Device Network SDK

Programming User Manual

V4.2

(For IPC)

The information in this documentation is subject to change without notice and does not represent any commitment on behalf of HIKVISION. HIKVISION disclaims any liability whatsoever for incorrect data that may appear in this documentation. The product(s) described in this documentation are furnished subject to a license and may only be used in accordance with the terms and conditions of such license.

This documentation is issued in strict confidence and is to be used only for the purposes for which it is supplied. It may not be reproduced in whole or in part, in any form, or by any means or be used for any other purpose without prior written consent of HIKVISION and then only on the condition that this notice is included in any such reproduction. No information as to the contents or subject matter of this documentation, or any part thereof, or arising directly or indirectly therefrom, shall be given orally or in writing or shall be communicated in any manner whatsoever to any third party being an individual, firm, or company or any employee thereof without the prior written consent of HIKVISION. Use of this product is subject to acceptance of the HIKVISION agreement required to use this product. HIKVISION reserves the right to make changes to its products as circumstances may warrant, without notice.

This documentation is provided "as-is," without warranty of any kind.

Please send any comments regarding the documentation to:
overseasbusiness@hikvision.com

Find out more about HIKVISION at www.hikvision.com

Index

1 SDK Overview

The device network SDK is developed based on private network communication protocol, and it is designed for the remote connection and configuration of embedded devices. This document is mainly for IP camera and IP dome.

The functions supported by the SDK:

1. Live view, playback, remote file download, PTZ control, arm/disarm, voice talk, log query, decoding card function, etc.

2. Remote upgrade, remotely reboot, remotely shut down, remotely format hard disk (SD card), and device configuration (system configuration, channel configuration, serial port configuration, alarm configuration, users configuration), etc.

This document introduces only the major function supported by IPC and IP dome, and please get more information about other function and related structures from "Device Network SDK Programming Manual.chm".

The device network SDK has both Windows and Linux version.

1. Windows version supports Windows7/XP/2000/2003/Vista(32bit), and it has the files:

Network Communication Library	HCNetSDK.h	head file
	HCNetSDK.lib	LIB file
	HCNetSDK.dll	DLL file
Qos Library	QosControl.dll	DLL file
RTSP Communication Library	StreamTransClient.dll	DLL file
Software Decode Library	PlayM4.h	head file
	PlayCtrl.lib	LIB file
	PlayCtrl.dll	DLL file
Encapsulation Transformation Library	SystemTransform.dll	DLL file
Hardware decode Library	DataType.h DecodeCardSdk.h	head file
	DsSdk.lib	LIB file
	DsSdk.dll	DLL file

2. Linux version supports the system(32bit) that gcc-v is 3.4 or above. The tested system have RedHat AS4/5/6, (Fedora)FC8/10/12, CentOS 4/5, SUSE 10, openSUSE 11, and Ubuntu 9.04/10.04. The SDK has the files:

Network Communication Library	hcnetsdk.h	head file
	libhcnetsdk.so	SO file
Qos Library	libQosControl.so	SO file
RTSP Communication Library	libStreamTransClient.so	SO file
Software Decode Library	playsdkpu.h	head file
	libm4play.so	SO file
Encapsulation Transformation Library	libSystemTransform.so	SO file

HCNetSDK is required to be loaded for client development, and the other '.dll' files are optional components.

- The network communication library is the main functional part of the device network SDK. It is used for communication between the client and devices, including remote control & configuration, video stream acquiring and handling, etc; and network communication library will dynamically loading RTSP communication library, Software decoding library, Hardware decoding library, etc. Network communication library combines a lot of functions from the Software decoding library and Hardware decoding library to facilitate the programming work. However, it is suggested the users to get video stream from 'HCNetSDK.dll', and call relative APIs in the Software decoding library or Hardware decoding library directly if you want to build a system with more complete functions, or in a more flexible way.
- The 'QosControl' library is stream bitrate control library, used for push mode SDK.
- RTSP Communication Library only supports IP devices. Users need to load this component for operations like streaming from products which support RTSP protocol.
- Software Decoding Library is used for decoding real-time video stream (remote live view), playback files, etc. It has included standard stream decoding function. If users needs to play real-time stream or recoding data and display(i.e. the second structure parameter play handle of NET_DVR_RealPlay_V30 interface set to effective), must load this component. However, if users just need to use it for capturing data, then do external operation, needn't load this component, this way is more flexible.
- Encapsulation transformation library function can be divided into two pieces: one is converting standard stream data to private encapsulation format stream data. When users need to capture private format stream data from products supporting RTSP protocol(that is setting callback function of NET_DVR_RealPlay_V30 interface for capturing data or call NET_DVR_SetRealDataCallBack interface to capture data), must load this component. Another is converting standard stream data to other package format, such as 3GPP,PS and so on. For example, when users need to capture specific package format real-time stream data from products supporting RTSP protocol(corresponding interface is NET_DVR_SaveRealData), must load this component.
- Hardware Decoding Library can only be used when there is MDI card. For IPC and IP dome, it is not required.

2 SDK Version Update

Version 4.1.0（2012-5-9）

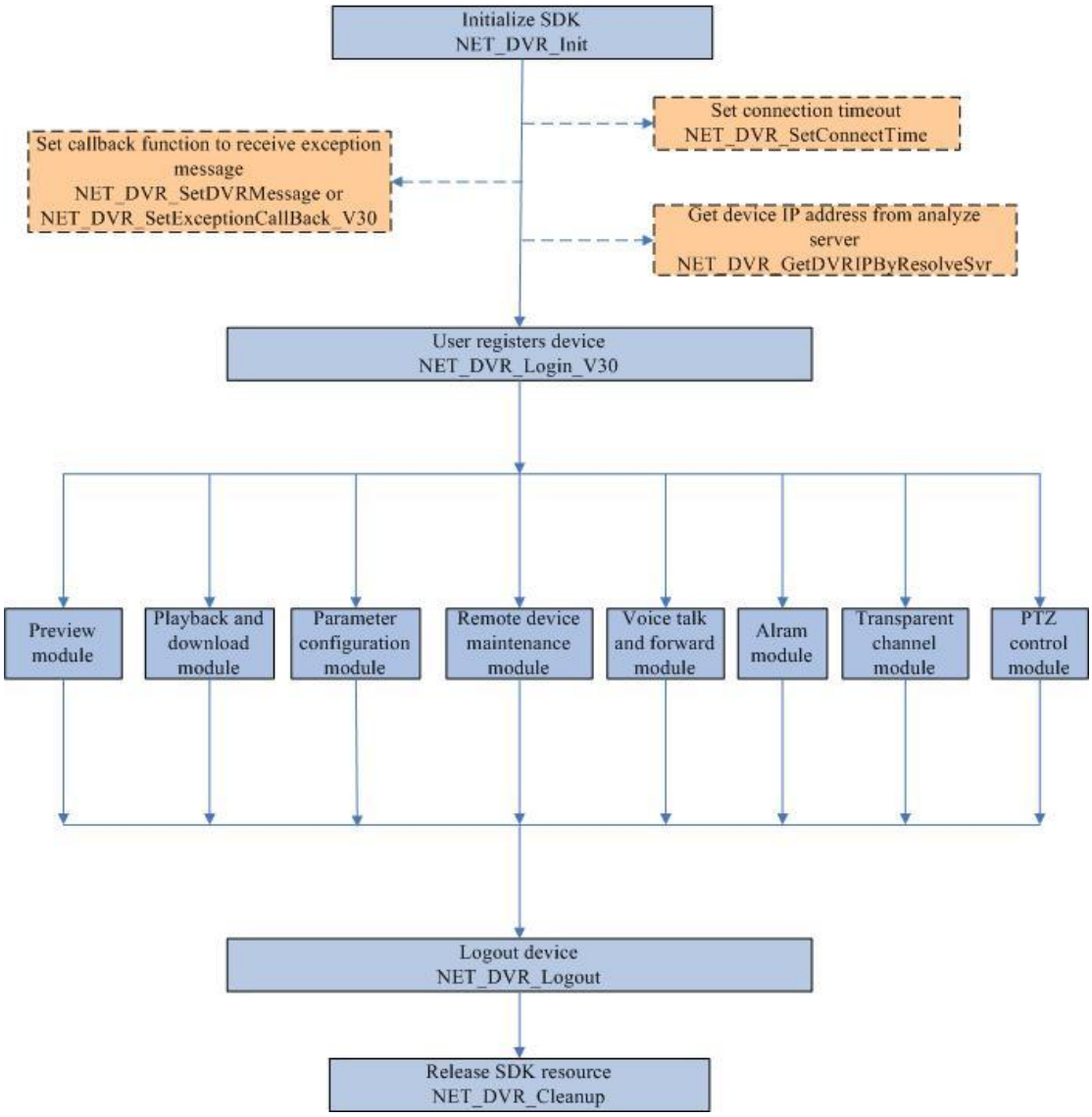
- New APIs of thermal network camera:
  NET_DVR_ShutterCompensation, NET_DVR_CorrectDeadPixel
- New APIs of IPC v4.0
  NET_DVR_FocusOnePush, NET_DVR_ResetLens, NET_DVR_RemoteControl
- New configuration function:
  NET_DVR_AUDIO_INPUT_PARAM, NET_DVR_CAMERA_DEHAZE_CFG, NET_IPC_AUX_ALARMCFG
- New alarm type:
  NET_IPC_AUXALARM_RESULT
- New capability set:
  DEVICE_ALARM_ABILITY

Version 4.1.0（2012-4-5）

- New APIs to encode and decode G726 audio data:
  NET_DVR_InitG726Encoder, NET_DVR_EncodeG726Frame, NET_DVR_ReleaseG726Encoder、
  NET_DVR_InitG726Decoder, NET_DVR_DecodeG726Frame, NET_DVR_ReleaseG726Decoder

3 API Calling Procedure

3.1 Main calling procedure of SDK



The part in dashed box is optional and will not affect the function and use of other process and modules. It can be divided into ten parts by different realization functions. The following four parts: initialize SDK, user register devices, logout and release SDK resource is essential to each module.

- SDK initial(NET_DVR_Init): Initialization of the whole network SDK, operations like memory pre-allocation.
- Set connection timeout (NET_DVR_SetConnectTime):This part is optional, and used to set

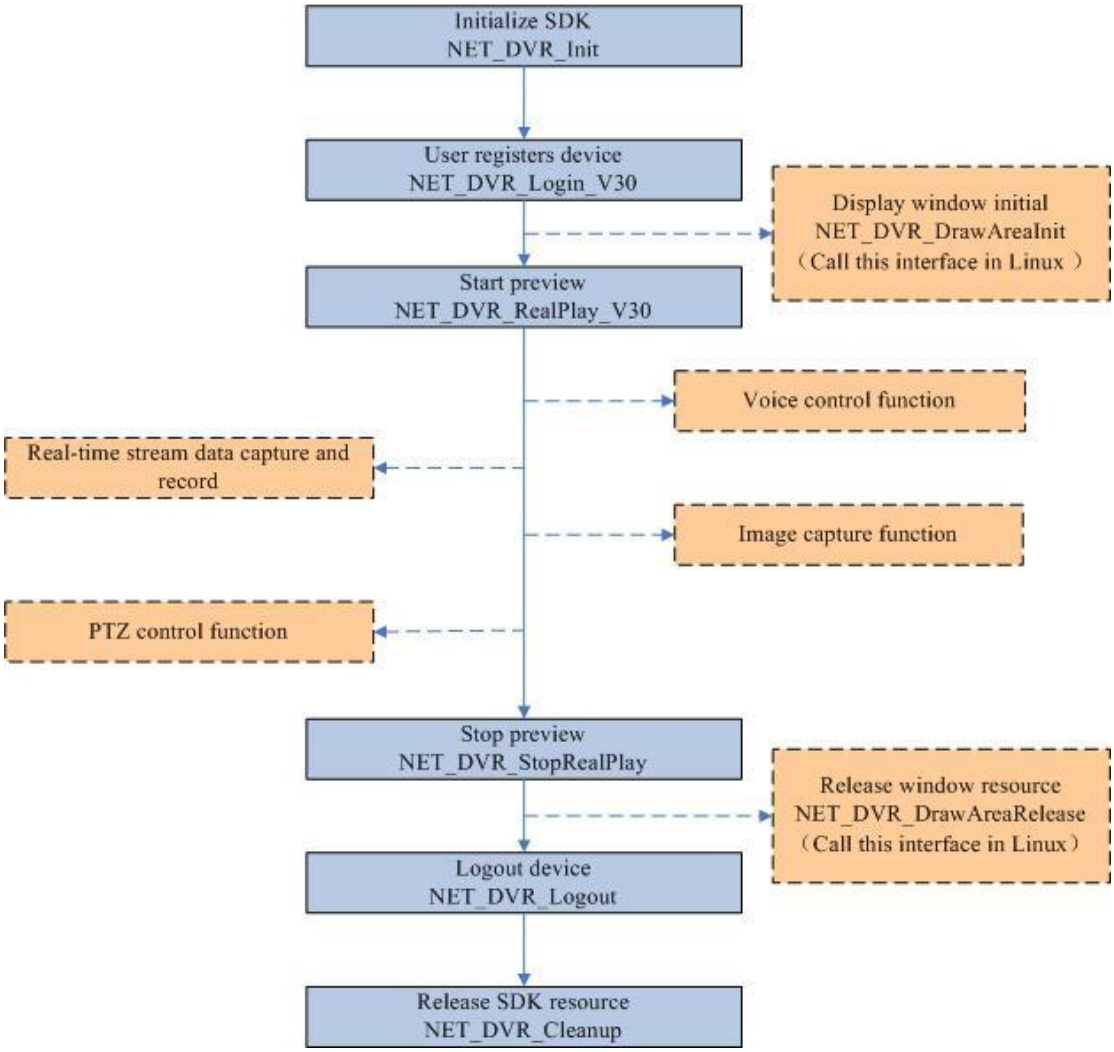
the network connection timeout of SDK. User can set this value to their own needs. You will use the default value when you don't call this interface to set timeout.

- Set reception message callback function(NET_DVR_SetDVRMessage or NET_DVR_SetExceptionCallBack_V30):Most module functions of the SDK are achieved by the asynchronous mode, so we provide this interface for receiving reception message of preview, alarm, playback, transparent channel and voice talk process. Clients can set this callback function after initializing SDK, receive process exception message of each module in application layer.
- Obtain the device IP address from IP analyze server(NET_DVR_GetDVRIPByResolveSvr_Ex): This interface provides a way to obtain device IP address information from IP analyze server when just know the device name and serial number. Such as: the current device obtain a dynamic IP address via dial-up access, and PC runs IPServer software can be aan analyze server, we could input the analyze server IP address, device name and serial number for searching the IP address of this device. IPServer is a domain name analyze server software provided by us.
- User register to device(NET_DVR_Login_V30): Realize user register function,After registering sucessfully,The returned user ID as a Unique identifier for other function operations. The max register users is 512. IPC or IP dome permits 16 register user names and at most 128 user register.
- Preview module:Get real-time stream data from front-end sever, functions like decoding display and play control, and support software and hardware decoding at the same time. See the specific process Live View Module Procedure.
- Playback and download module:Remote playback or download the record files in front-end server by time or file name, then do decoding or storing. Also supports HTTP functionality. See the specific process Playback and Download Module Procedure.
- Parameter configuration module: set and retrieve the parameters of front-end server, including information like device parameters, network parameters, channel compression parameters, serial port parameters, alarm parameters, abnormal parameters, transaction information and user configuration parameters. See Parameter Configuration Module Procedure.
- Remote equipment maintenance module: imlementing trun off the device, restart the device, resotre the default values, format a remote HDD, remote upgrade and configuration file import/export. See Remote Device Maintenance Module Procedure.
- Voice talk and forward module: implement voice talk with front-end and obtain voice data, audio encoding format can be specified. See Voice Talk And Forward Module Procedure.
- Alarm module: handle all kinds of alarm signals uploaded by front-end. Alarm can be devided into two ways into "arm" and "listen", it doesn't require you to do operations like "user register" when using "listen" module and without the need of obtain user ID. See the specific process Alarm Module Procedure.
- Transparent channel module: transparent channel is a technology that analyzing data packets and sent diretly to serial prot. Actually an extension of serail device control in distance. You can use IP network to control serial device, such as decoder, matrix, alarm host,access control, instrumentation and other serial devices, user only see point to point transparent, without concern for network transmission process, so it's called a transparent

serial channel. Network SDK provides 485 and 232 serial ports as transparent channels, you must set 232 work mode to transparent channel in 232 configuration information structure NET_DVR_RS232CFG at first, so that 232 can be used as transparent channel. See the specific process Transparent Channel Module Procedure.

- PTZ control module: To achieve the basic operations of PTZ, preset, cruise, track and transparent PTZ control. SDK will be divided into two modes: one is the handle returned by the image preview control, the other is no limited preview, do PTZ control through user register ID.

3.2 Live view procedure



The modules shown by dotted line is related with preview module, and these interfaces can be called only after starting preview. They are parallel and realize their corresponding function independently.

- Sound control function mainly realizes opening or closing the exclusive or share sound, and volume control. Related API: NET_DVR_OpenSound, NET_DVR_CloseSound, NET_DVR_OpenSoundShare, NET_DVR_CloseSoundShare, NET_DVR_Volume
- Module of real-time stream data capture and record mainly realizes data callback and local

record. Related API: NET_DVR_SetRealDataCallBack, NET_DVR_SetStandardDataCallBack, NET_DVR_SaveRealData.

