;
    // Start acquisition time
    iResult = VideoNetClient_GetConfig (hUser, & cfgi);
    if (iResult)
    {
        printf("[TimeCfg] Get time failed (%d) \n", iResult);
    } Else
    {
    }
```

```
        printf("[TimeCfg] Get time success [% 04d-% 02d-% 02d% 02d:% 02d:% 02d] \n", pTime->
nYear, pTime-> nMonth, pTime-> nDay, pTime-> nHour, pTime-> nMinute, pTime-> nSecond);
    }
    // Set the start time, day by a set

    cfigi.dwMainCommand = VIDEONETCLIENT_SET_SYSTIME;
    cfigi.dwAssistCommand = SYSTIME_ALL;
    cfigi.dwConfigLen = sizeof(VIDEONETCLIENT_TIME);
    pTime-> nDay ++;
    iResult = VideoNetClient_SetConfig(hUser, & cfigi);
    if (iResult)
    {
        printf("[TimeCfg] Set time failed (%d) \n", iResult);
    } Else
    {
        printf("[TimeCfg] Set time success \n");
    }
    // Get the new time minus one day

    cfigi.dwMainCommand = VIDEONETCLIENT_GET_SYSTIME;
    cfigi.dwAssistCommand = SYSTIME_ALL;
    pTime = (LPVIDEONETCLIENT_TIME) cfigi.sConfig;
// Start acquisition time
    iResult = VideoNetClient_GetConfig(hUser, & cfigi);
    if (iResult)
    {
        printf("[TimeCfg] Get time failed (%d) \n", iResult);
    } Else
    {
        printf("[TimeCfg] Get time success [% 04d-% 02d-% 02d% 02d:% 02d:% 02d] \n", pTime->
nYear, pTime-> nMonth, pTime-> nDay, pTime-> nHour, pTime-> nMinute, pTime-> nSecond);
    }
    // Set the start time, day by a set

    cfigi.dwMainCommand = VIDEONETCLIENT_SET_SYSTIME;
    cfigi.dwAssistCommand = SYSTIME_ALL;
    cfigi.dwConfigLen = sizeof(VIDEONETCLIENT_TIME);
    pTime-> nDay -;
    iResult = VideoNetClient_SetConfig(hUser, & cfigi);
    if (iResult)
    {
        printf("[TimeCfg] Set time failed (%d) \n", iResult);
    } Else
    {
        printf("[TimeCfg] Set time success \n");
    }
    return 0;
}
int main ()
{
    ...
    //////////////////////////////////////
    // Has landed successfully, can support all operations.
    TimeCfig(hUser);
    // Do something
    //////////////////////////////////////
    // Operation is complete, ready to cancel, exit
    ...
}
```

```
{
    RealStreamPara rsp;
    HSTREAM hRealStream [2] = {-1, -1};
    int iResult = 0;
    MediaLinkIDQueryResult LIDQuery;
    memset (& rsp, 0, sizeof (RealStreamPara));
    rsp.dwChannel = 0 ;// first channel
    rsp.eMediaType = eMainVideoAndSound;
    rsp.eTransferMode = eGeneralTCP;
    // Start the connection live stream
    printf("[RealStreamMan] to open the first channel \n");
    iResult = VideoNetClient_RealStreamConnect (& hRealStream [0], hUser, & rsp);
    if (iResult)
    {
        printf("[RealStreamMan] live stream connection [0] failed to return (%d)\n", iResult);
        return -1;
    }
    // Then open a second channel
    rsp.dwChannel = 1 ;// second channel
    rsp.eMediaType = eMainVideoAndSound;
    rsp.eTransferMode = eGeneralTCP;
    // Start the connection live stream
    printf("[RealStreamMan] to open the second channel \n");
    iResult = VideoNetClient_RealStreamConnect (& hRealStream [1], hUser, & rsp);
    if (iResult)
    {
        printf("[RealStreamMan] live stream connection [1] failed to return (%d)\n", iResult);
        VideoNetClient_RealStreamDisconnect (hRealStream [0]);
        return -1;
    }
    // Set the live stream callback
    printf("[RealStreamMan] set up a live stream callback \n");
    iResult = VideoNetClient_StreamMediaCB (hRealStream [0], Callback_RealStreamMedia, 0);
    if (iResult)
    {
        printf("[RealStreamMan] live stream [0] set callback fails, the return (%d)\n", iResult);
    }
    iResult = VideoNetClient_StreamMediaCB (hRealStream [1], Callback_RealStreamMedia, 0);
    if (iResult)
    {
        printf("[RealStreamMan] live stream [1] Set the callback fails, the return (%d)\n", iResult);
    }
    // Start the live stream
    printf("[RealStreamMan] to start the live stream \n");
    iResult = VideoNetClient_StreamMediaControl (hRealStream [0], eTaskStart);
    if (iResult)
    {
        printf("[RealStreamMan] live stream [0] failed to start, return (%d)\n", iResult);
    }
    iResult = VideoNetClient_StreamMediaControl (hRealStream [1], eTaskStart);
    if (iResult)
    {
        printf("[RealStreamMan] live stream [1] failed to start, return (%d)\n", iResult);
    }
    // LinkID
    printf("[RealStreamMan] set LinkID [0] -> 0x80000001 \n");
    iResult = VideoNetClient_SetStreamMediaLinkID (hRealStream [0], 0x80000001);
    if (iResult)
    {
        printf("[RealStreamMan] live stream [0] set LinkID failed to return (%d)\n", iResult);
    }
    printf("[RealStreamMan] set LinkID [1] -> 0x80000002 \n");
    iResult = VideoNetClient_SetStreamMediaLinkID (hRealStream [1], 0x80000002);
    if (iResult)
    {
        printf("[RealStreamMan] live stream [1] Set LinkID failed to return (%d)\n", iResult);
    }
}
```