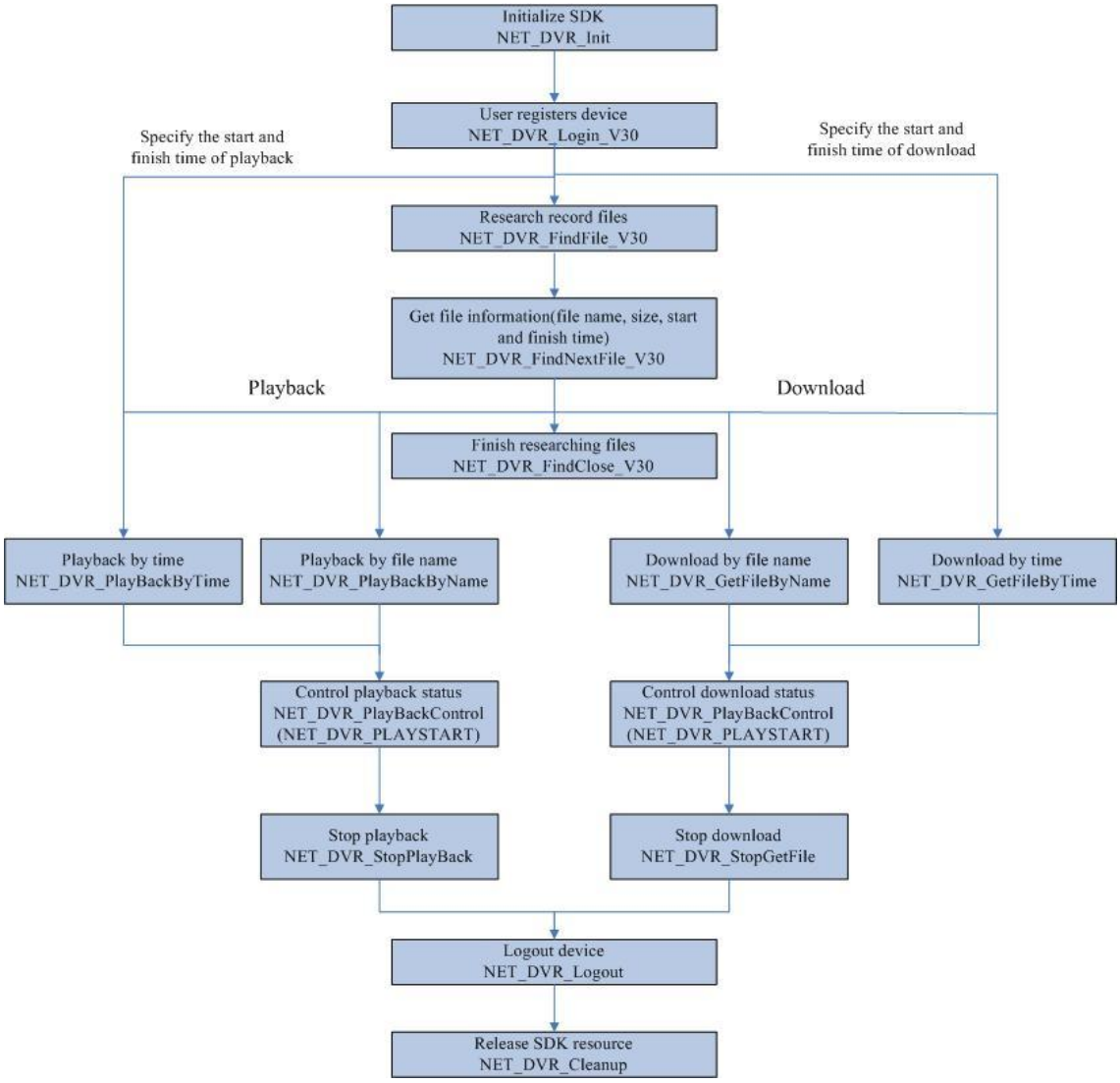
- Picture capture module mainly realizes capturing current decoded image and saving as BMP file. Related API: NET_DVR_CapturePicture.
- PTZ control module mainly realizes operating PTZ control which needs starting preview, including PTZ preset, patrol, pattern and transparent PTZ. Related API: NET_DVR_PTZControl, NET_DVR_PTZControl_EX, NET_DVR_PTZPreset, NET_DVR_PTZPreset_EX, NET_DVR_PTZCruise, NET_DVR_PTZCruise_EX, NET_DVR_PTZTrack, NET_DVR_PTZTrack_EX, NET_DVR_TransPTZ, NET_DVR_TransPTZ_EX.

Decoding method of real-time stream:

- Method 1: If set the handle of play window in preview interface NET_DVR_RealPlay_V30 to be valid handle, the data will be decoded and displayed by SDK: after initailizing SDK and logining device, call directly starting or stopping preview interface.
- Method 2: Users can get stream data to handle by setting the handle of play window in preview interface NET_DVR_RealPlay_V30 to be NULL and calling callback interface(set the callback function in NET_DVR_RealPlay_V30, or call NET_DVR_SetRealDataCallBack or NET_DVR_SetStandardDataCallBack).

Example Code

3.3 Playback and download procedure

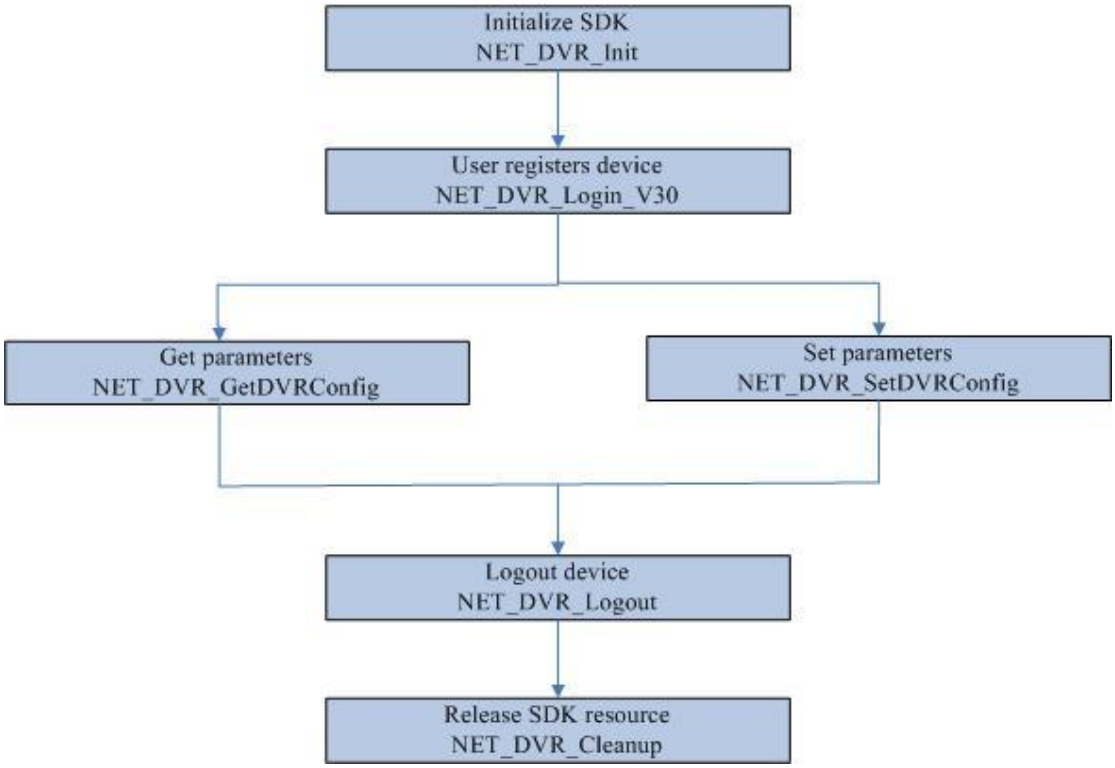


- Playback or download by file need to get file information through reseaching file function at first(regarding interface NET_DVR_FindFile_V30, NET_DVR_FindNextFile_V30),then start playback or download refer to obtained file name(regarding interface NET_DVR_PlayBackByName, NET_DVR_GetFileByName),especially note that you must use start play command(NET_DVR_PLAYSTART) of control interface (NET_DVR_PlayBackControl_V40)after calling playback or download interfaces.
- Playback or download by time, user couldn't call interfaces regarding researching record files. Just need to fix start and finish time of playback or download interface(regarding interface NET_DVR_PlayBackByTime, NET_DVR_GetFileByTime),Then must call start play command(NET_DVR_PLAYSTART)of control interface(NET_DVR_PlayBackControl_V40). At this time, start playback or download within the specified time with record videos in the recent period of time. User can call the relevant interfaces of researching record files, obtain start and finish time of file, and specify the time parameters of playback or download

interfaces in this time range.You must use start play command(NET_DVR_PLAYSTART) of control interface (NET_DVR_PlayBackControl_V40)after calling playback or download interfaces, too.

Example Code

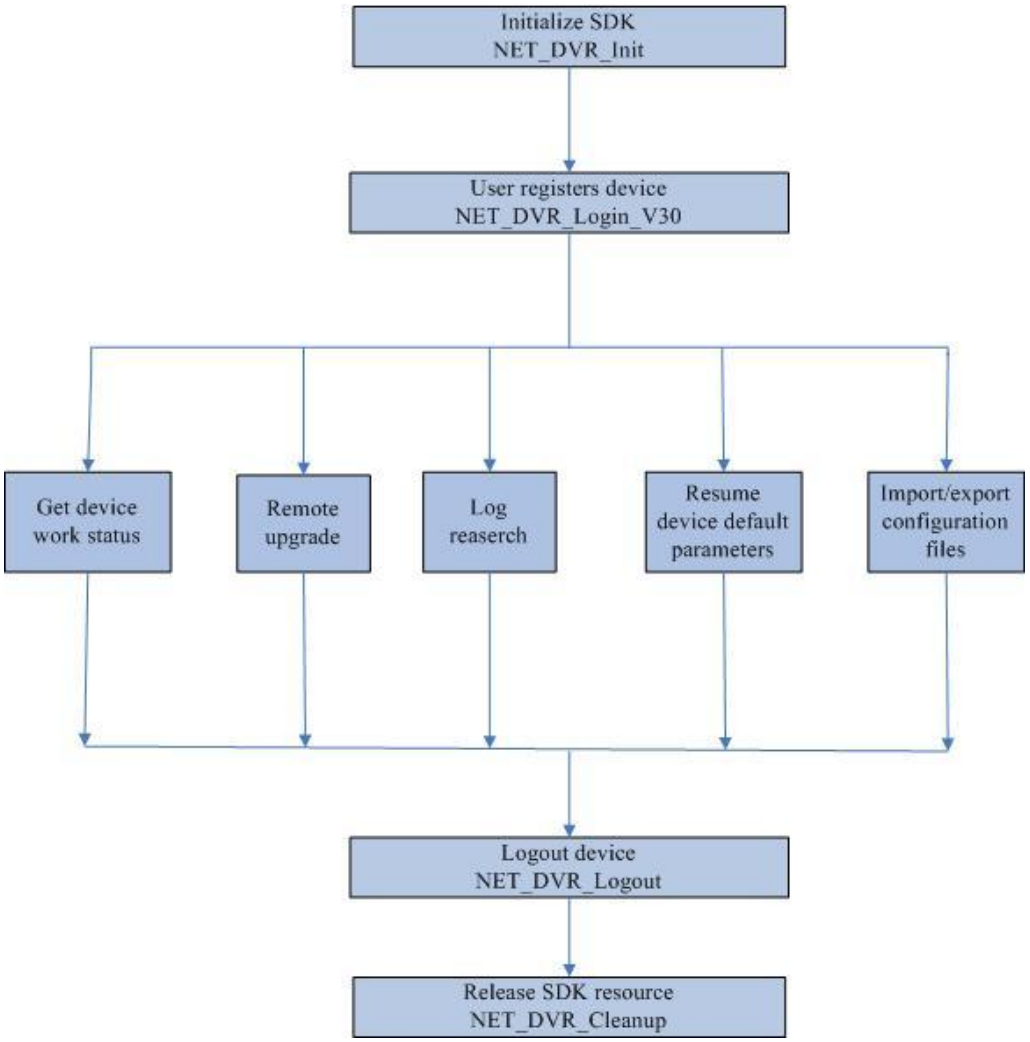
3.4 Parameter configuration procedure



- If you want to do parameters configuration, you must do SDK initialization and user register at first, use the returned ID number as the first parameter of interface configuration. Proposal to call interface(NET_DVR_GetDVRConfig) to get parameters for complete argument structure before setting each certain parameter, modify the parameters need to change, as input parameters for setting parameter interface. At last call setting parameter interface(NET_DVR_SetDVRConfig),Setting successfully if return successfully.

Example Code

3.5 Remote device maintainence procedure

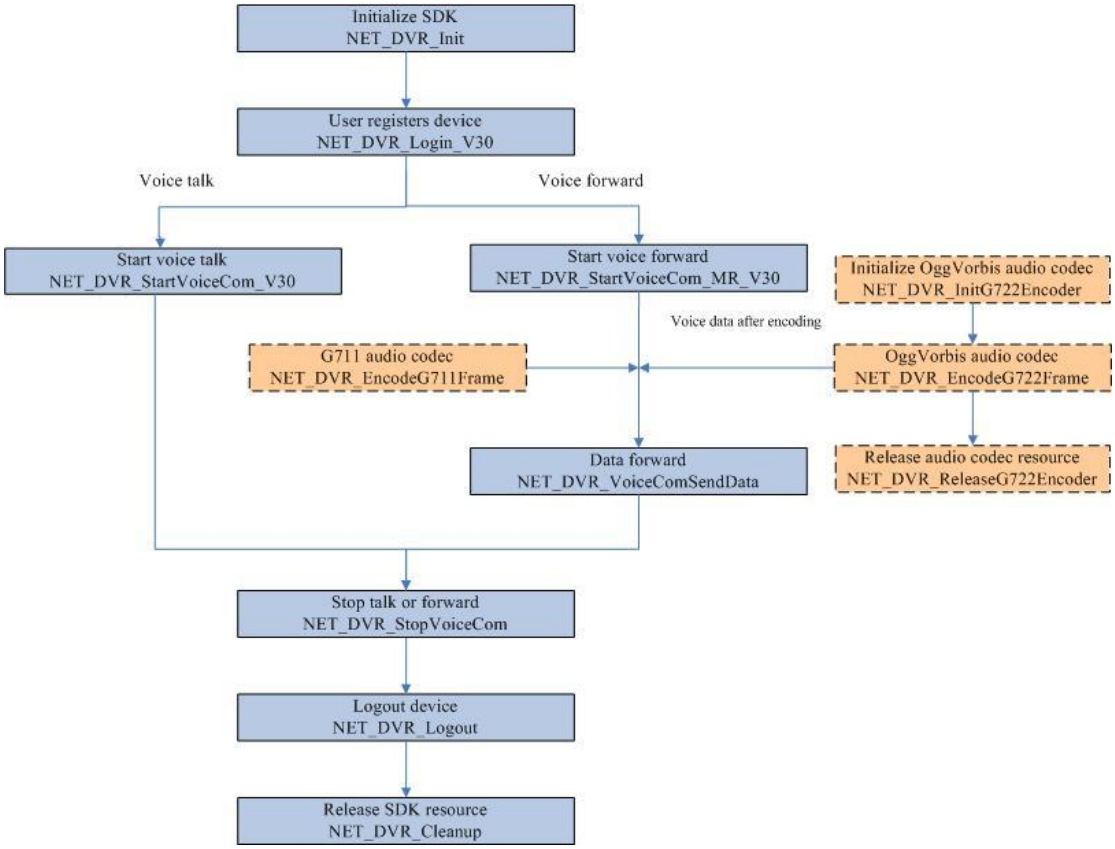


Remote maintenance module for device consists of getting device state, remote upgrade, log query, restoring default settings, and importing & exporting configuration file.

- Get device state: get current state of hard disks, channels, alarm input & output, local display, voice channels, and so on. Related API: NET_DVR_GetDVRWorkState_V30.
- Remote upgrade: upgrade device remotely, and get current progress and state of upgrade. Related API: NET_DVR_Upgrade, NET_DVR_GetUpgradeProgress, NET_DVR_GetUpgradeState.
- Query log: query log message, including alarm, exception, operation, and log with S.M.A.R.T information. Related API: NET_DVR_FindDVRLog_V30, NET_DVR_FindNextLog_V30.
- Restore default configuration for device. Related API: NET_DVR_RestoreConfig.
- Import or export configuration file: export and save all configuration information, or import configuration to the device. Related API: NET_DVR_GetConfigFile_V30, NET_DVR_GetConfigFile, NET_DVR_SetConfigFile_EX, NET_DVR_SetConfigFile.

Example Code

3.6 Voice talk or voice forward procedure



- Voice talk function realizes audio sending and receiving between PC client and device, by calling interface NET_DVR_StartVoiceCom_V30 after device registers successfully. User can set callback function with this interface to get data sent from current device or sample by PC (choose callback encoded or PCM data by requirments).
- Voice forward function realizes forward encoded audio data to device, the steps is as following:
  - Please call NET_DVR_StartVoiceCom_MR_V30 to start voice forward with a device(build connection with the device, wait for sending data at this time).
  - Ready for sending data(need to encode at first), corresponds dotted part of the above image, if data has been handled according the audio compression format, this part could be omitted. Data sources can be collected from the PC sound card, or read from files, but need to compressed by private algorithm, SDK provides a set of coding interfaces:

    If the audio format is G722: 1)initialize audio codec- NET_DVR_InitG722Encoder; 2)G722 audio codec- NET_DVR_EncodeG722Frame,parameters of the interface have certain requirments, please see details from the API description; 3)Please call NET_DVR_ReleaseG722Encoder to release encoding audio resources after all encoding process finished.

    If the auido format is G711: please call NET_DVR_EncodeG711Frame to encoding the audio data directly.

If the audio format is G726: 1)initialize audio codec- NET_DVR_InitG726Encoder; 2)G726 audio codec- NET_DVR_EncodeG726Frame,parameters of the interface have certain requirments, please see details from the API description; 3)Please call NET_DVR_ReleaseG726Encoder to release encoding audio resources after all encoding process finished.

- After the encoding operation, we can get fixed size and encoded data every time, and then call interface NET_DVR_VoiceComSendData to send these data to device. After all forward functions completed, call interface NET_DVR_StopVoiceCom to finish audio forward connection with device.

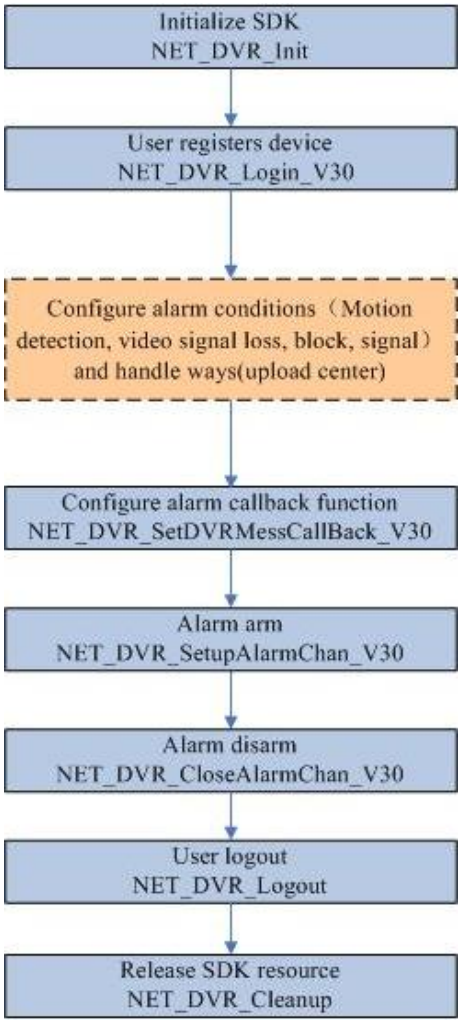
- Linux SDK only supports voice forward function currently, doesn't support voice talk.

Example Code

3.7 Alarm procedure

There are two alarm mode: "arm" and "listen". You can receive information like motion detection alarm, video loss alarm, block alarm and signal occlusion alarm uploaded by devices.

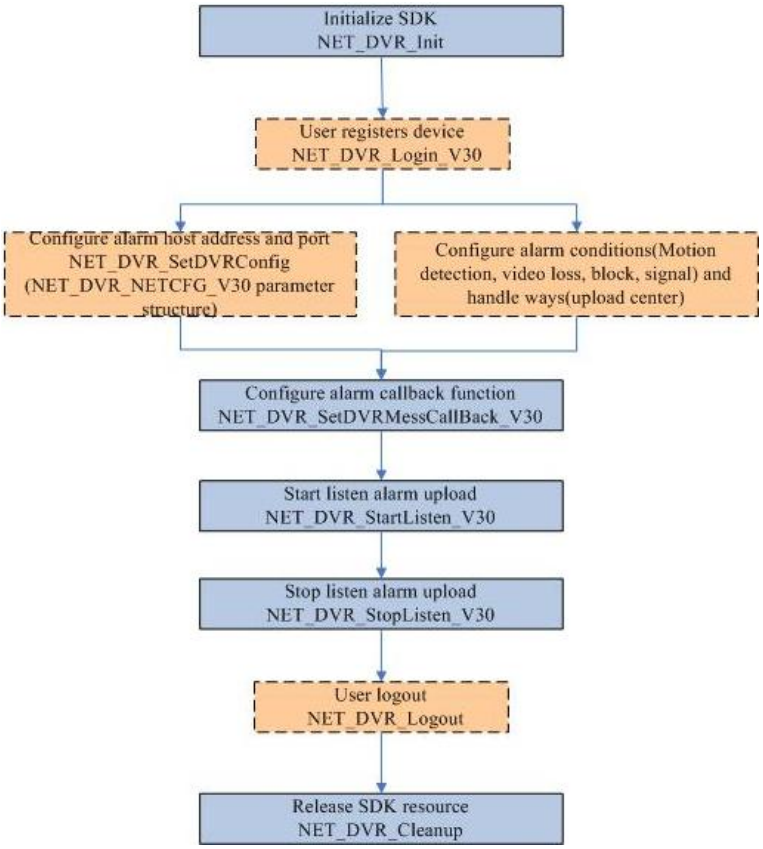
3.7.1 Alarm (arming) procedure



- "Arm" alarm mode: SDK connects to device actively, and send alarm uploading command to the device. And device will send alarm message to SDK immediatly when there's an alarm.
- Refer to the above "procedure chart","arm" needs to register (NET_DVR_Login_V30)at first. Dotted part is the necessary condition if you want the device uploading the alarm information, and this part mainly completes the configuration of relevant alarm conditions and handling ways, the parameter configuration interface is NET_DVR_GetDVRConfig and NET_DVR_SetDVRConfig. The supported alarm types are motion detection, video signal loss, block and signal alarm, the configuration structure of first three alarm types corresponding alarm conditions and handle ways is NET_DVR_PICCFG_V30,and signal alarm configuration structure is NET_DVR_ALARMINCFG_V30. If these parameters are already configured, dotted part can be omitted. The following is setting alarm callback function (NET_DVR_SetDVRMessageCallBack_V30 and other functions), and also need to arm the device on the client end(NET_DVR_SetupAlarmChan_V30). It needs to call function NET_DVR_CloseAlarmChan_V30 to disarm interface if you want to finish the whole alarm uploading process.

Example Code

3.7.2 Alarm (listening) procedure

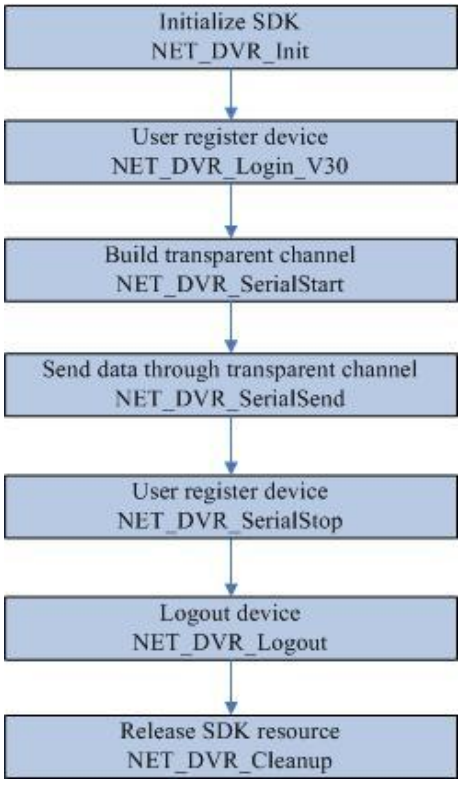


- "Listen" alarm mode: SDK dosen't connect to device actively, just listen to alarm message that uploaded actively by the device at the set listening port.
- This procedure needs to remotely configure device alarm host IP address(PC address) and alarm host port(PC listen port). Alarm host listens and receives the uploaded alarm message

at this port. If alarm host address and alarm host port have been configured, the dotted part of the above chart- "user register" and "cofigure alarm host address and port" parts, can be omitted. But if no configuration beforehand, must call parameter configuration interface (NET_DVR_GetDVRConfig and NET_DVR_SetDVRConfig) to configure network parameters by (NET_DVR_NETCFG_V30). And the dotted part"configure alarm conditions and handle ways" is the same with "arm". After setting all the parameters which need to be configured, please call NET_DVR_StartListen_V30 to open SDK listening port, ready for receiving device uploaded alarm information. This method is applicable, if some device upload alarm to a client, and the client doesn't need to login the device. Also, it doesn't affect alarm uploading if the device reboots. The drawback of this mode is that devices support to configure one alarm host address and one port number only.

Example Code

3.8 Transparent channel setup procedure



- SDK provides to use 485 and 232 as transparent channels. when using 232 serial port as transparent channel. At first you must set work mode of 232 configuration information to transparent channel mode.The specific way is calling interface NET_DVR_GetDVRConfig and NET_DVR_SetDVRConfig to get and set parameter dwWorkMode of NET_DVR_RS232CFG_V30 to transparent channel. When using 485 serial port as transparent channel. This step can be omitted. Call NET_DVR_SerialStart to build transparent channel and NET_DVR_SerialSend to send data . Need to do operations like break transparent channel (NET_DVR_SerialStop)after the whole process finished.

Example Code

4 API Calling Example

4.1 Example code of live view

Related procedure chart

Mode 1 SDK decodes real-time stream and display directly

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
#include <time.h>
using namespace std;

void CALLBACK g_ExceptionCallBack(DWORD dwType, LONG lUserID, LONG lHandle, void *pUser)
{
	char tempbuf[256] = {0};
	switch(dwType)
	{
	case EXCEPTION_RECONNECT:	// reconnect when preview
		printf("----------reconnect--------%d\n", time(NULL));
	break;
		default:
	break;
	}
}

void main() {

   //---------------------------------------
   //Initialize SDK

   NET_DVR_Init();
   //Set connect time and reconnect time
   NET_DVR_SetConnectTime(2000, 1);
   NET_DVR_SetReconnect(10000, true);

   //---------------------------------------
   //Login the device
   LONG lUserID;
   NET_DVR_DEVICEINFO_V30 struDeviceInfo;
```

```
    lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
    if (lUserID < 0)
    {
            printf("Login error, %d\n", NET_DVR_GetLastError());
            NET_DVR_Cleanup();
            return;
    }
    //---------------------------------------
    //Set exception callback function
    NET_DVR_SetExceptionCallBack_V30(0, NULL,g_ExceptionCallBack, NULL);
    //---------------------------------------
    //Start preview and set to callback stream data
    LONG lRealPlayHandle;
    HWND hWnd = GetConsoleWindow();         //Get window handle
    NET_DVR_CLIENTINFO ClientInfo = {0};
    ClientInfo.hPlayWnd = hWnd;
    //If need to decode, please set it valid. If want to get stream data only, it can be set to NULL
    ClientInfo.lChannel         = 1;            // Preview channel number.
    ClientInfo.lLinkMode        = 0;            /* The high bit (31) 0 means the main stream, while 1 means the sub
stream. Bit 0~bit 30 are used for link mode: 0- TCP mode, 1- UDP mode, 2- Multi-play mode, 3- RTP mode, 4- RTP
over RTSP, 5- RTSP over HTTP */
    ClientInfo.sMultiCastIP = NULL;     // Multicast IP. Please set when require to preview in multicast mode.

    BOOL bPreviewBlock = false;
    //Whether blocked when requiring a stream connection, 0 means unblocked, 1 means blocked
    lRealPlayHandle = NET_DVR_RealPlay_V30(lUserID, &ClientInfo, NULL, NULL, 0);
    if (lRealPlayHandle < 0)
    {
          printf("NET_DVR_RealPlay_V30 error\n");
                NET_DVR_Logout(lUserID);
          NET_DVR_Cleanup();
          return;
    }
    //---------------------------------------
    // Close preview
    NET_DVR_StopRealPlay(lRealPlayHandle);
    // Logout
    NET_DVR_Logout(lUserID);
    // Release SDK resource
    NET_DVR_Cleanup();

    return;
}
```

Mode 2 Users theirselves deal with stream data which called back by g_RealDataCallBack_V30. Here takes software decoding as an example.

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
#include <time.h>
#include "plaympeg4.h"
using namespace std;

LONG m_iPort; //Global Player port NO.

void CALLBACK g_RealDataCallBack_V30(LONG lRealHandle, DWORD dwDataType, BYTE *pBuffer,DWORD
dwBufSize,void* dwUser)
{
    HWND hWnd=GetConsoleWindow();
        switch (dwDataType)
        {
        case NET_DVR_SYSHEAD: //System head
                if (!PlayM4_GetPort(&lPort))    //Get unused port
                {
                        break;
                }
                m_iPort = lPort;    /*The data called back at the first time is system header. Please
assign this port to global port, and it will be used to play in next callback */
                if (dwBufSize > 0)
                {
                        if (!PlayM4_SetStreamOpenMode(lPort, STREAME_REALTIME))
                                        //Set real-time stream playing mode
                        {
                                break;
                        }
                        if (!PlayM4_OpenStream(lPort, pBuffer, dwBufSize, 1024*1024))
                                        //Open stream
                        {
                                break;
                        }
                        if (!PlayM4_Play(lPort, hWnd))    //Start play
                        {
                                break;
                        }
                }
        case NET_DVR_STREAMDATA:     //Stream data
```

```
                        if (dwBufSize > 0 && lPort != -1)
                        {
                                if (!PlayM4_InputData(lPort, pBuffer, dwBufSize))
                                {
                                        break;
                                }
                        }
                }
}

void CALLBACK g_ExceptionCallBack(DWORD dwType, LONG lUserID, LONG lHandle, void *pUser)
{
    char tempbuf[256] = {0};
    switch(dwType)
    {
    case EXCEPTION_RECONNECT:     //reconnect when preview
    printf("----------reconnect--------%d\n", time(NULL));
    break;
            default:
    break;
    }
}

void main() {

   //---------------------------------------
   //Initialize SDK
     NET_DVR_Init();
   //Set connection time and reconnection time
   NET_DVR_SetConnectTime(2000, 1);
   NET_DVR_SetReconnect(10000, true);

   //---------------------------------------
   // Login
   LONG lUserID;
   NET_DVR_DEVICEINFO_V30 struDeviceInfo;
   lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
   if (lUserID < 0)
   {
           printf("Login error, %d\n", NET_DVR_GetLastError());
           NET_DVR_Cleanup();
           return;
   }
```

```
    //---------------------------------------
    //Set exception callback function
    NET_DVR_SetExceptionCallBack_V30(0, NULL,g_ExceptionCallBack, NULL);

    //---------------------------------------
    //Start preview and set to callback stream data
    LONG lRealPlayHandle;
    NET_DVR_CLIENTINFO ClientInfo = {0};
    ClientInfo.hPlayWnd = NULL;
    //If need to decode, please set it valid. If want to get stream data only, we can set to NULL
    ClientInfo.lChannel       = 1;        //Preview channel number.
    ClientInfo.lLinkMode      = 0;        /*If 31st bit is 0, it means connect main stream, is 1 means sub stream.
Bit 0~bit 30 are used for link mode: 0- TCP mode, 1- UDP mode, 2- Multi-play mode, 3- RTP mode, 4- RTP over
RTSP, 5- RTP over HTTP */
    ClientInfo.sMultiCastIP = NULL;     //Multicast IP. Please set when require to preview in multicast mode.
    BOOL bPreviewBlock = false;
    //whether blocked when requiring a stream connection, 0 means unblocked, 1 means blocked
    lRealPlayHandle = NET_DVR_RealPlay_V30(lUserID, &ClientInfo, g_RealDataCallBack_V30, NULL, 0);
    if (lRealPlayHandle < 0)
    {
        printf("NET_DVR_RealPlay_V30 error\n");
            NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }
    //---------------------------------------
    //Close preview
    NET_DVR_StopRealPlay(lRealPlayHandle);
    //Logout
    NET_DVR_Logout_V30(lUserID);
    NET_DVR_Cleanup();
    return;
}
```

4.2 Example code of playback and download

Related procedure chart

Example no.1 Search the recording files and download the files

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
```

```
#include "HCNetSDK.h"
using namespace std;

int saveRecordFile(int userId,char * srcfile,char * destfile)
{
	int bRes = 1;
	int hPlayback = 0;
	if( (hPlayback = NET_DVR_GetFileByName(userId, srcfile, destfile)) < 0 )
	{
		printf( "GetFileByName failed. error[%d]\n", NET_DVR_GetLastError());
		bRes= -1;
		return bRes;
	}

	if(!NET_DVR_PlayBackControl(hPlayback, NET_DVR_PLAYSTART, 0, NULL))
	{
		printf("play back control failed [%d]\n",NET_DVR_GetLastError());
		bRes=-1;
		return bRes;
	}

	int nPos = 0;
	for(nPos = 0;    nPos < 100&&nPos>=0; nPos = NET_DVR_GetDownloadPos(hPlayback))
	{
		Sleep(5000);    //millisecond
	}
	printf("have got %d\n", nPos);

	if(!NET_DVR_StopGetFile(hPlayback))
	{
		printf("failed to stop get file [%d]\n",NET_DVR_GetLastError());
		bRes = -1;
		return bRes;
	}
	printf("%s\n",srcfile);

	if(nPos<0||nPos>100)
	{
		printf("download err [%d]\n",NET_DVR_GetLastError());
		bRes=-1;
		return bRes;
	}
	else
	{
```

```
            return 0;
        }
}

void main() {
    //---------------------------------------
    //Initialize SDK
    NET_DVR_Init();
    //Set connect time and reconnect time
    NET_DVR_SetConnectTime(2000, 1);
    NET_DVR_SetReconnect(10000, true);

    //---------------------------------------
    // Login the device
    LONG lUserID;
    NET_DVR_DEVICEINFO_V30 struDeviceInfo;
    lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
    if (lUserID < 0)
    {
            printf("Login error, %d\n", NET_DVR_GetLastError());
            NET_DVR_Cleanup();
            return;
    }

    NET_DVR_FILECOND struFileCond;
    struFileCond.dwFileType = 0xFF;
    struFileCond.lChannel = 1;
    struFileCond.dwIsLocked = 0xFF;
    struFileCond.dwUseCardNo = 0;
    struFileCond.struStartTime.dwYear      = 2011;
    struFileCond.struStartTime.dwMonth     = 3;
    struFileCond.struStartTime.dwDay       = 1;
    struFileCond.struStartTime.dwHour      = 10;
    struFileCond.struStartTime.dwMinute = 6;
    struFileCond.struStartTime.dwSecond =50;
    struFileCond.struStopTime.dwYear       = 2011;
    struFileCond.struStopTime.dwMonth      = 3;
    struFileCond.struStopTime.dwDay        = 1;
    struFileCond.struStopTime.dwHour       = 11;
    struFileCond.struStopTime.dwMinute     = 7;
    struFileCond.struStopTime.dwSecond     = 0;
    //---------------------------------------
    //Search recording files
    int lFindHandle = NET_DVR_FindFile_V30(lUserID, &struFileCond);
```

```
    if(lFindHandle < 0)
    {
        printf("find file fail,last error %d\n",NET_DVR_GetLastError());
        return;
    }
    NET_DVR_FINDDATA_V30 struFileData;
    while(true)
    {
        int result = NET_DVR_FindNextFile_V30(lFindHandle, &struFileData);
        if(result == NET_DVR_ISFINDING)
        {
            continue;
        }
        else if(result == NET_DVR_FILE_SUCCESS)
        {
            char strFileName[256] = {0};
            sprintf(strFileName, "./%s", struFileData.sFileName);
            saveRecordFile(lUserID, struFileData.sFileName,    strFileName);
            break;
        }
            else if(result == NET_DVR_FILE_NOFIND || result == NET_DVR_NOMOREFILE)
        {
            break;
        }
            else
            {
                printf("find file fail for illegal get file state");
            break;
            }
    }
    //Stop searching
    if(lFindHandle > 0)
    {
        NET_DVR_FindClose_V30(lFindHandle);
    }