```
//
printf("[RealStreamMan] live stream running one second \n");
Sleep (1000);

// Query real-time streaming LINKID
printf("[RealStreamMan] check real-time streaming LinkID \n");
memset (& LIDQuery, 0, sizeof (MediaLinkIDQueryResult));
iResult = VideoNetClient_QueryStreamMediaLinkID (hUser, eMediaFunctionRealStream, &
LIDQuery);
if (iResult)
{
    printf("[RealStreamMan] LinkID live stream query fails, the return (%d) \n", iResult);
} Else
{
    int i;
    printf("[RealStreamMan] query results LinkID (%d): \n", LIDQuery.dwNumber);
    for (i = 0; i < LIDQuery.dwNumber; i++)
    {
        printf ("%2x", LIDQuery.dwLinkID [i]);
        if (! (i & 7))
        {
            printf ("\n");
        }
    }
    printf ("\n");
}
// Stop Live Stream
printf("[RealStreamMan] stop the live stream \n");
iResult = VideoNetClient_StreamMediaControl (hRealStream [0], eTaskStop);
if (iResult)
{
    printf("[RealStreamMan] live stream [0] to stop failed, return (%d) \n", iResult);
}
iResult = VideoNetClient_StreamMediaControl (hRealStream [1], eTaskStop);
if (iResult)
{
    printf("[RealStreamMan] live stream [1] failed to stop, return (%d) \n", iResult);
}
// End of the operation, disconnect the live stream
printf("[RealStreamMan] disconnect live stream \n");
iResult = VideoNetClient_RealStreamDisconnect (hRealStream [0]);
if (iResult)
{
    printf("[RealStreamMan] live stream [0] disconnect the connection fails, the return (%d) \n",
iResult);
}
iResult = VideoNetClient_RealStreamDisconnect (hRealStream [1]);
if (iResult)
{
    printf("[RealStreamMan] live stream [1] disconnect the connection fails, the return (%d) \n",
iResult);
}
printf("[RealStreamMan] live streaming operation is complete \n");
return 0;
}
int main ()
{
    ...
    //////////////////////////////////////
    // Has landed successfully, can support all operations.
    TimeCfg (hUser);
    // Live Stream
    RealStreamMan (hUser);
    // Do something
    //////////////////////////////////////
    // Operation is complete, ready to cancel, exit
```



```
...  
}
```

6.2.4 Voice talk

```
// Voice intercom  
int CALLBACK CallBack_StreamVoice (IN HSTREAM hStream, IN eVoiceStreamSource eSource, IN const  
StreamVoiceFrame * cFrame, IN DWORD dwUserData)  
{  
    if (eSource == eVoiceStreamCapture)  
    {  
        printf ("[CallBack_StreamVoice] to capture audio hStream = 0x% x, eSource =% d,  
Frame_eSource =% d, userdate = 0x% x \n", hStream,  
eSource, cFrame-> eVoiceSource, dwUserData);  
        // Send voice  
        VideoNetClient_VoiceStreamSend (hStream, & cFrame-> cFrameBuffer);  
    } Else  
    {  
        printf ("[CallBack_StreamVoice] received audio hStream = 0x% x, eSource =% d,  
Frame_eSource =% d, userdate = 0x% x \n", hStream,  
eSource, cFrame-> eVoiceSource, dwUserData);  
    }  
  
    return 0;  
}  
int CaptureVoice ()  
{  
    // Voice capture circuit:  
    // 1. Settings to capture the voice callback CallBack_StreamVoice  
    // 2. Capture voice input wave  
    // 3. Using G.726 speech coding standard  
    // 4. Calls CallBack_StreamVoice to capture the source type to send voice  
  
    return 0;  
}  
int VoiceMan (HUSER hUser)  
{  
    HSTREAM hVoice = -1;  
    int iResult;  
    VoiceStreamPara vsp;  
    memset (& vsp, 0, sizeof (VoiceStreamPara));  
    vsp.dwChannel = 0 ;// first channel  
    vsp.eTransferMode = eGeneralTCP;  
    vsp.eMode = eVoiceG726;  
    printf ("[VoiceMan] began to voice connections \n");  
    iResult = VideoNetClient_VoiceStreamConnect (& hVoice, hUser, & vsp);  
    if (iResult)  
    {  
        printf ("[VoiceMan] voice connection fails, the return (%d) \n", iResult);  
        return -1;  
    }  
  