    // Logout
    NET_DVR_Logout(lUserID);
    // Release SDK resource
    NET_DVR_Cleanup();
    return;
}
```

Eaxmple no.2 Playback the file by time

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
using namespace std;

void main() {

   //---------------------------------------
   //Initialize SDK
   NET_DVR_Init();
   //Set connect time and reconnect time
   NET_DVR_SetConnectTime(2000, 1);
   NET_DVR_SetReconnect(10000, true);

   //---------------------------------------
   // Login device
   LONG lUserID;
   NET_DVR_DEVICEINFO_V30 struDeviceInfo;
   lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
   if (lUserID < 0)
   {
          printf("Login error, %d\n", NET_DVR_GetLastError());
          NET_DVR_Cleanup();
          return;
   }

   NET_DVR_TIME struStartTime, struStopTime;
   struStartTime.dwYear      = 2011;
   struStartTime.dwMonth     = 3;
   struStartTime.dwDay       = 1;
   struStartTime.dwHour      = 9;
   struStartTime.dwMinute = 0;
   struStartTime.dwSecond =0;
   struStopTime.dwYear       = 2011;
   struStopTime.dwMonth      = 3;
   struStopTime.dwDay        = 1;
   struStopTime.dwHour       = 10;
   struStopTime.dwMinute     = 7;
   struStopTime.dwSecond     = 0;
   HWND hWnd = GetConsoleWindow();          //Get window handle

   //---------------------------------------
   //Playback by time
```

```
    int hPlayback;
    hPlayback = NET_DVR_PlayBackByTime(lUserID, 1, &struStartTime, &struStopTime, hWnd);
    if(hPlayback < 0)
    {
        printf("NET_DVR_GetFileByTime fail,last error %d\n",NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }

    //---------------------------------------
    //Start playing
    if(!NET_DVR_PlayBackControl(hPlayback, NET_DVR_PLAYSTART, 0, NULL))
    {
        printf("play back control failed [%d]\n",NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }

    Sleep(15000);    //millisecond
    if(!NET_DVR_StopPlayBack(hPlayback))
    {
        printf("failed to stop file [%d]\n",NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }

    // Logout
    NET_DVR_Logout(lUserID);
    // Release SDK resource
    NET_DVR_Cleanup();
    return;
}
```

Example 3 Download recording files by time

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
using namespace std;
```

```
void main() {

    //---------------------------------------
    //Initialize SDK
    NET_DVR_Init();
    //Set connect time and reconnect time
    NET_DVR_SetConnectTime(2000, 1);
    NET_DVR_SetReconnect(10000, true);

    //---------------------------------------
    // Login device
    LONG lUserID;
    NET_DVR_DEVICEINFO_V30 struDeviceInfo;
    lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
    if (lUserID < 0)
    {
            printf("Login error, %d\n", NET_DVR_GetLastError());
            NET_DVR_Cleanup();
            return;
    }

    NET_DVR_TIME struStartTime, struStopTime;
    struStartTime.dwYear      = 2011;
    struStartTime.dwMonth     = 3;
    struStartTime.dwDay       = 1;
    struStartTime.dwHour      = 9;
    struStartTime.dwMinute = 0;
    struStartTime.dwSecond =0;
    struStopTime.dwYear       = 2011;
    struStopTime.dwMonth      = 3;
    struStopTime.dwDay        = 1;
    struStopTime.dwHour       = 10;
    struStopTime.dwMinute     = 7;
    struStopTime.dwSecond     = 0;

    //---------------------------------------
    //Download by time
    int hPlayback;
    hPlayback = NET_DVR_GetFileByTime(lUserID, 1, &struStartTime, &struStopTime, "./test.mp4");
    if(hPlayback < 0)
    {
          printf("NET_DVR_GetFileByTime fail,last error %d\n",NET_DVR_GetLastError());
          NET_DVR_Logout(lUserID);
          NET_DVR_Cleanup();
```

```
        return;
    }

    //---------------------------------------
    //Start downloading
    if(!NET_DVR_PlayBackControl(hPlayback, NET_DVR_PLAYSTART, 0, NULL))
    {
        printf("play back control failed [%d]\n",NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }

    int nPos = 0;
    for(nPos = 0; nPos < 100&&nPos>=0; nPos = NET_DVR_GetDownloadPos(hPlayback))
    {
        Sleep(5000);    //millisecond
    }
    if(!NET_DVR_StopGetFile(hPlayback))
    {
        printf("failed to stop get file [%d]\n",NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }
    if(nPos<0||nPos>100)
    {
        printf("download err [%d]\n",NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }

    //Logout
    NET_DVR_Logout(lUserID);
    // Release SDK resource
    NET_DVR_Cleanup();
    return;
}
```

4.3 Example code of parameter configuration

Related procedure chart

Configure the compression parameter (NET_DVR_COMPRESSIONCFG_V30)

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
using namespace std;

void main() {

   //---------------------------------------
   //Initialize SDK
   NET_DVR_Init();
   //Set connect time and reconnect time
   NET_DVR_SetConnectTime(2000, 1);
   NET_DVR_SetReconnect(10000, true);

   //---------------------------------------
   // Login device
   LONG lUserID;
   NET_DVR_DEVICEINFO_V30 struDeviceInfo;
   lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
   if (lUserID < 0)
   {
          printf("Login error, %d\n", NET_DVR_GetLastError());
          NET_DVR_Cleanup();
          return;
   }

   int iRet;
   //Get compression parameter
   DWORD dwReturnLen;
   NET_DVR_COMPRESSIONCFG_V30 struParams = {0};
   iRet = NET_DVR_GetDVRConfig(lUserID, NET_DVR_GET_COMPRESSCFG_V30, struDeviceInfo.byStartChan, \
                &struParams, sizeof(NET_DVR_COMPRESSIONCFG_V30), &dwReturnLen);
   if (!iRet)
   {
         printf("NET_DVR_GetDVRConfig NET_DVR_GET_COMPRESSCFG_V30 error.\n");
         NET_DVR_Logout_V30(lUserID);
         NET_DVR_Cleanup();
```

```
        return;
    }

    // Set compression parameter
    struParams.struNormHighRecordPara.dwVideoBitrate = 22;
    iRet = NET_DVR_SetDVRConfig(lUserID, NET_DVR_SET_COMPRESSCFG_V30, struDeviceInfo.byStartChan, \
                &struParams, sizeof(NET_DVR_COMPRESSIONCFG_V30));
    if (!iRet)
    {
        printf("NET_DVR_GetDVRConfig NET_DVR_SET_COMPRESSCFG_V30 error.\n");
        NET_DVR_Logout_V30(lUserID);
        NET_DVR_Cleanup();
        return;
    }

    // Get compression parameter
    iRet = NET_DVR_GetDVRConfig(lUserID, NET_DVR_GET_COMPRESSCFG_V30, struDeviceInfo.byStartChan, \
                &struParams, sizeof(NET_DVR_COMPRESSIONCFG_V30), &dwReturnLen);
    if (!iRet)
    {
        printf("NET_DVR_GetDVRConfig NET_DVR_GET_COMPRESSCFG_V30 error.\n");
        NET_DVR_Logout_V30(lUserID);
        NET_DVR_Cleanup();
        return;
    }

    printf("Video Bitrate is %d\n", struParams.struNormHighRecordPara.dwVideoBitrate);
    //Logout
    NET_DVR_Logout(lUserID);
    // Release SDK resource
    NET_DVR_Cleanup();
    return;
}
```

4.4 Example code of remote device maintainence

Related procedure chart

Log query

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
```

```
using namespace std;

void main() {

    //---------------------------------------
    //Initialize SDK
    NET_DVR_Init();
    //Set connect time and reconnect time
    NET_DVR_SetConnectTime(2000, 1);
    NET_DVR_SetReconnect(10000, true);

    //---------------------------------------
    // Login device
    LONG lUserID;
    NET_DVR_DEVICEINFO_V30 struDeviceInfo;
    lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
    if (lUserID < 0)
    {
        printf("Login error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Cleanup();
        return;
    }

    NET_DVR_TIME struStartTime, struStopTime;
    struStartTime.dwYear       = 2011;
    struStartTime.dwMonth      = 3;
    struStartTime.dwDay        = 2;
    struStartTime.dwHour       = 9;
    struStartTime.dwMinute     = 0;
    struStartTime.dwSecond     =0;

    struStopTime.dwYear        = 2011;
    struStopTime.dwMonth       = 3;
    struStopTime.dwDay         = 2;
    struStopTime.dwHour        = 9;
    struStopTime.dwMinute      = 10;
    struStopTime.dwSecond      = 0;

    //---------------------------------------
    //Query log
    int lFindHandle = NET_DVR_FindDVRLog_V30(lUserID, 0, 0, 0, &struStartTime, &struStopTime, FALSE);
    if(lFindHandle < 0)
    {
        printf("find log fail,last error %d\n",NET_DVR_GetLastError());
```

```
            return;
        }
        NET_DVR_LOG_V30 struLog;
        while(true)
        {
            int result = NET_DVR_FindNextLog_V30(lFindHandle, &struLog);
            if(result == NET_DVR_ISFINDING)
            {
                printf("finding\n");
                continue;
            }
            else if(result == NET_DVR_FILE_SUCCESS)
            {
                char strLog[256] = {0};
                printf("log:%04d-%02d-%02d %02d:%02d:%02d\n", struLog.strLogTime.dwYear,
struLog.strLogTime.dwMonth, struLog.strLogTime.dwDay, \
                    struLog.strLogTime.dwHour,struLog.strLogTime.dwMinute, struLog.strLogTime.dwSecond);
            }
            else if(result == NET_DVR_FILE_NOFIND || result == NET_DVR_NOMOREFILE)
            {
                printf("find ending\n");
                break;
            }
            else
            {
                printf("find log fail for illegal get file state\n");
                break;
            }
        }

        //Stop log query
        if(lFindHandle > 0)
        {
            NET_DVR_FindLogClose_V30(lFindHandle);
        }

        //Logout
        NET_DVR_Logout(lUserID);
        // Release SDK resource
        NET_DVR_Cleanup();
        return;
}
```

4.5 Example code of voice talk and voice forward

Related procedure chart

Voice talk

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
using namespace std;

void CALLBACK fVoiceDataCallBack(LONG lVoiceComHandle, char *pRecvDataBuffer, DWORD dwBufSize, BYTE
byAudioFlag, void* pUser)
{
	printf("receive voice data, %d\n", dwBufSize);
}

void main() {

	//---------------------------------------
	//Initialize SDK
	NET_DVR_Init();
	//Set connect time and reconnect time
	NET_DVR_SetConnectTime(2000, 1);
	NET_DVR_SetReconnect(10000, true);

	//---------------------------------------
	// Login device
	LONG lUserID;
	NET_DVR_DEVICEINFO_V30 struDeviceInfo;
	lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
	if (lUserID < 0)
	{
		printf("Login error, %d\n", NET_DVR_GetLastError());
		NET_DVR_Cleanup();
		return;
	}

	//Voice talk
	LONG lVoiceHanle;
	lVoiceHanle = NET_DVR_StartVoiceCom_V30(lUserID, 1,0, fVoiceDataCallBack, NULL);
	if (lVoiceHanle < 0)
	{
```

```
            printf("NET_DVR_StartVoiceCom_V30 error, %d!\n", NET_DVR_GetLastError());
            NET_DVR_Logout(lUserID);
            NET_DVR_Cleanup();
            return;
        }

        Sleep(5000);    //millisecond
        //Stop voice talk
        if (!NET_DVR_StopVoiceCom(lVoiceHanle))
        {
            printf("NET_DVR_StopVoiceCom error, %d!\n", NET_DVR_GetLastError());
            NET_DVR_Logout(lUserID);
            NET_DVR_Cleanup();
            return;
        }

        //Logout
        NET_DVR_Logout(lUserID);
        // Release SDK resource
        NET_DVR_Cleanup();
        return;
}
```

4.6 Example code of alarm

Example of arming mode:

Related procedure chart

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
using namespace std;

void CALLBACK MessageCallback(LONG lCommand, NET_DVR_ALARMER *pAlarmer, char *pAlarmInfo, DWORD
dwBufLen, void* pUser)
{
    int i;
    NET_DVR_ALARMINFO struAlarmInfo;
    memcpy(&struAlarmInfo, pAlarmInfo, sizeof(NET_DVR_ALARMINFO));
    switch(lCommand)
    {
```

```
    case COMM_ALARM:
        {
            switch (struAlarmInfo.dwAlarmType)
            {
            case 3: //motion detection alarm
                for (i=0; i<16; i++)     //#define MAX_CHANNUM     16    //The max mumber of channels
                {
                    if (struAlarmInfo.dwChannel[i] == 1)
                    {
                        printf("Motion detection channel number: %d\n", i+1);
                    }
                }
            break;
            default:
            break;
            }
        }
    break;
    default:
    break;
    }
}

void main() {
    //---------------------------------------
    //Initialize SDK
    NET_DVR_Init();
    //Set connect time and reconnect time
    NET_DVR_SetConnectTime(2000, 1);
    NET_DVR_SetReconnect(10000, true);
    //---------------------------------------
    // Login device
    LONG lUserID;
    NET_DVR_DEVICEINFO_V30 struDeviceInfo;
    lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
    if (lUserID < 0)
    {
        printf("Login error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Cleanup();
        return;
    }

    //Set alarm callback function
    NET_DVR_SetDVRMessageCallBack_V30(MessageCallback, NULL);
```

```
    //Setup alarm channel (arming)
    LONG lHandle;
    lHandle = NET_DVR_SetupAlarmChan_V30(lUserID);
    if (lHandle < 0)
    {
        printf("NET_DVR_SetupAlarmChan_V30 error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }
    Sleep(5000);
    //Close alarm channel
    if (!NET_DVR_CloseAlarmChan_V30(lHandle))
    {
        printf("NET_DVR_CloseAlarmChan_V30 error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }
    //Logout
    NET_DVR_Logout(lUserID);
    //Release SDK resource
    NET_DVR_Cleanup();
    return;
}
```

Example of listening mode:

Related procedure chart

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
using namespace std;

void CALLBACK MessageCallback(LONG lCommand, NET_DVR_ALARMER *pAlarmer, char *pAlarmInfo, DWORD
dwBufLen, void* pUser)
{
    int i;
    NET_DVR_ALARMINFO struAlarmInfo;
    memcpy(&struAlarmInfo, pAlarmInfo, sizeof(NET_DVR_ALARMINFO));
    switch(lCommand)
    {
    case COMM_ALARM:
```

```
        {
            switch (struAlarmInfo.dwAlarmType)
            {
            case 3: // motion detection alarm
                for (i=0; i<16; i++)     //#define MAX_CHANNUM     16    // The max mumber of channels
                {
                    if (struAlarmInfo.dwChannel[i] == 1)
                    {
                        printf("Motion detection channel number: %d\n", i+1);
                    }
                }
            break;
            default:
            break;
            }
        }
    break;
    default:
    break;
    }
}

void main() {
    //---------------------------------------
    //Initialize SDK
    NET_DVR_Init();
    //Set connect time and reconnect time
    NET_DVR_SetConnectTime(2000, 1);
    NET_DVR_SetReconnect(10000, true);
    //---------------------------------------
    // Login device
    LONG lUserID;
    NET_DVR_DEVICEINFO_V30 struDeviceInfo;
    lUserID = NET_DVR_Login_V30("172.0.0.100", 8000, "admin", "12345", &struDeviceInfo);
    if (lUserID < 0)
    {
        printf("Login error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Cleanup();
        return;
    }

    //Set alarm callback function
    NET_DVR_SetDVRMessageCallBack_V30(MessageCallback, NULL);
```

```
    //Start listening
    LONG lHandle;
    lHandle = NET_DVR_StartListen_V30(NULL,7200, MessageCallback, NULL);
    if (lHandle < 0)
    {
        printf("NET_DVR_SetupAlarmChan_V30 error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }
    Sleep(5000);
    //Stop listening
    if (!NET_DVR_StopListen_V30(lHandle))
    {
        printf("NET_DVR_StopListen_V30 error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }
    //Logout
    NET_DVR_Logout(lUserID);
    // Release SDK resource
    NET_DVR_Cleanup();
    return;
}
```

4.7 Example code of transparent channel

Related procedure chart

```
#include <stdio.h>
#include <iostream>
#include "Windows.h"
#include "HCNetSDK.h"
using namespace std;

//External implement of callback transparent function
void CALLBACK g_fSerialDataCallBack(LONG lSerialHandle, char *pRecvDataBuffer, DWORD dwBufSize, DWORD
dwUser)
{
    //…… Deal with the transparent data, the data recieved are in pRecvDataBuffer.
}

void main() {
    //---------------------------------------
```

```
    //Init device
    NET_DVR_Init();
    //Set connect time and reconnect time
    NET_DVR_SetConnectTime(2000, 1);
    NET_DVR_SetReconnect(10000, true);
    //----------------------------------------
    //login device
    LONG lUserID;
    NET_DVR_DEVICEINFO_V30 struDeviceInfo;
    lUserID = NET_DVR_Login_V30("192.0.0.64", 8000, "admin", "12345", &struDeviceInfo);
    if (lUserID < 0)
    {
        printf("Login error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Cleanup();
        return;
    }
    /*Set 232 to transparent channel mode(485 is not necessary to call this interface used for 232 transparent channel)*/
    DWORD dwReturned = 0;
    NET_DVR_RS232CFG_V30 struRS232Cfg;
    memset(&struRS232Cfg, 0, sizeof(NET_DVR_RS232CFG_V30));
    if (!NET_DVR_GetDVRConfig(lUserID, NET_DVR_GET_RS232CFG_V30, 0, &struRS232Cfg, sizeof(NET_DVR_RS232CFG_V30), &dwReturned))
    {
        printf("NET_DVR_GET_RS232CFG_V30 error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }
    struRS232Cfg.struRs232.dwWorkMode = 2;
    //set 232 to transparent channel mode: 0- narrow-band transmission, 1- console, 2- transparent channel
    if (!NET_DVR_SetDVRConfig(lUserID, NET_DVR_SET_RS232CFG_V30, 0, &(struRS232Cfg), sizeof(NET_DVR_RS232CFG)))
    {
        printf("NET_DVR_SET_RS232CFG_V30 error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }
    //Set up transparent channel
    LONG lTranHandle;
    int iSelSerialIndex = 1; //1:RS-232;RS-485
    lTranHandle = NET_DVR_SerialStart(lUserID, iSelSerialIndex, g_fSerialDataCallBack, lUserID);
    //configure callback function to obtain transparent data
```

```
    if (lTranHandle < 0)
    {
        printf("NET_DVR_SerialStart error, %d\n", NET_DVR_GetLastError());
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }

    //Send data through transparent channel
    LONG lSerialChan = 0;//valid when using 485, begin with 1; set to 2 when using 232
    char szSendBuf[1016] = {0};
    if (!NET_DVR_SerialSend(lTranHandle, lSerialChan, szSendBuf, sizeof(szSendBuf)))
    //szSendBuf is send data buffer,iBufLen is buffer size
    {
        printf("NET_DVR_SerialSend error, %d\n", NET_DVR_GetLastError());
        NET_DVR_SerialStop(lTranHandle);
        NET_DVR_Logout(lUserID);
        NET_DVR_Cleanup();
        return;
    }
    //Stop transparent channel
    NET_DVR_SerialStop(lTranHandle);
    //Logout device
    NET_DVR_Logout(lUserID);
    //Release sdk resource
    NET_DVR_Cleanup();
    return;
}
```

5 API Description

5.1 SDK Initialization

5.1.1 Initialize SDK: NET_DVR_Init

API: BOOL NET_DVR_Init()
Parameters: None
Return: Return TRUE on success, FALSE on failure.
Remarks: This API is used to initialize SDK. Please call this API before calling any other API.

Return to index

5.1.2 Release SDK resource: NET_DVR_Cleanup

API: BOOL NET_DVR_Cleanup()
Parameters: None
Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.
Remarks: This API is used to release SDK resource. Please calling it before closing the program.

Return to index

5.1.3 Set network connection timeout and connection attempt times: NET_DVR_SetConnectTime

API: BOOL NET_DVR_SetConnectTime(DWORD dwWaitTime,DWORD dwTryTime)
Parameters: [in] dwWaitTime Timeout,unit: ms, value range: [300,75000], the actual max timeout time is different with different system connecting timeout
[in] dwTryTimes Connecting attempt times (reserved)
Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.
Remarks: Default timeout of SDK to establish a connection is 3 seconds. Interface will not return FASLE when the set timeout value is greater or less than the limit, it will take the nearest upper and lower limit value as the actual timeout.

Return to index

5.1.4 Set reconnecting time interval: NET_DVR_SetReconnect

API: BOOL NET_DVR_SetReconnect (DWORD dwInterval,BOOL bEnableRecon)

Parameters:

[in] dwInterval	Reconnecting interval, unit: milliseconds, default value:30 seconds
[in] bEnableRecon	Enable or disable reconnect function, 0-disable, 1-enable(default)

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API can set the reconnect function for preview, transparent channel and alar on guard state. If the user does not call this API, the SDK will initial the reconnect function for preview, transparent channel and alarm on guard state by default, and the reconnect interval is 5 seconds.

Return to index

5.1.5 Get the dynamic IP address of the device by IP server or EasyDDNS: NET_DVR_GetDVRIPByResolveSvr_EX

API: BOOL NET_DVR_GetDVRIPByResolveSvr_EX (char* sServerIP,WORD wServerPort,BYTE* sDVRName,WORD wDVRNameLen,BYTE* sDVRSerialNumber,WORD wDVRSerialLen,char* sGetIP,DWORD* dwPort)

Parameters:

[in] sServerIP	IP address of the IP server or EasyDDNS sever
[in] wServerPort	The server port of the IP server. Default port of IP server is 7071
[in] sDVRName	The name of the device
[in] wDVRNameLen	The length of the device's name
[in] sDVRSerialNumber	The serial number of the device
[in] wDVRSerialLen	The length of the serial number of the device
[out] sGetIP	Pointer to save the returned IP
[out] dwPort	Pointer to save the returned device port

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: The name and the serial no. of the DVR cannot be NULL at the same time. IPServer and EasyDDNS is one private dynamic DNS server.

Return to index

5.2 Exception Message Callback

5.2.1 Register window handle or callback function to receive exception, reconnection or other message:

NET_DVR_SetExceptionCallBack_V30

API: API in Windows system:
BOOL NET_DVR_SetExceptionCallBack_V30 (UINT nMessage,HWND hWnd,fExceptionCallBack cbExceptionCallBack,void* pUser)
API in Linux system:
BOOL NET_DVR_SetExceptionCallBack_V30(UINT nMessage,void* hWnd,fExceptionCallBack cbExceptionCallBack,void* pUser)

Parameters:

[in] nMessage	Message, this parameter is reserved in Linux
[in] hWnd	Window handle to receive exception message, this parameter is reserved in Linux SDK
[in] cbExceptionCallBack	Callback function to receive exception message and callback current exception relevant message
[in] pUser	User data

typedef void(CALLBACK* fExceptionCallBack)(DWORD dwType, LONG lUserID, LONG lHandle, void *pUser)

[out] dwType	Message types of exception or reconnection, see the below **macro definition table of exception message**
[out] lUserID	Login ID
[out] lHandle	Handle of relevant exception type
[out] pUser	User data

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: hWnd and cbExceptionCallBack can't be NULL at the same time in Windows system, and cbExceptionCallBack can't be set to NULL in Linux system, or it will not receive exception message.

Macro definition table of exception message:

Macro Definition	Value	Implication
EXCEPTION_EXCHANGE	0x8000	User interaction exception(timeput when heartbeat to register, the interval of heartbeat is 2 minutes)
EXCEPTION_AUDIOEXCHANGE	0x8001	Exception during voice talk
EXCEPTION_ALARM	0x8002	Exception during alarm uploading
EXCEPTION_PREVIEW	0x8003	Exception during live view

EXCEPTION_SERIAL	0x8004	Exception during transmitting data by transparent channel
EXCEPTION_RECONNECT	0x8005	Reconnect during live view
EXCEPTION_ALARMRECONNECT	0x8006	Reconnect during alarm
EXCEPTION_SERIALRECONNECT	0x8007	Reconnect during transparent channel
SERIAL_RECONNECTSUCCESS	0x8008	Transparent channel reconnected successfully
EXCEPTION_PLAYBACK	0x8010	Exception during playback
EXCEPTION_DISKFMT	0x8011	Exception during formatting hard disk
EXCEPTION_PASSIVEDECODE	0x8012	Exception during passive decoding
EXCEPTION_EMAILTEST	0x8013	Exception during e-mail test
EXCEPTION_BACKUP	0x8014	Exception during backup
PREVIEW_RECONNECTSUCCESS	0x8015	Live view reconnected successfully
ALARM_RECONNECTSUCCESS	0x8016	Alarm uploading reconnected successfully
RESUME_EXCHANGE	0x8017	User interaction resume to normal

If this structure feedbacks exception message by callback method, the exception callback function implement in the application is as follows, the parameter dwType of this function indicates exception message type(see the above table), lHandle indicates handle of the current exception relevant types.

Example:

```
//Register callback function for receiving exception message
NET_DVR_SetExceptionCallBack_V30(WM_NULL, NULL, g_ExceptionCallBack, NULL);

//External implement of callback function for receiving exception message
    void CALLBACK g_ExceptionCallBack(DWORD dwType, LONG lUserID, LONG lHandle, void *pUser)
{
        char tempbuf[256];
        ZeroMemory(tempbuf,256);
        switch(dwType)
        {
        case EXCEPTION_AUDIOEXCHANGE:                    //Network exception during voice talk
          sprintf(tempbuf,"Network exception during voice talk!!!");
                TRACE("%s",tempbuf);
                //TODO: Close voice talk
                break;
        case EXCEPTION_ALARM:                            //Network exception during uploading alarm
                sprintf(tempbuf," Network exception during uploading alarm!!!");
                TRACE("%s",tempbuf);
                //TODO: Close alarm uploading
                break;
```

```
        case EXCEPTION_PREVIEW:                    //Network exception during live view
                sprintf(tempbuf," Network exception during live view!!!");
                TRACE("%s",tempbuf);
                //TODO: Close live view
                break;
        case EXCEPTION_SERIAL:      //Exception during transmitting data by transparent channel
                sprintf(tempbuf," Exception during transmitting data by transparent channel!!!");
                TRACE("%s",tempbuf);
                //TODO: Close transparent channel
                break;
        case EXCEPTION_RECONNECT:                  //Reconnect during live view
                break;
        default:
                break;
        }
};
```

Return to index

5.3 SDK Information and Log

5.3.1 Get SDK version: NET_DVR_GetSDKVersion

API: DWORD NET_DVR_GetSDKVersion()

Parameters:

Return: SDK version information. Please call NET_DVR_GetLastError to get the error code.

Remarks: 2 higher bytes mean the major version, 2 lower bytes mean the minor version, e.g. 0x00030000 means version 3.0.

Return to index

5.3.2 Get SDK version and build information:

NET_DVR_GetSDKBuildVersion

API: DWORD NET_DVR_GetSDKBuildVersion()

Parameters:

Return: SDK version and build information. Please call NET_DVR_GetLastError to get the error code.

Remarks: The API is used to get the SDK version and build number. 2 higher bytes mean the major version: the bits from 25 to 32 mean major vesion number, and bits form 17 to 24 mean minor version number. 2 lower bytes mean build number, e.g. 0x03000101: the version is 3.0, build number is 0101.

Return to index

5.3.3 Get SDK current state: NET_DVR_GetSDKState

API: BOOL NET_DVR_GetSDKState(LPNET_DVR_SDKSTATE pSDKState);

Parameters: [out] pSDKState State information

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API is used to get SDK state.

Return to index

5.3.4 Get SDK ability: NET_DVR_GetSDKAbility

API: BOOL NET_DVR_GetSDKAbility(LPNET_DVR_SDKABL pSDKAbl)

Parameters: [out] pSDKAbl Ability information

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API is used to get ability of current SDK.

Return to index

5.3.5 Start writing log to file: NET_DVR_SetLogToFile

API: BOOL NET_DVR_SetLogToFile(DWORD bLogEnable,char* strLogDir,BOOL bAutoDel)

Parameters:

[in] bLogEnable Log level:
0- close log(default),
1- output ERROR log only,
2- output ERROR and DEBUG log,
3- output all log, including ERROR, DEBUG and INFO log

[in] strLogDir Log file saving path, if set to NULL, the default path for Windows is "C:\\SdkLog\\", and the default path for Linux is "/home/sdklog/"

[in] bAutoDel Whether to delete the files which exceed the number limit. Default: TRUE

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: The log file path must be absolute path, and should be finished with "\\ ", e.g."C:\\SdkLog\\". It is suggested to manually create file firstly. If no assigned file path, it will use the default path: "C:\\SdkLog\\". It supports to call the API multi times to create new log files and supports max 10 files at the

same time. If set bAutoDel to TRUE, it will automatically delete the files which exceed the limit. If the path is changed, it will use the new path when writing next file.

Return to index

5.4 Get Error Message

5.4.1 Return the Error Code of last operation: NET_DVR_GetLastError

API: DWORD NET_DVR_GetLastError()
Parameters:
Return: The error code of last operation.
Remarks: Return the error code. Generally, there are 3 different types of error information: error of network communication library, error of RTSP library, and error of software/hardware decoding library, see detail to **macro definition of error code**.

Return to index

5.4.2 Return the error message of last operation:

NET_DVR_GetErrorMsg

API: char* NET_DVR_GetErrorMsg(LONG *pErrorNo)
Parameters: [out] pErrorNo The pointer of the error code number
Return: The pointer that saves the error message. Please call NET_DVR_GetLastError to get the error code.
Remarks: Generally, there are 3 different types of error information: error of network communication library, error of RTSP library, and error of software/hardware decoding library, see detail to **macro definition of error code**.

Return to index

5.5 Login the Device

5.5.1 Loin the device: NET_DVR_Login_V30

API: LONG NET_DVR_Login_V30(char *sDVRIP,WORD wDVRPort, char *sUserName, char *sPassword,LPNET_DVR_DEVICEINFO_V30 lpDeviceInfo)
Parameters: [in] Sdvrip IP address of the device
[in] wDVRPort Port number of the devic
[in] sUserName User name
[in] sPassword Password

[out] lpDeviceInfo Device information

Return: Return -1 if it is failed, and other value is the value of returned user ID. The user ID is unique, and next operations should be realized through this ID. Please call NET_DVR_GetLastError to get the error code.

Remarks: IPC supports 16 different user names and 128 users login at the same time. SDK supports 512 * login. UserID is incremented one by one, from 0 to 511 and then return to 0. Logout and NET_DVR_Cleanup will not initialize the UserID to 0.

Return to index

5.5.2 Logout: NET_DVR_Logout

API: BOOL NET_DVR_Logout(LONG lUserID)

Parameters: [in] lUserID User ID, the return value of NET_DVR_Login_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It is suggested to call this API to logout.

Return to index

5.6 Get the capability set of the device

5.6.1 Get the capability set: NET_DVR_GetDeviceAbility

API: BOOL NET_DVR_GetDeviceAbility(LONG lUserID, DWORD dwAbilityType, char* pInBuf, DWORD dwInLength, char* pOutBuf, DWORD dwOutLength)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

[in] dwAbilityType Capability type, details listed below

[in] pInBuf Pointer of the input buffer (according to description mode of ability parameter, defined by device, it supports XML text or structure format)

[in] dwInLength Length of input buffer

[out] pOutBuf Pointer of the output buffer (according to description mode of ability set, defined by device, it supports XML text or structure format)

[in] dwOutLength Length of output buffer

Macro Definition	Value	Implication
DEVICE_SOFTHARDWARE_ABILITY	0x001	Software/hardware capability
DEVICE_NETWORK_ABILITY	0x002	Network capability
DEVICE_ENCODE_ALL_ABILITY	0x003	All encoding capability
DEVICE_ENCONE_CURRENT	0x004	Current encoding capability

IPC_FRONT_PARAMETER	0x005	Front-end parameter capability
DEVICE_ALARM_ABILITY	0x00a	Capability set of alarm

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: The definitions of pInBuf are different according to different devices, described by structure or XML text format. Similarly, pOutBuf can be described in structure or XML format according to different devices, too. The first 6 types of abilities are described in XML files. The details are available in each device's ability definition. The input and output parameter format when getting other types of abilities are defined as below:

Macro Definition	Type of Ability	pInBuf	pOutBuf
DEVICE_SOFTHARDWARE_ABILITY	Get software and hardware ability of current device	None	Device software and hardware ability described by XML
DEVICE_NETWORK_ABILITY	Get network ability of current device	None	Device network ability described by XML
DEVICE_ENCODE_ALL_ABILITY	Get all encoding ability of current device	None	Device all encoding ability described by XML
DEVICE_ENCONE_CURRENT	Get current encoding ability of current device	Device current encoding ability described by XML	Device current encoding ability described by XML
IPC_FRONT_PARAMETER	Get front-end parameter of current device	None	Device front-end camera parameter described by XML
DEVICE_ALARM_ABILITY	Get capability of alarm	Alarm capability described by XML	Alarm capability described by XML

Return to index

5.7 Live View

5.7.1 Set display mpde: NET_DVR_SetShowMode

API: BOOL NET_DVR_ SetShowMode (DWORD dwShowType, COLORREF colorKey)

Parameters: [in] dwShowType Display mode

```
enum{
        NORMALMODE = 0,
        OVERLAYMODE
}
```

[in] colorKey The transparent color set by user, which should

	be set when in OVERLAY mode. The transparent color just like a transparent film, the display picture only can go through this color, while other colors will prevent the display picture. User should put the color in the display window to show the display picture. Usually only one color are chosen as the transparent color. colorKey is the value of 32 bit 0x00bbggrr, the highest byte is 0, the last three byte is correspondingly refer to the value of b, g,r

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: There are two play modes: the common mode and OVERLAY mode. The advantage of OVERLAY mode is: most of the graphic adapter support OVERLAY, using OVERLAY mode in some graphic adapter which do not support BLT hardware shrink and enlarge and the switch of th color like SIS series graphic adapters, it will greatly reduce the CPU resources and improve the picture quality (which is correspondingly to using software to realize the shrink and enlarge, switch of color). And the disadvantage is it can only play one channel picture at a time, cannot realize large scale centralization surveillance.

There can only be one OVERLAY surface in the active state at one graphic adapter and at the sametime. If at that time there is a program using OVERLAY in the system, the player cannot establish an OVERLAY surface any more, it will change into the common mode automatically, while not return to FALSE. Some common player possibly use OVERLAY surface, thus the other program cannot use OVERLAY surface any more.

Return to index

5.7.2 Make the mian stream create a key frame(I frame):

NET_DVR_MakeKeyFrame

API: BOOL NET_DVR_MakeKeyFrame(LONG lUserID, DWORD lChannel)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] lChannel	Channel number

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: The interface is used to reset I frame, please call NET_DVR_MakeKeyFrame or NET_DVR_MakeKeyFrameSub to reset I frame for the main stream or sub stream according to the set preview parameter NET_DVR_CLIENTINFO.

Return to index

5.7.3 Make the sub stream create a key frame(I frame):

NET_DVR_MakeKeyFrameSub

API: BOOL NET_DVR_MakeKeyFrameSub(LONG lUserID, DWORD lChannel)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] lChannel	Channel number

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: The interface is used to reset I frame, please call NET_DVR_MakeKeyFrame or NET_DVR_MakeKeyFrameSub to reset I frame for the main stream or sub stream according to the set preview parameter NET_DVR_CLIENTINFO.