    // Set the Voice Callback  
    printf ("[VoiceMan] set up voice callback \n");  
    iResult = VideoNetClient_VoiceStreamCB (hVoice, CallBack_StreamVoice, 0);  
    if (iResult)  
    {  
        printf ("[VoiceMan] set up voice callback failed \n");  
    }  
  
    // Activate voice  
    printf ("[VoiceMan] activate voice intercom \n");  
    iResult = VideoNetClient_StreamMediaControl (hVoice, eTaskStart);  
    if (iResult)  
    {
```

```
        printf("[VoiceMan] voice intercom failed to start, return (%d)\n", iResult);
    }
// Local Voice Capture
    CaptureVoice ();
    printf("[VoiceMan] voice intercom run one second and then turn off\n");
    Sleep (1000);
// Stop Voice Intercom
    printf("[VoiceMan] stop the voice intercom\n");
    iResult = VideoNetClient_StreamMediaControl (hVoice, eTaskStop);
    if (iResult)
    {
        printf("[VoiceMan] voice intercom stop fails, the return (%d)\n", iResult);
    }
    printf("[VoiceMan] disconnected voice connections\n");
    iResult = VideoNetClient_VoiceStreamDisconnect (hVoice);
    if (iResult)
    {
        printf("[VoiceMan] disconnected voice failed, return (%d)\n", iResult);
    }
    return 0;
}
int main ()
{
    ...
    //////////////////////////////////////
// Has landed successfully, can support all operations.
    TimeCfg (hUser);
// Live Stream
    RealStreamMan (hUser);
// Voice intercom
    VoiceMan (hUser);
        // Do something
    //////////////////////////////////////
// Operation is complete, ready to cancel, exit
    ...
}
```

```
// Callback historical flow data
int CALLBACK CallBack_StreamMedia (IN HSTREAM hStream, IN const StreamMediaFrame
*cFrame, IN DWORD dwUserData)
{
    printf("[CallBack_StreamMedia] get the historical flow data hStream = 0x%08x, ch=%d, FrameType=%d, \
        Time=%04d-%02d-%02d%02d:%02d:%02d, Userdate = 0x%08x\n",
        hStream,
        cFrame-> dwChannel,
        cFrame-> dwFrameType,
        cFrame-> cFrameTime.wYear + 1900,
        cFrame-> cFrameTime.wMonth + 1,
        cFrame-> cFrameTime.wDay,
        cFrame-> cFrameTime.wHour,
        cFrame-> cFrameTime.wMinute,
        cFrame-> cFrameTime.wSecond, dwUserData);
    return 0;
}
// History stream operations
int HistoryMan (HUSER hUser)
{
    HSTREAM_QUERY hStreamQuery = -1 ;// historical stream query
```

```
HSTREAM hStream = -1 ;// historical flow
HistoryStreamQueryFactor hsqf;
HistoryStreamQueryResult StreamQueryResult, LastQueryResult;
HistoryStreamPara StreamPara;
int iResult;
int iYear, iDay, iMonth, iHour, iMin, iSec;
int bQueryResult = 0 ;// record is a valid query results
char * strStart = "20091105093000";
char * strEnd = "20091105183000";
// Query history stream fragment
memset (& hsqf, 0, sizeof (HistoryStreamQueryFactor));
hsqf.dwChannel = 0 ;// first channel
hsqf.dwDiskGroup = eDiskGroupNormal ;// ordinary disk group
hsqf.eStreamType = eAllStreamMedia ;// all media types
sprintf (hsqf.cBeginTime, "% s", strStart);
sprintf (hsqf.cEndTime, "% s", strEnd);
// Connect query fragments of history flow
printf ("[HistoryMan] query history streaming clips, the time range (% s-% s) \ n",
strStart, strEnd);
iResult = VideoNetClient_HistoryStreamQueryConnect (& hStreamQuery, hUser, &
hsqf);
if (iResult)
{
    printf ("[HistoryMan] historical stream fragment query connection fails,
return% d \ n", iResult);
    return iResult;
}
memset (& LastQueryResult, 0, sizeof (HistoryStreamQueryResult));
// Get query results
while (1)
{
    iResult = VideoNetClient_HistoryStreamQueryNext (hStreamQuery, &
StreamQueryResult);
    if (iResult == eQueryFinished)
    {
        printf ("[HistoryMan] query returns completed \ n");
        break;
    } Else if (iResult == eQueryBusy)
    {
        printf ("[HistoryMan] query returns busy, retrying \ n");
        continue;
    } Else if (iResult != EQueryOK)
    {
        printf ("[HistoryMan] query returns an error \ n");
        break;
    }
    printf ("[HistoryMan] query results ch=% d, type=% d, time:% s -% s \ n",
        StreamQueryResult.dwChannel, StreamQueryResult.eStreamType,
StreamQueryResult.cBeginTime, StreamQueryResult.cEndTime);
    // Here, as an example, recording only the last fragment for subsequent flow
created history
        memcpy (& LastQueryResult, & StreamQueryResult, sizeof
(HistoryStreamQueryResult));
        bQueryResult = 1 ;// valid results
    }

// Fragment inquiry ends, disconnect
```

```
printf("[HistoryMan] historical stream fragment query is finished, disconnect \n");
iResult = VideoNetClient_HistoryStreamQueryDisconnect(hStreamQuery);
if (iResult)
{
    printf("[HistoryMan] Query disconnected fragments of historical stream fails,
return% d \n", iResult);
    return iResult;
}

if (! bQueryResult)
{
    printf("[HistoryMan] query specified period of time not to fragment, the
subsequent history of stream operations no longer \n");
    return 0;
}

// Start the flow of history created for playback or download
// Start creating the historical flow
memset(& StreamPara, 0, sizeof(HistoryStreamPara));
StreamPara.dwChannel = LastQueryResult.dwChannel;
StreamPara.dwDiskGroup = hsqf.dwDiskGroup ;// query fragments corresponding disk
group
StreamPara.eStreamType = LastQueryResult.eStreamType ;// corresponding fragment
inquiry

StreamPara.eTransferMode = eGeneralTCP;
StreamPara.dwEnableEndTime = 1;
sscanf (LastQueryResult.cBeginTime, "% 04d% 02d% 02d% 02d% 02d% 02d", &
iYear, & iMonth, & iDay, & iHour, & iMin, & iSec);
    StreamPara.cBeginTime.wYear = iYear;
    StreamPara.cBeginTime.wMonth = iMonth;
    StreamPara.cBeginTime.wDay = iDay;
    StreamPara.cBeginTime.wHour = iHour;
    StreamPara.cBeginTime.wMinute = iMin;
    StreamPara.cBeginTime.wSecond = iSec;
    sscanf (LastQueryResult.cEndTime, "% 04d% 02d% 02d% 02d% 02d% 02d", & iYear,
& iMonth, & iDay, & iHour, & iMin, & iSec);
    StreamPara.cEndTime.wYear = iYear;
    StreamPara.cEndTime.wMonth = iMonth;
    StreamPara.cEndTime.wDay = iDay;
    StreamPara.cEndTime.wHour = iHour;
    StreamPara.cEndTime.wMinute = iMin;
    StreamPara.cEndTime.wSecond = iSec;

printf("[HistoryMan] History stream created: TCP, ch =% d, DiskGroup =% d,
StreamType =% d, Time = <% s -% s> \n",
    StreamPara.dwChannel, StreamPara.dwDiskGroup, StreamPara.eStreamType,
LastQueryResult.cBeginTime, LastQueryResult.cEndTime);
iResult = VideoNetClient_HistoryStreamCreate (& hStream, hUser, & StreamPara, 0);
if (iResult)
{
    printf("[HistoryMan] historical stream creation failed, returned% d \n",
iResult);
    return iResult;
}

// Set the historical flow data callback
```

```
printf ("[HistoryMan] historical flow data set callback \n");
iResult = VideoNetClient_HistoryStreamCB (hStream, CallBack_StreamMedia, 0);
if (iResult)
{
    printf ("[HistoryMan] historical flow data set callback failed, return (% d) \n",
iResult);
}
// Media Control
printf ("[HistoryMan] historical stream open \n");
iResult = VideoNetClient_StreamMediaControl (hStream, eTaskStart);
if (iResult)
{
    printf ("[HistoryMan] failed to open stream historical returns (% d) \n",
iResult);
}

printf ("[HistoryMan] historical stream running 500 ms \n");
Sleep (500);
printf ("[HistoryMan] set the historical flow 16X \n");
iResult = VideoNetClient_HistoryStreamSetSpeed (hStream, 16);
if (iResult)
{
    printf ("[HistoryMan] historical flow speed settings fail, return (% d) \n",
iResult);
}
printf ("[HistoryMan] historical flow continues to operate 500 ms \n");
Sleep (500);
// Location to begin
printf ("[HistoryMan] historical stream relocation to:% s \n",
LastQueryResult.cBeginTime);
iResult = VideoNetClient_HistoryStreamPosition (hStream, eOperationSet, &
StreamPara.cBeginTime);
if (iResult)
{
    printf ("[HistoryMan] failed to locate the historical flow, return (% d) \n",
iResult);
}
printf ("[HistoryMan] historical flow continues to operate 500 ms \n");
Sleep (500);

// Stop the flow of history
printf ("[HistoryMan] to stop the flow of history \n");
iResult = VideoNetClient_StreamMediaControl (hStream, eTaskStop);
if (iResult)
{
    printf ("[HistoryMan] failed to stop the flow of history, the return (% d) \n",
iResult);
}
// Destroy the historical flow
printf ("[HistoryMan] historical flow destroyed \n");
iResult = VideoNetClient_HistoryStreamDestroy (hStream);
if (iResult)
{
    printf ("[HistoryMan] failed to destroy the historical returns (% d) \n", iResult);
    return iResult;
}
return 0;
```