Return to index

5.7.4 Live view: NET_DVR_RealPlay_V30

API: LONG NET_DVR_RealPlay_V30(LONG lUserID, LPNET_DVR_CLIENTINFO lpClientInfo, fRealDataCallBack_V30 cbRealDataCallBack, void* pUser, BOOL bBlocked)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] lpClientInfo	Live view parameter
[in] fRealDataCallBack_V30	Real-time stream data callback function
[in] pUser	User data
[in] bBlocked	Whether to set data stream requesting process blocked or not: 0-no, 1-yes

typedef void(CALLBACK *fRealDataCallBack_V30)(LONG lRealHandle,DWORD dwDataType,BYTE *pBuffer,DWORD dwBufSize, void *pUser)

[out] lRealHandle	Curent live view handle
[out] dwDataType	Data type, details refer to **data type list table** below.
[out] pBuffer	Buffer pointer for saving data
[out] dwBufSize	Buffer size
[out] pUser	User data

Macro Definition	Value	Implication
NET_DVR_SYSHEAD	1	System head data
NET_DVR_STREAMDATA	2	Stream data (include video and audio stream, or only the video data of stream that video and audio is separate)
NET_DVR_AUDIOSTREAMDATA	3	Audio data

Return: -1 means failed, and other values could be used as handle of interface like NET_DVR_StopRealPlay. Please call NET_DVR_GetLastError to get the

error code.

Remarks: This API is used to realize live view. It supports to set current operation to be blocked or not(by the parameter: bBlocked). If set to be unblocked, it means it will think the connection is successful when start to connect with the device. If failed to receive stream and play, it will notify the upper layer by preview exception mode. And it can reduce dwell time of loop play, the same to NET_DVR_RealPlay. If set to be blocked, it means it will return whether successful or not after playing operation.
The callback function of this API can be set to NULL, and it will not callback the stream data to user. And then user can call NET_DVR_SetRealDataCallBack or NET_DVR_SetStandardDataCallBack to register callback function to capture stream data.

Return to index

5.7.5 Stop live view: NET_DVR_StopRealPlay

API: LONG NET_DVR_StopRealPlay (LONG lRealHandle)

Parameters: [in] lRealHandle Live view handle, the return value of NET_DVR_RealPlay_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API is used to stop live view.

Return to index

5.7.6 Get player handle for decoding and display when live view: NET_DVR_GetRealPlayerIndex

API: int NET_DVR_GetRealPlayerIndex(LONG lRealHandle)

Parameters: [in] lRealHandle Live view handle, the return value of NET_DVR_RealPlay_V30

Return: Return -1 if it is failed, and other returned values could be used as the play handle. Please call NET_DVR_GetLastError to get the error code.

Remarks: User can realize other functions supported by player SDK by returned handle.
For example:
When using PlayM4_GetBMP(LONG nPort,……),
PlayM4_GetJPEG(LONG nPort,……),
You can do like following:
PlayM4_GetBMP(NET_DVR_GetPlayBackPlayerIndex(),……)
PlayM4_GetJPEG(NET_DVR_GetPlayBackPlayerIndex(),……)
We can capture picture and save the data to memory.
Please refer <Player SDK Programmer Manual> for details.

Return to index

5.8 Video Parameter Configuration

5.8.1 Get video parameter: NET_DVR_ClientGetVideoEffect

API: BOOL NET_DVR_ClientGetVideoEffect(LONG lRealHandle,DWORD *pBrightValue, DWORD *pContrastValue,DWORD *pSaturationValue,DWORD *pHueValue)

Parameters:

[in] lRealHandle	The return value of NET_DVR_RealPlay_V30
[out] pBrightValue	Pointer of brightness, range: 1-10
[out] pContrastValue	Pointer of contrast, range: 1-10
[out] pSaturationValue	Pointer of saturation, range: 1-10
[out] pHueValue	Pointer of hue, range: 1-10

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Please call this API after starting live view.

Return to index

5.8.2 Get video parameter: NET_DVR_GetVideoEffect

API: BOOL NET_DVR_GetVideoEffect(LONG lUserID, LONG lChannel,DWORD *pBrightValue, DWORD *pContrastValue,DWORD *pSaturationValue,DWORD *pHueValue)

Parameters:

[in] lRealHandle	The return value of NET_DVR_Login_V30
[in] lChannel	Channel number
[out] pBrightValue	Pointer of brightness, range: 1-10
[out] pContrastValue	Pointer of contrast, range: 1-10
[out] pSaturationValue	Pointer of saturation, range: 1-10
[out] pHueValue	Pointer of hue, range: 1-10

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It supports get video parameter after login the device.

Return to index

5.8.3 Set video parameter: NET_DVR_ClientSetVideoEffect

API: BOOL NET_DVR_ClientSetVideoEffect(LONG lRealHandle,DWORD pBrightValue, DWORD pContrastValue,DWORD pSaturationValue,DWORD pHueValue)

Parameters:

[in] lRealHandle	The return value of NET_DVR_RealPlay_V30
[in] dwBrightValue	Brightness value, range: 1-10
[in] dwContrastValue	Contrast value, range: 1-10
[in] dwSaturationValue	Saturation value, range: 1-10

	[in] dwHueValue	Hue value, range: 1-10
Return:	Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.	
Remarks:	Please call this API after starting live view.	

Return to index

5.8.4 Set video parameter: NET_DVR_SetVideoEffect

API:	BOOL NET_DVR_SetVideoEffect(LONG lUserID, LONG lChannel,DWORD *pBrightValue, DWORD *pContrastValue,DWORD *pSaturationValue,DWORD *pHueValue)	
Parameters:	[in] lRealHandle	The return value of NET_DVR_Login_V30
	[in] lChannel	Channel number
	[in] dwBrightValue	Brightness value, range: 1-10
	[in] dwContrastValue	Contrast value, range: 1-10
	[in] dwSaturationValue	Saturation value, range: 1-10
	[in] dwHueValue	Hue value, range: 1-10
Return:	Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.	
Remarks:	It supports set video parameter after login the device.	

Return to index

5.9 Overlay Characters or Images onto Live View Screen

5.9.1 Overlay characters or images onto live view screen:

NET_DVR_RigisterDrawFun

API:	BOOL NET_DVR_RigisterDrawFun(LONG lRealHandle, fDrawFun cbDrawFun, DWORD dwUser)	
Parameters:	[in] lRealHandle	The return value of NET_DVR_RealPlay_V30
	[in] fDrawFun	Draw callback function
	[in] dwUser	User data
	typedef void(CALLBACK *fDrawFun)(LONG lRealHandle, HDC hDc, DWORD dwUser)	
	[out] lRealHandle	Current live view handle
	[out] hDc	Draw DC
	[out] dwUser	User data
Return:	Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.	
Remarks:	No such interface on Linux system. This API is mainly used to register callback function, and get device context of the current surface. User could draw or	

write on the DC, like drawing on the window client DC. But this DC is not DC of window client area, it is DC on the Off-Screen surface of Player DirectDraw.bBlocked should be set to 1(TRUE) when call NET_DVR_RealPlay_V30, or this API will return FALSE, and the error code will be 12 (calling order error) .

Return to index

5.10 Parameter Control of Decoding Effect When Live View

5.10.1 Set the number of player's frame buffers:

NET_DVR_SetPlayerBufNumber

API: BOOL NET_DVR_SetPlayerBufNumber(LONG lRealHandle,DWORD dwBufNum)

Parameters:

[in] lRealHandle	The return value of NET_DVR_RealPlay_V30
[in] dwBufNum	The max number of video frames set for single video playing, value range: [1,50],and the default number is 15

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Network delay and playing fluency can be adjusted through this interface. dwBufNum value is larger, the playing fluency is better and delay is larger; dwBufNum value is larger, the playing delay is smaller, but when network is not smooth, there will be frame loss phenomenon, affecting playing fluency. If current is mixed flow, in order to ensure effective proposal to set audio and video synchronization, frame buffer is advised to be greater than or equal to 6 frames. This function must be used immediately after NET_DVR_RealPlay, and the settings will not take effect if set after the video has been played.

Return to index

5.10.2 Set the number of B frames to be thrown when decoding:

NET_DVR_ThrowBFrame

API: BOOL NET_DVR_ThrowBFrame(LONG lRealHandle,DWORD dwNum)

Parameters:

[in] lRealHandle	The return value of NET_DVR_RealPlay_V30
[in] dwNum	The number of B frames to be thrown: 0- no throw, 1- throw 1 B frame, 2- throw 2 B frames

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Throw B frame can reduce CPU utilization when doing multi-channel playing. When play more than one channel, throw B frame can reduce the CPU resources, while if play one channel only, it'd better not to throw the B frame.

Return to index

5.11 Control Sound Playing When Live View

5.11.1 Set sound playing mode: NET_DVR_SetAudioMode

API: BOOL NET_DVR_SetAudioMode(DWORD dwMode)

Parameters: [in] dwMode Sound playing mod: 1- exclusive mode, single channel audio mode; 2- shared mode, multi-channel audio mode

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: If you don't call this interface to set sounding play mode, the default mode is exclusive.

Return to index

5.11.2 Open sound in exclusive mode: NET_DVR_OpenSound

API: BOOL NET_DVR_OpenSound(LONG lRealHandle)

Parameters: [in] lRealHandle The return value of NET_DVR_RealPlay_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: If currently it is in shared mode, this API will return false. It supports only opening one channel to play sound in the exclusive mode, that is, it only opens the sound of the last channel when more one channels are opened one by one.

Return to index

5.11.3 Close sound in exclusive mode: NET_DVR_CloseSound

API: BOOL NET_DVR_CloseSound()

Parameters: None

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API is used to close sound on exclusive sound card mode.

Return to index

5.11.4 Open sound in shared mode: NET_DVR_OpenSoundShare

API: BOOL NET_DVR_OpenSoundShare(LONG lRealHandle)
Parameters: [in] lRealHandle The return value of NET_DVR_RealPlay_V30
Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.
Remarks: This API is used to open sound in shared sound card mode.

Return to index

5.11.5 Close sound in shared mode: NET_DVR_CloseSoundShare

API: BOOL NET_DVR_CloseSoundShare (LONG lRealHandle)
Parameters: [in] lRealHandle The return value of NET_DVR_RealPlay_V30
Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.
Remarks: This API is used to close sound in share sound card mode.

Return to index

5.11.6 Adjust playing volume: NET_DVR_Volume

API: BOOL NET_DVR_Volume(LONG lRealHandle,WORD wVolume)
Parameters: [in] lRealHandle The return value of NET_DVR_RealPlay_V30
[in] wVolume Volume, value arrange:[0,0xffff]
Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.
Remarks: This API is used to adjust playing volume.

Return to index

5.12 Stream Data Callback When Live View

5.12.1 Register callback function to capture real-time stream date: NET_DVR_SetRealDataCallBack

API: BOOL NET_DVR_SetRealDataCallBack(LONG lRealHandle, fRealDataCallBack cbRealDataCallBack,DWORD dwUser)
Parameters: [in] lRealHandle Live view handle, the return value of NET_DVR_RealPlay_V30
[in] fRealDataCallBack Stream data callback function

[in] dwUser User data

typedef void(CALLBACK *fRealDataCallBack)(LONG lRealHandle,DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize,DWORD dwUser)

[out] lRealHandle Current live view handle

[out] dwDataType Data type, details refer to **data type list table**

[out] pBuffer Buffer pointer to save data

[out] dwBufSize Buffer size

[out] dwUser User data

Macro Definition	Value	Implication
NET_DVR_SYSHEAD	1	System head data
NET_DVR_STREAMDATA	2	Stream data (include video and audio stream, or only the video data of stream that video and audio is separate)

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This function includes starting and stopping the user to handle the data captured by SDK. When fRealDataCallBack is not NULL, it means SDK will callback the stream data and user can handle the data. When fRealDataCallBack is NULL, it means stop calling back the data and handling the data. The first package called back by the function is a system head of 40 bytes, and it is used to decode the stream data. The afterward data called back is the compressed data stream. The max size of the data called back one time is 256K bytes. ***The example, please refer to Example code of live view.***

Return to index

5.12.2 Register callback function to capture real-time stream date (standard encoded data): NET_DVR_SetStandardDataCallBack

API: BOOL NET_DVR_SetStandardDataCallBack(LONG lRealHandle, fStdDataCallBack cbStdDataCallBack,DWORD dwUser)

Parameters: [in] lRealHandle Live view handle, the return value of NET_DVR_RealPlay_V30

[in] fStdDataCallBack Standard data callback function

[in] dwUser User data

typedef void(CALLBACK *fStdDataCallBack)(LONG lRealHandle,DWORD dwDataType, BYTE *pBuffer,DWORD dwBufSize,DWORD dwUser)

[out] lRealHandle Current live view handle

[out] dwDataType Data type, details refer to **data type list table**

[out] pBuffer Buffer pointer to save data

[out] dwBufSize Buffer size

[out] dwUser User data

Macro Definition	Value	Implication
NET_DVR_SYSHEAD	1	System header
NET_DVR_STD_VIDEODATA	4	Standard video stream data
NET_DVR_STD_AUDIODATA	5	Standard audio steam data
NET_DVR_PRIVATE_DATA	2 or 112	Private data

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This function includes starting and stopping the user to handle the data captured by SDK. When fRealDataCallBack is not NULL, it means SDK will callback the stream data and user can handle the data. When fRealDataCallBack is NULL, it means stop calling back the data and handling the data. The first package called back by the function is a system head of 40 bytes, and it is used to decode the stream data. The afterward data called back is the compressed data stream(include RTP header of 12bytes).
This function currently supports to callback standard stream data from devices that support RTSP protocol only.

Return to index

5.12.3 Capture data and save to assigned file: NET_DVR_SaveRealData

API: BOOL NET_DVR_SaveRealData(LONG lRealHandle,char *sFileName)

Parameters: [in] lRealHandle The return value of NET_DVR_RealPlay_V30
[in] sFileName Pointer of file path

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.12.4 Stop data callback: NET_DVR_StopSaveRealData

API: BOOL NET_DVR_StopSaveRealData(LONG lRealHandle)

Parameters: [in] lRealHandle The return value of NET_DVR_RealPlay_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.13 Capture Picture

5.13.1 Set capturing mode: NET_DVR_SetCapturePictureMode

API: BOOL NET_DVR_SetCapturePictureMode(DWORD dwCaptureMode)

Parameters: [in] dwCaptureMode Capturing mode

```
enum tagPDC_PARAM_KEY{
    BMP_MODE    = 0,    // BMP mode
    JPEG_MODE   = 1     // JPEG mode
}CAPTURE_MODE
```

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: After calling this API to set capturing mode, please call NET_DVR_CapturePicture to get the corresponding picture.

Return to index

5.13.2 Capture a frame and save to file: NET_DVR_CapturePicture

API: BOOL NET_DVR_CapturePicture(LONG lRealHandle,char *sPicFileName)

Parameters: [in] lRealHandle The return value of NET_DVR_RealPlay_V30

[in] sPicFileName URL to save picture, path length is less than or equal to 256 bytes(includes file name)

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It supports to call NET_DVR_SetCapturePictureMode to set capture mode, before calling this API to get picture. The default mode is BMP mode. If set capturing mode to BMP mode, the captured file is a BMP file, and the suffix of file path should be ".bmp"; If set to JPEG mode, it captures a JPEG file, and the suffix of file path should be ".jpg".

If the current resolution of device is 2CIF, the resolution of captured bmp picture is 4CIF.

Return to index

5.13.3 Capture a file and save as JPEG picture:

NET_DVR_CaptureJPEGPicture

API: BOOL NET_DVR_CaptureJPEGPicture(LONG lUserID, LONG lChannel, LPNET_DVR_JPEGPARA lpJpegPara, char *sPicFileName)

Parameters:	[in] lUserID	The return value of NET_DVR_Login_V30
	[in] lChannel	Channel number
	[in] lpJpegPara	JPEG image parameter
	[in] sPicFileName	File path to save JPEG picture
Return:	Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.	
Remarks:	The API is used to capture a frame and save as JPEG file. For IPC, it supports to capture JPEG image of current resolution.	

Return to index

5.13.4 Capture a frame and save as JPEG image to the assigned buffer:

NET_DVR_CaptureJPEGPicture_NEW

API:	BOOL NET_DVR_CaptureJPEGPicture_NEW(LONG lUserID, LONG lChannel, LPNET_DVR_JPEGPARA lpJpegPara, char *sJpegPicBuffer, DWORD dwPicSize, LPDWORD lpSizeReturned)	
Parameters:	[in] lUserID	The return value of NET_DVR_Login_V30
	[in] lChannel	Channel number
	[in] lpJpegPara	JPEG image parameter
	[in] sJpegPicBuffer	The buffer to save JPEG data
	[in] dwPicSize	The buffer size
	[out] lpSizeReturned	The returned size of the picture
Return:	Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.	
Remarks:	The API is used to capture a frame and save as JPEG picture to the assigned buffer. For IPC, it supports to capture JPEG image of current resolution.	

Return to index

5.14 Operation with Remote Files Recorded in the Device: Playback, Download, Lock or Backup

Get the video's starting time and stopping time of the channel

Search record files

5.14.1 Search files by file type and time: NET_DVR_FindFile_V40

API: LONG NET_DVR_FindFile_V40(LONG lUserID, LPNET_DVR_FILECOND_V40 pFindCond)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

[in] pFindCond The structure of file information to be found

Return: Return -1 if it is failed, and other values could be used as a parameter of NET_DVR_FindClose and other APIs. Please call NET_DVR_GetLastError to get the error code.

Remarks: The interface has assigned the file type and time-range to search. After calling it successfully, please call NET_DVR_FindNextFile_V30 to get file information.

Return to index

5.14.2 Get record file one by one: NET_DVR_FindNextFile_V30

API: LONG NET_DVR_FindNextFile_V30(LONG lFindHandle, LPNET_DVR_FINDDATA_V30 lpFindData)

Parameters: [in] lFindHandle Handle of file searching, return value of NET_DVR_FindFile_V30

[in] lpFindData Pointer for saving file information

Return: Return -1 if it is failed, and the other values stand for current state or other information, details listed below:

Macro Definition	Value	Implication
NET_DVR_FILE_SUCCESS	1000	Get the file information successfully
NET_DVR_FILE_NOFIND	1001	No file found
NET_DVR_ISFINDING	1002	Searching, please wait
NET_DVR_NOMOREFILE	1003	No more file found, search is finished
NET_DVR_FILE_EXCEPTION	1004	Exception when search file

Please call NET_DVR_GetLastError to get the error code.

Remarks: Before calling this function, please call NET_DVR_FindFile_V30 to get current

handle firstly. The interface only supports to get one file. We should call the interface repetitively to get all files. We can get other information, like card number and whether the file is locked, by calling this API as well.

The max number of files searched once is 4000.