```
}

int main ()
{
...
// Has landed successfully, can support all operations.
TimeCfg (hUser);
// Live Stream
RealStreamMan (hUser);
// Voice intercom
VoiceMan (hUser);
// History stream operations
HistoryMan (hUser);
// Do something
// Operation is complete, ready to cancel, exit
...
}
```

```
// Log query
int LogMan (HUSER hUser)
{
    int iResult;
    HLOG_QUERY hLogQuery;
    LogQueryFactor Lqf;
    LogQueryResult lqResult;
    // Log the connection

    memset (& Lqf, 0, sizeof (LogQueryFactor));
    Lqf.m_dwChannel = 0;
    Lqf.m_eMajorType = eHistoryLogMajorAll;
    sprintf (Lqf.m_sStartTime, "20091101000000");
    sprintf (Lqf.m_sStopTime, "20091111000000");
    printf ("[LogMan] to start the connection log <% s -% s> \n", Lqf.m_sStartTime,
Lqf.m_sStopTime);
    iResult = VideoNetClient_HistoryLogQueryConnect (& hLogQuery, hUser, & Lqf);
    if (iResult)
    {
        printf ("[LogMan] connection logs fails, the return (% d) \n", iResult);
        return -1;
    }
    while (1)
    {
        iResult = VideoNetClient_HistoryLogQueryNext (hLogQuery, & lqResult);
        if (iResult == eQueryBusy)
        {
            printf ("[LogMan] Query busy, retry \n!");
            continue;
        } Else if (iResult == eQueryFinished)
        {
            printf ("[LogMan] inquiry is complete, exit the loop \n!");
            break;
        } Else if (iResult == eQueryFailed)
        {
            printf ("[LogMan] query fails, exit the loop \n!");
            break;
        }
    }
}
```

```

        }
        printf ("[LogMan] query result:% s (% s)% s, major = 0x% x, minor = 0x% x,
detail = 0x% x \ n",
                ?  lqResult.m_sUserName,  strlen  (lqResult.m_sUserIP)
lqResult.m_sUserIP:      "Local",      lqResult.m_sLogTime,      lqResult.m_eMajorType,
lqResult.m_eMinorType, lqResult.m_dwDetailInfo);
    }
    printf ("[LogMan] disconnected log connection \ n");
    iResult = VideoNetClient_HistoryLogQueryDisconnect (hLogQuery);
    if (iResult)
    {
        printf ("[LogMan] log disconnect failed \ n!");
        return iResult;
    }
    return 0;
}

int main ()
{
    ...
    //////////////////////////////////////
    // Has landed successfully, can support all operations.
    TimeCfg (hUser);
    // Live Stream
    RealStreamMan (hUser);
    // Voice intercom
    VoiceMan (hUser);
    // History stream operations
    HistoryMan (hUser);
    // Log operations
    LogMan (hUser);
    // Do something
    //////////////////////////////////////
    // Operation is complete, ready to cancel, exit
    ...
}

```

6.2.7 File Query

```

// File Query
int FileQueryMan (HUSER hUser)
{
    int iResult;
    HFILE_QUERY hFileQuery;
    FileQueryFactor fqf;
    FileQueryResult fqResult;
    // File Query Connection

    memset (& fqf, 0, sizeof (LogQueryFactor));
    fqf.dwChannel = 0;
    fqf.dwFileType = eImage;
    sprintf (fqf.cBeginTime, "20091101000000");
    sprintf (fqf.cEndTime, "20091111000000");
    printf ("[FileQueryMan] began to check the connection file <% s -% s> \ n",

```

```
fqf.cBeginTime, fqf.cEndTime);
    iResult = VideoNetClient_FileQueryConnect (& hFileQuery, hUser, & fqf);
    if (iResult)
    {
        printf ("[FileQueryMan] connection file check fails, the return (% d) \ n",
iResult);
        return -1;
    }
    while (1)
    {
        iResult = VideoNetClient_FileQueryNext (hFileQuery, & fqResult);
        if (iResult == eQueryBusy)
        {
            printf ("[FileQueryMan] Query busy, retry \ n!");
            continue;
        } Else if (iResult == eQueryFinished)
        {
            printf ("[FileQueryMan] inquiry is complete, exit the loop \ n!");
            break;
        } Else if (iResult == eQueryFailed)
        {
            printf ("[FileQueryMan] query fails, exit the loop \ n!");
            break;
        }
        printf ("[FileQueryMan] query results: CH% 02d-type (% d),% s, bLock (% d),
size (% d), name <% s> \ n",
            fqResult.dwChannel, fqResult.dwFileType, fqResult.cCreateTime,
fqResult.dwLock, fqResult.dwDataSize, fqResult.cFileName);
    }
    printf ("[FileQueryMan] disconnect file query connection \ n");
    iResult = VideoNetClient_FileQueryDisconnect (hFileQuery);
    if (iResult)
    {
        printf ("! [FileQueryMan] file queries disconnect failed \ n");
        return iResult;
    }
    return 0;
}
```

```
int main ()
```

```
{
```

```
...
```

```
////////////////////////////////////
// Has landed successfully, can support all operations.
```

```
TimeCfg (hUser);
```

```
// Live Stream
```

```
RealStreamMan (hUser);
```

```
// Voice intercom
```

```
VoiceMan (hUser);
```

```
// History stream operations
```

```
HistoryMan (hUser);
```

```
// Log operations
```

```
LogMan (hUser);
```

```
// File Query
```

```
FileQueryMan (hUser);
```

```
// Do something
```

```
////////////////////////////////////
```



```
// Operation is complete, ready to cancel, exit
...
}
```

6.2.8 Transparent Channel Operation

```
// Clear Channel Management
int CALLBACK CallBack_TransparentChannel (IN HTRANSPARENT hTransparent, IN const
LPBuffer pBuffer, IN DWORD dwUserData)
{
    printf ("[CallBack_TransparentChannel] Clear Channel received string <% s> \n",
pBuffer-> pBuffer);
    return 0;
}

int TransparentMan (HUSER hUser)
{
    int iResult;
    HTRANSPARENT hTransparent;
    TransparentChannelPara tscp;
    Buffer buff;
    char strWrite [1024];
    printf ("[TransparentMan] began transparent channel connection <first channel,
TTY422> \n");
    memset (& tscp, 0, sizeof (TransparentChannelPara));
    tscp.dwMajorType = eTTY422;
    tscp.dwMinorType = 1 ;// second channel
    iResult = VideoNetClient_TransparentChannelConnect (& hTransparent, hUser, & tscp);
    if (iResult)
    {
        printf ("[TransparentMan] transparent channel connection fails, the return (% d)
\n", iResult);
        return -1;
    }
    iResult = VideoNetClient_TransparentChannelCB (hTransparent,
CallBack_TransparentChannel, 0);
    if (iResult)
    {
        printf ("[TransparentMan] Clear Channel set a callback fails, the return (% d) \n", iResult);
    }
    sprintf (strWrite, "transparent channel to send the string, abcdefg");
    buff.pBuffer = strWrite;
    buff.dwBufLen = strlen (strWrite);
    iResult = VideoNetClient_TransparentChannelWrite (hTransparent, & buff);
    if (iResult)
    {
        printf ("[TransparentMan] transparent channel to send the string fails, the return
(% d) \n", iResult);
    }
    printf ("[TransparentMan] Wait 10 seconds \n");
    Sleep (10000);
    printf ("[TransparentMan] disconnected transparent channel \n");
    iResult = VideoNetClient_TransparentChannelDisconnect (hTransparent);
}
```

```
        if (iResult)
        {
            printf ("[TransparentMan] Clear Channel disconnect failed, return (%d) \n",
iResult);
            return -1;
        }
        printf ("[TransparentMan] Clear Channel completed \n");
        return 0;
    }

int main ()
{
    ...
    //////////////////////////////////////
    // Has landed successfully, can support all operations.
    TimeCfg (hUser);
    // Live Stream
    RealStreamMan (hUser);
    // Voice intercom
    VoiceMan (hUser);
    // History stream operations
    HistoryMan (hUser);
    // Log operations
    LogMan (hUser);
    // File Query
    FileQueryMan (hUser);
    // Transparent channel operation
    TransparentMan (hUser);
    // Do something
    //////////////////////////////////////
    // Operation is complete, ready to cancel, exit
    ...
}
```

6.2.9 File download

```
// File Download
static int m_iTotalSize = 0 ;// Download total file size
static int m_iSize = 0 ;// size of the downloaded file
static int m_bFinished = 0 ;// label download is complete
static int CALLBACK CallBack_FileDownload (IN HFILE_TRANSFER hFileTransfer, IN const
Buffer cBuffer, IN DWORD dwUserData)
{
    m_iSize += cBuffer.dwBufLen;
    if (cBuffer.dwBufLen == 0)
    {
        printf ("[CallBack_FileDownload] Download Complete <%d> \n!", m_iSize);
        m_bFinished = 1;
    } Else
    {
        printf ("[CallBack_FileDownload] Downloaded%8d \n.", m_iSize);
    }
    return 0;
}
```