Return to index

5.14.3 Close searching files and release the resource :

NET_DVR_FindClose_V30

API:	BOOL NET_DVR_FindClose_V30(LONG lFindHandle)	
Parameters:	[in] lFindHandle	The handle of file search, the return value of NET_DVR_FindFile_V30
Return:	Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.	
Remarks:		

Return to index

Playback record files

5.14.4 Playback by file name: NET_DVR_PlayBackByName

API:	LONG NET_DVR_PlayBackByName(LONG lUserID,char *sPlayBackFileName, HWND hWnd)	
Parameters:	[in] lUserID	The return value of NET_DVR_Login_V30
	[in] sPlayBackFileName	File name to playback, the length can not exceed 100 bytes
	[in] hWnd	Handle of playback window. If set to NULL, SDK still can receive stream data, but not decode and display
Return:	Return -1 if it is failed, and other values could be used as parameter of NET_DVR_StopPlayBack. Please call NET_DVR_GetLastError to get the error code.	
Remarks:	This API assigns the record file to play currently. After calling the API successfully, it requires to call the NET_DVR_PlayBackControl with the command **NET_DVR_PLAYSTART** to start playback.	
	After calling the API successfully, you can register callback function by calling NET_DVR_SetPlayDataCallBack to capture the stream data and handle by yourself.	
	In Linux system	
	For v4.1 SDK or above version,HWND means the handle of playing window, defined as below:	

```
typedef unsigned int HWND;
```

If you use the Qt interface development, here take an example:

```
NET_DVR_CLIENTINFO tmpclientinfo;
tmpclientinfo.hPlayWnd = (HWND)m_framePlayWnd->GetPlayWndId();
```

For the SDK under v4.1, HWND is defined as below:

```
typedef struct __PLAYRECT
{
    int x;          //X axis coordinate of the display region's upper left corner
    int y;          //Y axis coordinate of the display region's upper left corner
    int uWidth;     //Width of the display region
    int uHeight;    //Height of the display region
}PLAYRECT;
typedef PLAYRECT HWND;
```

For the structure NET_DVR_CLIENTINFO, if hPlayWnd = {0}, SDK can still get stream but not decode and display, so it is able to record on the client end. It is not able to set hPlayWnd = 0(that is, NULL), or it will result to crumble when calling hPlayWnd.x.

In Linux system, HWND definition as follows:

```
typedef struct __PLAYRECT
{
    int x;      //X axis coordinate of the display region's upper left corner
    int y;      //Y axis coordinate of the display region's upper left corner
    int uWidth;     //Width of the display region
    int uHeight;    //Height of the display region
}PLAYRECT;
```

Return to index

5.14.5 Playback by time: NET_DVR_PlayBackByTime_V40

API: LONG NET_DVR_PlayBackByTime_V40(LONG lUserID, LPNET_DVR_VOD_PARA pVodPara)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] pVodPara	Playback parameter

Return: Return -1 if it is failed, and other values could be used as parameter of NET_DVR_StopPlayBack. Please call NET_DVR_GetLastError to get the error code.

Remarks: This interface assigns the record file to play currently. After calling the API successfully, it requries to call the NET_DVR_PlayBackControl_V40 with the command NET_DVR_PLAYSTART to start playback.

When the record files to playback are searched by event, for each file has pre-record and delay part, please extend the end time and ahead the starting time to playback. The recommended value: bup to 10 minutes, at least 5

seconds.

After calling the API successfully, you can register callback function by calling NET_DVR_SetPlayDataCallBack, capture the stream data and handle by yourself.

Return to index

5.14.6 Control the playback state: NET_DVR_PlayBackControl_V40

API: BOOL NET_DVR_PlayBackControl_V40(LONG lPlayHandle,DWORD dwControlCode, LPVOID lpInBuffer = NULL, DWORD dwInLen = 0, LPVOID lpOutBuffer = NULL, DWORD *lpOutLen = NULL)

Parameters:

[in] lPlayHandle	Playback handle, the return value of NET_DVR_PlayBackByName or NET_DVR_PlayBackByTime.
[in] dwControlCode	Command to control video playback, details see to **the list table** below.
[in] lpInBuffer	Pointer to input parameter.
[in] dwInLen	Length of input parameter.
[out] lpOutBuffer	Pointer to output parameter.
[out] lpOutLen	Length of output parameter.

Macro Definition	Value	Implication
NET_DVR_PLAYSTART	1	Start playing
NET_DVR_PLAYPAUSE	3	Pause
NET_DVR_PLAYRESTART	4	Resume
NET_DVR_PLAYFAST	5	Fast
NET_DVR_PLAYSLOW	6	Slow
NET_DVR_PLAYNORMAL	7	Normal speed
NET_DVR_PLAYFRAME	8	Play frame one by one (using the command NET_DVR_PLAYNORMAL to resume normal playback)
NET_DVR_PLAYSTARTAUDIO	9	Open sound
NET_DVR_PLAYSTOPAUDIO	10	Close sound
NET_DVR_PLAYAUDIOVOLUME	11	Adjust the volume
NET_DVR_PLAYSETPOS	12	Change the progress of the file playback
NET_DVR_PLAYGETPOS	13	Get the progress of the file playback
NET_DVR_PLAYGETTIME	14	Get currently played time(valid when playing back by file)
NET_DVR_PLAYGETFRAME	15	Get currently played frames(valid when playing back by file)

NET_DVR_GETTOTALFRAMES	16	Get currently total frames(valid when playing back by file)
NET_DVR_GETTOTALTIME	17	Get currently total time(valid when playing back by file)
NET_DVR_THROWBFRAME	20	Throw B frame
NET_DVR_SETSPEED	24	Set speed of stream
NET_DVR_KEEPALIVE	25	Keep heartbeat with device (If the callback blocked, suggest setting 2s to send one time)
NET_DVR_PLAYSETTIME	26	Positioning by absolute time
NET_DVR_PLAYGETTOTALLEN	27	Get total length of all files in corresponding time period of playback by time
NET_DVR_PLAY_FORWARD	29	Switch rewind to forward playback
NET_DVR_PLAY_REVERSE	30	Switch forward playback to rewind

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Whether the third parameter of this interface requires to input value is related with the control command, details refer to the following table (**The relationship of dwControlCode, lpInBuffer and lpOutBuffer**). Specially, when control command is starting to play(NET_DVR_PLAYSTART), the third parameter value means offset value of current played file, if this value is 0, it means play from the file's starting position; if this value isn't 0, it means offset value (Byte).

The fifth parameter of this API means corresponding parameter got by current control command. The control commands, NET_DVR_PLAYGETPOS, NET_DVR_PLAYGETTIME, NET_DVR_PLAYGETFRAME, NET_DVR_GETTOTALFRAMES, NET_DVR_GETTOTALTIME, NET_DVR_PLAYSETTIME and NET_DVR_PLAYGETTOTALLEN, can get the corresponding values by this parameter; details refer to the following table.

When command value is NET_DVR_PLAYGETPOS, to get file playback or download progress, 0-100 means normal progress value, value larger than 100 means playback or download is abnormal.

When getting the progress of playback or download by time, DS-91xxHF-ST/DS-90xxHF-ST/DS-96xxHF-ST/DS-81xxHF-ST supports to get the progress of 0~100 and 200(exception), and other devices can get the progress of 0, 100(finished), and 200(exception).

The relationship of dwControlCode, lpInBuffer and lpOutBuffer:

Command Macro Definition	Command Description	lpInBuf	lpOutBuf
NET_DVR_PLAYSTART	Start playing	A 4-byte integer offset	None
NET_DVR_PLAYSETPOS	Change playback progress	A 4-byte integer progress(0-100)	None

NET_DVR_PLAYGETPOS	Get playback progress	None	A 4-byte integer progress (0-100)
NET_DVR_PLAYGETTIME	Get currently played time (valid when playing back by file)	None	A 4-byte integer time value
NET_DVR_PLAYGETFRAME	Get currently played frames (valid when playing back by file)	None	A 4-byte integer frame number
NET_DVR_GETTOTALFRAMES	Get total frames current playing file (valid when playing back by file)	None	A 4-byte integer frame number
NET_DVR_GETTOTALTIME	Get total time of current playing file (valid when playing back by file)	None	A 4-byte integer time value
NET_DVR_THROWBFRAME	Throw B frame	4-byte integer, total number of B frames	None
NET_DVR_SETSPEED	Set speed of stream	A 4-byte integer speed value	None
NET_DVR_PLAYSETTIME	Locate playback by absolute time	NET_DVR_TIME	None
NET_DVR_PLAYGETTOTALLEN	Get total length of all files in corresponding time period of playback by time	None	A 8-byte integer length value
NET_DVR_PLAY_FORWARD	Switch rewind to forward playback	If decoded by user at the application layer, lpInBuffer should input NET_DVR_TIME and it means the current playing time; If decoded by the SDK directly, lpInBuffer could be set as NULL	None
NET_DVR_PLAY_REVERSE	Switch forward playback to rewind		None

Return to index

5.14.7 Stop playback: NET_DVR_StopPlayBack

API: BOOL NET_DVR_StopPlayBack(LONG lPlayHandle)

Parameters: [in] lPlayHandle Playback handle, the return value of NET_DVR_PlayBackByName or NET_DVR_PlayBackByTime.

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

Data callback when playback

5.14.8 Callback the playing data, and save as a file:

NET_DVR_PlayBackSaveData

API: BOOL NET_DVR_PlayBackSaveData(LONG lPlayHandle,char *sFileName)

Parameters: [in] lPlayHandle Playback handle, the return value of NET_DVR_PlayBackByName or NET_DVR_PlayBackByTime.

[in] sFileName Pointer of file path

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.14.9 Stop saving data: NET_DVR_StopPlayBackSave

API: BOOL NET_DVR_StopPlayBackSave(LONG lPlayHandle)

Parameters: [in] lPlayHandle Playback handle, the return value of NET_DVR_PlayBackByName or NET_DVR_PlayBackByTime.

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.14.10 Register callback function to get record data:

NET_DVR_SetPlayDataCallBack

API: BOOL NET_DVR_SetPlayDataCallBack(LONG lPlayHandle, fPlayDataCallBack cbPlayDataCallBack,DWORD dwUser)

Parameters: [in] lPlayHandle Playback handle, the return value of NET_DVR_PlayBackByName or NET_DVR_PlayBackByTime.

[in] fPlayDataCallBack Callback function of record data
[in] dwUser User data
typedef void(CALLBACK *fPlayDataCallBack)(LONG lPlayHandle,DWORD dwDataType,BYTE *pBuffer,DWORD dwBufSize,DWORD dwUser)
[out] lPlayHandle Current playback handle
[out] dwDataType Data type, see to **the list table** below
[out] pBuffer Buffer of saving the captured data
[out] dwBufSize Buffer size
[out] dwUser User data

Macro Definition	Value	Implication
NET_DVR_SYSHEAD	1	System head data
NET_DVR_STREAMDATA	2	Stream data(compound stream or only video stream)

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This function includes starting and stopping user to handle the data captured by SDK. When the callback function cbPlayDataCallback is set to not NULL value, it indicates to callback and process the data; when set to NULL, it indicates to stop callback and handle the data. The first callback package is a system head of 40 bytes, used for following decoding. Then, after the system head, the callback data is compressed stream data.

Return to index

Other operation about playback

5.14.11 Get the display OSD time when playback the record file:

NET_DVR_GetPlayBackOsdTime

API: BOOL NET_DVR_GetPlayBackOsdTime(LONG lPlayHandle, LPNET_DVR_TIME lpOsdTime)

Parameters: [in] lPlayHandle Playback handle, the return value of NET_DVR_PlayBackByName or NET_DVR_PlayBackByTime.
[out] lpOsdTime The OSD time

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.14.12 Capture picture when playback, and save as a file:

NET_DVR_PlayBackCaptureFile

API: BOOL NET_DVR_PlayBackCaptureFile(LONG lPlayHandle,char *sFileName)

Parameters:

[in] lPlayHandle	Playback handle, the return value of NET_DVR_PlayBackByName or NET_DVR_PlayBackByTime.
[in] sFileName	The file path to save picture

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Generally, the time of captured picture will delay after the time of starting capturing. That is because the OSD time on playing screen is the display time after decoding, while there should be about 1M data in decoding buffer that have not been decoded, and the picture data to be captured is got from the network buffer. Currently, the decoding library hasn't the interface to get data from the decoding buffer.

Return to index

5.14.13 Refresh to display the playback window:

NET_DVR_RefreshPlay

API: BOOL NET_DVR_RefreshPlay(LONG lPlayHandle)

Parameters:

[in] lPlayHandle	Playback handle, the return value of NET_DVR_PlayBackByName or NET_DVR_PlayBackByTime.

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: When pausing the playback or playing back frame one by one, if refresh the window, the image will disappear. Please call this interface to display the last frame again. This interface is valid only for pausing the playback or playing back frame one by one.

Return to index

5.14.14 Get player handle for decoding and display when playback:

NET_DVR_GetPlayBackPlayerIndex

API: int NET_DVR_GetPlayBackPlayerIndex(LONG lPlayHandle)

Parameters:	[in] lPlayHandle	Playback handle, the return value of NET_DVR_PlayBackByName or NET_DVR_PlayBackByTime.
Return:	Return -1 if it is failed, and other returned values could be used as the play handle. Please call NET_DVR_GetLastError to get the error code.	
Remarks:	User can realize other functions supported by player SDK by returned handle. For example: When using PlayM4_GetBMP(LONG nPort,……), PlayM4_GetJPEG(LONG nPort,……), You can do like following: PlayM4_GetBMP(NET_DVR_GetPlayBackPlayerIndex(),……) PlayM4_GetJPEG(NET_DVR_GetPlayBackPlayerIndex(),……) We can capture picture and save the data to memory. Please refer <Player SDK Programmer Manual> for details.	

Return to index

Download the record files from the remote device

5.14.15 Download by file name: NET_DVR_GetFileByName

API:	LONG NET_DVR_GetFileByName(LONG lUserID,char *sDVRFileName,char *sSavedFileName)	
Parameters:	[in] lUserID	The return value of NET_DVR_Login_V30
	[in] sDVRFileName	The file name to be downloaded, the size of file name should be less than 100 bytes
	[in] sSavedFileName	The files name saved in the computer after downloaded, it should be absolute path
Return:	Return -1 if it is failed, and other values could be used as the parameter of functions NET_DVR_StopGetFile. Please call NET_DVR_GetLastError to get the error code.	
Remarks:	Before calling this interface to download file, we can call the interface of searching record file (NET_DVR_FindFile_V30) to get file name. The interface have assigned the file to be downloaded currently. After calling it successfully, it needs to call starting play control command **NET_DVR_PLAYSTART** of NET_DVR_PlayBackControl to download file.	

Return to index

5.14.16 Download by time: NET_DVR_GetFileByTime

API:	LONG NET_DVR_GetFileByTime(LONG lUserID,LONG lChannel,

LPNET_DVR_TIME lpStartTime, LPNET_DVR_TIME lpStopTime, char *sSavedFileName)

Parameters:	[in] lUserID	The return value of NET_DVR_Login_V30
	[in] lChannel	Channel number
	[in] lpStartTime	Starting time
	[in] lpStopTime	Ending time
	[in] sSavedFileName	The files name saved in the computer after downloaded, it should be absolute path

Return: Return -1 if it is failed, and other values could be used as the parameter of functions NET_DVR_StopGetFile. Please call NET_DVR_GetLastError to get the error code.

Remarks: The API has assigned the file to be downloaded currently. After calling it successfully, it needs to call starting play control command **NET_DVR_PLAYSTART** of NET_DVR_PlayBackControl to download the file.

Return to index

5.14.17 Control the download state: NET_DVR_PlayBackControl

API: BOOL NET_DVR_PlayBackControl(LONG lPlayHandle,DWORD dwControlCode,DWORD dwInValue,DWORD *LPOutValue)

Parameters:	[in] lPlayHandle	Playing handle, the return value of NET_DVR_GetFileByName or NET_DVR_GetFileByTime
	[in] dwControlCode	Command to control video playback, details see to **the list table** below.
	[in] dwInValue	Configured parameter. if set file downloading progress(**NET_DVR_PLAYSETPOS**), it means progress value; if start to download (**NET_DVR_PLAYSTART**), it means offset (Byte).
	[out] LPOutValue	Obtained parameters, such as to get total time of current file downloading (command value: NET_DVR_GETTOTALTIME), this parameter is the obtained total time.

Macro Definition	Value	Implication
NET_DVR_PLAYSTART	1	Start downloading
NET_DVR_PLAYPAUSE	3	Pause
NET_DVR_PLAYRESTART	4	Resume
NET_DVR_PLAYSETPOS	12	Change the progress of the file download (valid when downloading by file)
NET_DVR_PLAYGETPOS	13	Get the progress of the file download (valid when downloading by file)

NET_DVR_GETTOTALFRAMES	16	Get the file current total downloaded frames(valid when downloading by file)
NET_DVR_GETTOTALTIME	17	Get the file current total downloaded time(valid when downloading by file)
NET_DVR_SET_DOWNLOAD_SPEED	28	Set download speed, stream control range: 0~32Mbps

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Whether the third parameter of this API requires to input value is related with the control command. It means download progress in NET_DVR_PLAYSETPOS command; when control command is starting to download (**NET_DVR_PLAYSTART**), value of the third parameter means offset of current file downloading. If the parameter's value is 0, it means downloading from file starting position; if this value isn't 0, it means offset value (Bytes). *Currently, DS-90xx and DS-81xx series DVR support resuming to download after interrupted.*

The fifth parameter of this API means corresponding parameter got by current control command. The control commands, NET_DVR_PLAYGETPOS, NET_DVR_GETTOTALFRAMES, and NET_DVR_GETTOTALTIME, can get the corresponding values by this parameter; details refer to the following table.

When command value is NET_DVR_PLAYGETPOS, to get file playback or download progress, 0-100 means normal progress value, value larger than 100 means playback or download is abnormal.

When getting the progress of download by time, DS-91xxHF-ST/DS-90xxHF-ST /DS-96xxHF-ST/DS-81xxHF-ST supports to get the progress of 0~100 and 200(exception), and other devices can get the progress of 0, 100(finished), and 200(exception).