```
int FileDownloadMan (HUSER hUser)
{
    HFILE_TRANSFER hStranFile;
    FileDownloadPara fdp;
    int iResult;

    m_bFinished = 0;
    m_iSize = 0;
    m_iTotalSize = 0;
    memset (& fdp, 0, sizeof (FileDownloadPara));
    sprintf (fdp.strLocalFilePath, "c: \\ Edvr.cfg");
    sprintf (fdp.strRemoteFilePath, "Edvr.cfg");
    fdp.eFileType = eDeviceConfig;
    printf ("[FileDownloadMan] start connecting Download <% s ->% s> \ n",
fdp.strRemoteFilePath, fdp.strLocalFilePath);
    iResult = VideoNetClient_FileDownloadConnect (& hStranFile, hUser, & fdp);
    if (iResult)
    {
        printf ("[FileDownloadMan] file download connection fails return (% d) \ n!",
iResult);
        return -1;
    }
    printf ("[FileDownloadMan] Set Download callback \ n");
    iResult = VideoNetClient_FileDownloadCB (hStranFile, CallBack_FileDownload, 0);
    if (iResult)
    {
        printf ("! [FileDownloadMan] settings file downloads callback failed to return
(% d) \ n", iResult);
    }
    printf ("[FileDownloadMan] Start download \ n");
    iResult = VideoNetClient_FileDownloadControl (hStranFile, eFileTransferStart);
    if (iResult)
    {
        printf ("[FileDownloadMan] file download control [starts] failed to return (% d)
\ n!", iResult);
    }
    printf ("[FileDownloadMan] file download ... \ n");
    while (1)
    {
        if (m_bFinished)
        {
            break;
        }
        Sleep (1000);
    }

    printf ("[FileDownloadMan] finished, disconnect the download \ n");
    iResult = VideoNetClient_FileDownloadDisconnect (hStranFile);
    if (iResult)
    {
        printf ("! [FileDownloadMan] file download disconnect failed to return (% d) \
n", iResult);
        return -1;
    }

    printf ("[FileDownloadMan] file download is complete ... \ n");
    return 0;
}
```

```
}

int main ()
{
...
/////////////////////////////////////////////////////////////////
// Has landed successfully, can support all operations.
TimeCfg (hUser);
// Live Stream
RealStreamMan (hUser);
// Voice intercom
VoiceMan (hUser);
// History stream operations
HistoryMan (hUser);
// Log operations
LogMan (hUser);
// File Query
FileQueryMan (hUser);
// Transparent channel operation
TransparentMan (hUser);
// File Download
FileDownloadMan (hUser);
// Do something
/////////////////////////////////////////////////////////////////
// Operation is complete, ready to cancel, exit
...
}
```

6.2.10 File upload operation

```
// File Upload
static int CALLBACK CallBack_FileUpload (IN HFILE_TRANSFER hFileTransfer, IN const
FileUploadState cState, IN DWORD dwUserData)
{
    /*! <Upload status, 0 is being sent, one for the cancellation,          */
    /*! <      Flash 2 full, 3 is the wrong version or file errors, 4 indicating a write failure,
5 indicates successful completion, six said transmission fails, 7 represents Error reading from
file          */
    if (cState.dwStatus == 5)
    {
        printf ("[CallBack_FileUpload] Upload complete <% d> \ n!",
cState.dwUploadSize);
        m_bFinished = 1;
    } Else if (cState.dwStatus == 0)
    {
        printf ("[CallBack_FileUpload] Uploaded% 8d \ n.", cState.dwUploadSize);
    } Else if (cState.dwStatus == 1)
    {
        printf ("[CallBack_FileUpload] upload cancel% 8d \ n.", cState.dwUploadSize);
    }
    else
    {
        printf ("[CallBack_FileUpload] Upload failed% 8d \ n.", cState.dwUploadSize);
    }
}
```

```
        return 0;
    }
    int FileUploadMan (HUSER hUser)
    {
        FileUploadPara fup;
        HFILE_TRANSFER hStranFile;
        int iResult;

        m_bFinished = 0;
        m_iSize = 0;
        m_iTotalSize = 0;

        memset (& fup, 0, sizeof (FileUploadPara));
        sprintf (fup.strLocalFilePath, "c: \\ Edvr.cfg");
        sprintf (fup.strRemoteFilePath, "Edvr.cfg");
        fup.eFileType = eDeviceConfig;
        printf ("[FileUploadMan] start connecting uploading <% s <-% s> \\ n",
fup.strRemoteFilePath, fup.strLocalFilePath);
        iResult = VideoNetClient_FileUploadConnect (& hStranFile, hUser, & fup);
        if (iResult)
        {
            printf ("[FileUploadMan] file upload connection failed return (% d) \\ n!",
iResult);
            return -1;
        }
        printf ("[FileUploadMan] Set upload callback \\ n");
        iResult = VideoNetClient_FileUploadCB (hStranFile, CallBack_FileUpload, 0);
        if (iResult)
        {
            printf ("[FileUploadMan] failed to set file upload callback return (% d) \\ n!",
iResult);
        }
        printf ("[FileUploadMan] start uploading \\ n");
        iResult = VideoNetClient_FileUploadControl (hStranFile, eFileTransferStart);
        if (iResult)
        {
            printf ("[FileUploadMan] file upload control [starts] failed to return (% d) \\ n!",
iResult);
        }
        printf ("[FileUploadMan] file upload ... \\ n");
        while (1)
        {
            if (m_bFinished)
            {
                break;
            }
            Sleep (1000);
        }

        printf ("[FileUploadMan] finished, disconnect upload \\ n");
        iResult = VideoNetClient_FileUploadDisconnect (hStranFile);
        if (iResult)
        {
            printf ("[FileUploadMan] file upload disconnect failed to return (% d) \\ n!",
iResult);
            return -1;
        }
    }
}
```

```
        printf("[FileUploadMan] file upload is complete ... \n");
    }

int main ()
{
    ...
    //////////////////////////////////////
    // Has landed successfully, can support all operations.
    TimeCfg (hUser);
    // Live Stream
    RealStreamMan (hUser);
    // Voice intercom
    VoiceMan (hUser);
    // History stream operations
    HistoryMan (hUser);
    // Log operations
    LogMan (hUser);
    // File Query
    FileQueryMan (hUser);
    // Transparent channel operation
        TransparentMan (hUser);
    // File Download
    FileDownloadMan (hUser);
    // File Upload
    FileUploadMan (hUser);
    // Do something
    //////////////////////////////////////
    // Operation is complete, ready to cancel, exit
    ...
}
```

6.2.11 Remote command control

```
char m_strUserEvent [15] [50] = {"alarm event", "heartbeat loss", "network reconnection", "User
disconnected", "streaming off", "Disk Management", "history flow notice" "Real Time Streaming
notice initiates the connection ID"
"Real-time flow stops the connection ID notification", "the voice stream initiates the connection
ID notification," "Stop the voice stream connection ID notification", "history flow Destruction",
"history flow start", "stop the flow of history", "unknown event"} ;

static int m_bGetDataSizeFinished = 0 ;// labeling video data acquisition is complete
// Event notification callback
static int CALLBACK CallBack_UserEvent (IN HUSER hUser, IN DWORD dwEventType,
DWORD dwParam1, DWORD dwParam2, DWORD dwParam3, IN DWORD dwUserData)
{
    int index = 0;
    for (index = 0; index <14; index ++ )
    {
        if ((1 << index) == dwEventType)
        {
            break;
        }
    }
}
```

```

        printf ("\n [CallBack_UserEvent] [%s] Event callback hUser =%d, EventType =%d,
dwParam1 =%d, dwParam2 =%d, dwParam3 =%d, dwUserData =%d \n",
            m_strUserEvent [index], hUser, dwEventType, dwParam1, dwParam2,
dwParam3, dwUserData);

        if (dwEventType == (1 << 6))
        { // History stream notification events
            if (dwParam1 == 1)
            { // Video data size
                if (dwParam3 == -1)
                {
                    printf ("[CallBack_UserEvent] video data size to get wrong,
userdata =%d \n", dwParam2);
                } Else
                {
                    printf ("[CallBack_UserEvent] video data size for success:
size =%dKB, userdata =%d \n", dwParam3, dwParam2);
                }

                m_bGetDataSizeFinished = 1;
            }
        }

        return 0;
    }

// Get the video data size
int DataSizeGet (HUSER hUser)
{
    int chbits;
    char strStartTime [20];
    char strStopTime [20];
    int iResult;
    DWORD dwDataSize;
    m_bGetDataSizeFinished = 0;
    chbits = 3 ;// first and second channel
    sprintf (strStartTime, "20091112000000");
    sprintf (strStopTime, "20091112180000");
    printf ("[DataSizeGet] began to get the video data size <1/2 channel, ordinary disk
group, all video types, time% s -% s> \n", strStartTime, strStopTime);
    iResult = VideoNetClient_GetDataSize (hUser, chbits, eDiskGroupNormal,
eAllStreamMedia, strStartTime, strStopTime, 0, & dwDataSize);
    if (iResult)
    {
        printf("[DataSizeGet] retrieve recording data size failure (%d)\n",iResult);
        return -1;
    }
    printf("[DataSizeGet] waiting for return!\n");
    while (1)
    {
        if (m_bGetDataSizeFinished)
        {
            break;
        }
        Sleep(1000);
    }
    printf("[DataSizeGet] finished !\n");
}

```

```
        return 0;
    }
    int main ()
    {
        ...
        //////////////////////////////////////
        // LOGON, YOU CAN DO EVERYTHING WHAT SUPPORTED
        TimeCfg(hUser);
        // REAL STREAM
        RealStreamMan(hUser);
        / VOICE TALK
        VoiceMan(hUser);
        //HISTROY STREAM
        HistoryMan(hUser);
        //LOG
        LogMan (hUser);
        //QUERY FILE
        FileQueryMan (hUser);
        //TRANSPARENT CHANNEL
            TransparentMan(hUser);
            //DOWNLOAD FILE
            FileDownloadMan(hUser);
            //UPLOAD FILE
            FileUploadMan(hUser);
        //RETRIEVE RECORDING DATA SIZE
            DataSizeGet(hUser);
            //Do something
        //////////////////////////////////////
        //PREPARE LOGOUT AND EXIT
        ...
    }
```