Return to index

5.14.18 Stop downloading: NET_DVR_StopGetFile

API: BOOL NET_DVR_StopGetFile(LONG lFileHandle)

Parameters: [in] lFileHandle Playing handle, the return value of NET_DVR_GetFileByName or NET_DVR_GetFileByTime

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.14.19 Get the progess of the downloading:

NET_DVR_GetDownloadPos

API: int NET_DVR_GetDownloadPos(LONG lFileHandle)

Parameters: [in] lFileHandle Playing handle, the return value of NET_DVR_GetFileByName or NET_DVR_GetFileByTime

Return: -1 means it is failed; 0-100: the progress of the download; 100 means download finished; 200 means the network problem is abnormal. Please call NET_DVR_GetLastError to get the error code.

Remarks: The interface is used to get current progress when downloading by file name.

Return to index

Lock and unlock files recorded in the device

5.14.20 Lock files by file name: NET_DVR_LockFileByName

API: BOOL NET_DVR_LockFileByName(LONG lUserID, char *sLockFileName)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

[in] sLockFileName File name of which to be locked, the length should be less than 100 bytes

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Before calling the API to lock file, we can call NET_DVR_FindFile_V30 to get file name. When the file is locked, it will not be overlaid.

Return to index

5.14.21 Unlock files by file name: NET_DVR_UnlockFileByName

API: BOOL NET_DVR_UnlockFileByName(LONG lUserID, char *sUnlockFileName)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

[in] sUnlockFileName File name of which to be unlocked

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Before calling the API to lock file, we can call NET_DVR_FindFile_V30 to get file name.

Return to index

5.15 Manual Recording

5.15.1 Remotely start manual recording in the device:

NET_DVR_StartDVRRecord

API: BOOL NET_DVR_StartDVRRecord(LONG lUserID,LONG lChannel,LONG lRecordType)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] lChannel	Channel number: *0x00ff means all analog channels,* *0xff00 means all digital channels,* *0xffff means all analog and digital channels*
[in] lRecordType	Recording type: 0- manual, 1- alarm, 2- postback, 3- signal, 4- motion detection, 5- tampering

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Not all devices support to set lRecordType. If the device doesn't support it, the type will default to manual recording.

If the channel has opened the schedule recording, and then call this API to start manual recording, the operation will be invalid, and the device will keep the schedule recording. At the moment, if call NET_DVR_GetDVRWorkState_V30 to get the recording state,the value of byRecordStatic (parameter of the structre NET_DVR_CHANNELSTATE_V30) will be still 1 (being recording). Then if call NET_DVR_StopDVRRecord to stop manual recording, it will stop the schedule recording. Afterward, if call NET_DVR_StartDVRRecord again, the device will start manual recording. Then, if call NET_DVR_StopDVRRecord to stop the manual recording, and reboot the device, the device will resume the schedule recording.

Return to index

5.15.2 Remotely stop manual recording: NET_DVR_StopDVRRecord

API: BOOL NET_DVR_StopDVRRecord(LONG lUserID,LONG lChannel)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] lChannel	Channel number: *0x00ff means all analog channels,* *0xff00 means all digital channels,* *0xffff means all analog and digital channels*

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.16 Alarm of Arming Mode

Set the callback function of the alarm message uploaded by the device

5.16.1 Register the callback function to receive the alarm message:

NET_DVR_SetDVRMessageCallBack_V30

API: BOOL NET_DVR_SetDVRMessageCallBack_V30(MSGCallBack fMessageCallBack, void* pUser)

Parameters:

[in] fMessageCallBack Callback function

[in] pUser User data

typedef void(CALLBACK *MSGCallBack)(LONG lCommand,NET_DVR_ALARMER *pAlarmer, char *pAlarmInfo,DWORD dwBufLen,void *pUser)

[out] lCommand Message type, see to the list table below.

[out] pAlarmer The device that uploads the message

[out] pAlarmInfo The buffer to save uploaded alarm message

[out] dwBufLen The buffer size

[out] pUser User data

Macro Definition	Value	Implication
COMM_ALARM	0x1100	Alarm message uploading of the devices supported by the SDK version lower than V3.0
COMM_IPC_AUXALARM_RESULT	0x2820	PIR alarm, wireless alarm, or calling for help alarm upload

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: The first parameter(lCommand)and the third parameter (pAlarmInfo)is closely related, as follows:

lCommand	Uploaded Content	pAlarmInfo
COMM_ALARM	Alarm message of the devices supported by the SDK version lower than V3.0	NET_DVR_ALARMINFO
COMM_IPC_AUXALARM_RESULT	PIR alarm, wireless alarm, or calling for help alarm	NET_IPC_AUXALARM_RESULT

Return to index

Arm and disarm

5.16.2 Setup the uploading channel of alarm meassge:

NET_DVR_SetupAlarmChan_V30

API: BOOL NET_DVR_SetupAlarmChan_V30(LONG lUserID)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

Return: -1 means false, other values are as handle parameters of function NET_DVR_CloseAlarmChan. Please call NET_DVR_GetLastError to get the error code.

Remarks: Before calling this API to start arming, it requires to call NET_DVR_SetDVRMessageCallBack_V30 to get the uploaded alarm message.

Return to index

5.16.3 Close the uploading channel of alarm message:

NET_DVR_CloseAlarmChan_V30

API: BOOL NET_DVR_CloseAlarmChan_V30(LONG lAlarmHandle)

Parameters: [in] lAlarmHandle The return value of NET_DVR_SetupAlarmChan_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.17 Alarm of Listening Mode

Listening

5.17.1 Start listening to receive the alarm message uploaded actively by the device: NET_DVR_StartListen_V30

API: LONG NET_DVR_StartListen_V30(char *sLocalIP, WORD wLocalPort, MSGCallBack DataCallback, void* pUserData)

Parameters: [in] sLocalIP Local IP, can set to NULL

[in] wLocalPort	Local listening port number of PC, configured by user, should be consistent with that set in device
[in] DataCallback	Callback function, can't be NULL
[in] pUserData	User data

typedef void(CALLBACK *MSGCallBack)(LONG lCommand,NET_DVR_ALARMER *pAlarmer,char *pAlarmInfo,DWORD dwBufLen,void *pUser)

[out] lCommand	Message type, see to **the list table** below.
[out] pAlarmer	The device that uploads the message
[out] pAlarmInfo	The buffer to save uploaded alarm message
[out] dwBufLen	The buffer size
[out] pUser	User data

Macro Definition	Value	Implication
COMM_ALARM	0x1100	Alarm message uploading of the devices supported by the SDK version lower than V3.0
COMM_IPC_AUXALARM_RESULT	0x2820	PIR alarm, wireless alarm, or calling for help alarm upload

Return: Return -1 if it is failed, other values are as handle parameters of function NET_DVR_StopListen_V30. Please call NET_DVR_GetLastError to get the error code.

Remarks: The total number of listening supported by the SDK is 512.
In order to make PC able to receive alarm message uploaded actively by device, it requires to set "remote management host IP address" or "remote alarm host IP address" in network configuration of device to same with the IP address of PC (the parameter *sLocalIP* in the API) , and set "remote management host port" or "remote alarm host port" to same with the listening port of the PC (the parameter *wLocalPort* in the API)
The callback in the API is higher priority than other callback function. That is, if the callback function set here, other callback function will not able to receive the alarm information.
The first parameter(lCommand) and third parameter(pAlarmInfo) of this interface callback function is related:

lCommand	Uploaded Content	pAlarmInfo
COMM_ALARM	Alarm message of the devices supported by the SDK version lower than V3.0	NET_DVR_ALARMINFO
COMM_IPC_AUXALARM_RESULT	PIR alarm, wireless alarm, or calling for help alarm	NET_IPC_AUXALARM_RESULT

Return to index

5.17.2 Stop listening (support multi-thread): NET_DVR_StopListen_V30

API: BOOL NET_DVR_StopListen_V30(LONG lListenHandle)

Parameters: [in] lListenHandle Listening handle, the return value of NET_DVR_StartListen_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.18 PTZ Control

PTZ control operation

5.18.1 PTZ control (requires starting live view firstly):

NET_DVR_PTZControl

API: BOOL NET_DVR_PTZControl(LONG lRealHandle,DWORD dwPTZCommand,DWORD dwStop)

Parameters: [in] lRealHandle The return value of NET_DVR_RealPlay_V30

[in] dwPTZCommand PTZ control command, see to the list table

[in] dwStop PTZ stop or start operation: 0-start, 1-stop

Macro Definition	Value	Implication
LIGHT_PWRON	2	Connect lighting power
WIPER_PWRON	3	Turn on wiper switch
FAN_PWRON	4	Turn on fan switch
HEATER_PWRON	5	Turn on heater switch
AUX_PWRON1	6	Turn on auxiliary device switch
AUX_PWRON2	7	Turn on auxiliary device switch
ZOOM_IN	11	Focal distance enlarge(Magnification enlarge)
ZOOM_OUT	12	Focal distance decrease(Magnification decrease)
FOCUS_NEAR	13	Focus front
FOCUS_FAR	14	Focus back
IRIS_OPEN	15	Aperture enlarge
IRIS_CLOSE	16	Aperture narrow
TILT_UP	21	Tilt up

TILT_DOWN	22	Tilt down
PAN_LEFT	23	Pan left
PAN_RIGHT	24	Pan right
UP_LEFT	25	Tilt up and pan left
UP_RIGHT	26	Tilt up and pan right
DOWN_LEFT	27	Tilt down and pan left
DOWN_RIGHT	28	Tilt down and pan right
PAN_AUTO	29	PTZ scans left and right automatically

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every movement of operating PTZ needs to call the interface twice: start and stop control, decided by the last parameter(dwStop) in the interface.

It needs to start preview before calling this interface. Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.

If decoder configuration of the current device doesn't match the PTZ device, it needs to re-configure the decoder parameter. If the PTZ doesn't support the parameter, it is not able to control PTZ.

Default: PTZ turns around at the maximum speed.

Return to index

5.18.2 PTZ control (not require live view before calling it):

NET_DVR_PTZControl_Other

API: BOOL NET_DVR_PTZControl_Other(LONG lUserID,LONG lChannel,DWORD dwPTZCommand,DWORD dwStop)

Parameters:
[in] lUserID The return value of NET_DVR_Login_V30
[in] lChannel Channel number
[in] dwPTZCommand PTZ control command, see to the list table
[in] dwStop PTZ stop or start operation: 0-start, 1-stop

Macro Definition	Value	Implication
LIGHT_PWRON	2	Connect lighting power
WIPER_PWRON	3	Turn on wiper switch
FAN_PWRON	4	Turn on fan switch
HEATER_PWRON	5	Turn on heater switch
AUX_PWRON1	6	Turn on auxiliary device switch
AUX_PWRON2	7	Turn on auxiliary device switch

ZOOM_IN	11	Focal distance enlarge(Magnification enlarge)
ZOOM_OUT	12	Focal distance decrease(Magnification decrease)
FOCUS_NEAR	13	Focus front
FOCUS_FAR	14	Focus back
IRIS_OPEN	15	Aperture enlarge
IRIS_CLOSE	16	Aperture narrow
TILT_UP	21	Tilt up
TILT_DOWN	22	Tilt down
PAN_LEFT	23	Pan left
PAN_RIGHT	24	Pan right
UP_LEFT	25	Tilt up and pan left
UP_RIGHT	26	Tilt up and pan right
DOWN_LEFT	27	Tilt down and pan left
DOWN_RIGHT	28	Tilt down and pan right
PAN_AUTO	29	PTZ scans left and right automatically

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every movement of operating PTZ needs to call the interface twice: start and stop control, decided by the last parameter(dwStop) in the interface.
It needs to start preview before calling this interface. Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.
If decoder configuration of the current device doesn't match the PTZ device, it needs to re-configure the decoder parameter. If the PTZ doesn't support the parameter, it is not able to control PTZ.
Default: PTZ turns around at the maximum speed.
If call NET_DVR_PTZControl to control PTZ, after the device receive the command and PTZ runs according to the command, it will return success to client when PTZ runs normally, and return false when PTZ failed to run. While, if call NET_DVR_PTZControl_Other, it will return success immediately after the device receive the command.

Return to index

5.18.3 PTZ control with speed (requires starting live view firstly):

NET_DVR_PTZControlWithSpeed

API: BOOL NET_DVR_PTZControlWithSpeed(LONG lRealHandle, DWORD

dwPTZCommand, DWORD dwStop, DWORD dwSpeed)

Parameters:

[in] lRealHandle	The return value of NET_DVR_RealPlay_V30.
[in] dwPTZCommand	PTZ control command, see to **the list table.**
[in] dwStop	PTZ stop or start operation: 0-start, 1-stop.
[in] dwSpeed	PTZ control speed, please set it according to different speed control value of PTZ decoder. Value range: [1,7].

Macro Definition	Value	Implication
LIGHT_PWRON	2	Connect lighting power
WIPER_PWRON	3	Turn on wiper switch
FAN_PWRON	4	Turn on fan switch
HEATER_PWRON	5	Turn on heater switch
AUX_PWRON1	6	Turn on auxiliary device switch
AUX_PWRON2	7	Turn on auxiliary device switch
ZOOM_IN	11	Focal distance enlarge(Magnification enlarge)
ZOOM_OUT	12	Focal distance decrease(Magnification decrease)
FOCUS_NEAR	13	Focus front
FOCUS_FAR	14	Focus back
IRIS_OPEN	15	Aperture enlarge
IRIS_CLOSE	16	Aperture narrow
TILT_UP	21	Tilt up
TILT_DOWN	22	Tilt down
PAN_LEFT	23	Pan left
PAN_RIGHT	24	Pan right
UP_LEFT	25	Tilt up and pan left
UP_RIGHT	26	Tilt up and pan right
DOWN_LEFT	27	Tilt down and pan left
DOWN_RIGHT	28	Tilt down and pan right
PAN_AUTO	29	PTZ scans left and right automatically

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every movement of operating PTZ needs to call the API twice: start and stop control, decided by the last parameter(dwStop) in the API.
It needs to start live view before calling this API. Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.
If decoder configuration of the current device doesn't match the PTZ device, it

needs to re-configure the decoder parameter. If the PTZ doesn't support the parameter, it will not able to control PTZ.

Return to index

5.18.4 PTZ control with speed (not require live view before calling it):

NET_DVR_PTZControlWithSpeed_Other

API: BOOL NET_DVR_PTZControlWithSpeed(LONG lUserID, LONG lChannel, DWORD dwPTZCommand, DWORD dwStop, DWORD dwSpeed)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30.
[in] lChannel	Channel number.
[in] dwPTZCommand	PTZ control command, see to **the list table**.
[in] dwStop	PTZ stop or start operation: 0-start, 1-stop.
[in] dwSpeed	PTZ control speed, please set it according to different speed control value of PTZ decoder. Value range: [1,7].

Macro Definition	Value	Implication
LIGHT_PWRON	2	Connect lighting power
WIPER_PWRON	3	Turn on wiper switch
FAN_PWRON	4	Turn on fan switch
HEATER_PWRON	5	Turn on heater switch
AUX_PWRON1	6	Turn on auxiliary device switch
AUX_PWRON2	7	Turn on auxiliary device switch
ZOOM_IN	11	Focal distance enlarge(Magnification enlarge)
ZOOM_OUT	12	Focal distance decrease(Magnification decrease)
FOCUS_NEAR	13	Focus front
FOCUS_FAR	14	Focus back
IRIS_OPEN	15	Aperture enlarge
IRIS_CLOSE	16	Aperture narrow
TILT_UP	21	Tilt up
TILT_DOWN	22	Tilt down
PAN_LEFT	23	Pan left
PAN_RIGHT	24	Pan right
UP_LEFT	25	Tilt up and pan left
UP_RIGHT	26	Tilt up and pan right
DOWN_LEFT	27	Tilt down and pan left

DOWN_RIGHT	28	Tilt down and pan right
PAN_AUTO	29	PTZ scans left and right automatically

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every movement of operating PTZ needs to call the API twice: start and stop control, decided by the last parameter(dwStop) in the API.
It doesn't need to start live view before calling this API. Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.
If decoder configuration of the current device doesn't match the PTZ device, it needs to re-configure the decoder parameter. If the PTZ doesn't support the parameter, it will not able to control PTZ.

Return to index

PTZ preset operation

5.18.5 PTZ preset operation (requires starting live view firstly):

NET_DVR_PTZPreset

API: BOOL NET_DVR_PTZPreset(LONG lRealHandle,DWORD dwPTZPresetCmd,DWORD dwPresetIndex)

Parameters:

[in] lRealHandle	The return value of NET_DVR_RealPlay_V30
[in] dwPTZPresetCmd	The command to operate preset, see to the list table below.
[in] dwPresetIndex	The number of preset, it supports max 255 presets, the number starts from 1

Macro Definition	Value	Implication
SET_PRESET	8	Set preset point
CLE_PRESET	9	Clear preset point
GOTO_PRESET	39	Goto preset point

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.
If PTZ decoder configuration of the current device doesn't match the PTZ device, it needs to re-configure the decoder parameter.
If the PTZ doesn't support the parameter, it will not able to control PTZ.

Return to index

5.18.6 PTZ preset operation: NET_DVR_PTZPreset_Other

API: BOOL NET_DVR_PTZPreset_Other(LONG lUserID,LONG lChannel,DWORD dwPTZPresetCmd,DWORD dwPresetIndex))

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] lChannel	Channel number
[in] dwPTZPresetCmd	The command to operate preset, see to the list table below.
[in] dwPresetIndex	The number of preset, it supports max 255 presets, the number starts from 1

Macro Definition	Value	Implication
SET_PRESET	8	Set preset point
CLE_PRESET	9	Clear preset point
GOTO_PRESET	39	Goto preset point

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.
If decoder configuration of the current device doesn't match the PTZ device, it needs to re-configure the decoder parameter. If the PTZ doesn't support the parameter, it will not able to control PTZ.
If call NET_DVR_PTZPreset to control PTZ, after the device receive the command and PTZ runs according to the command, it will return success to client when PTZ runs normally, and return false when PTZ failed to run. While, if call NET_DVR_PTZPreset_Other, it will return success immediately after the device receive the command.

Return to index

PTZ Patrol operation

5.18.7 PTZ patrol operation (requires starting live view firstly): NET_DVR_PTZPCruise

API: BOOL NET_DVR_PTZCruise(LONG lRealHandle,DWORD dwPTZCruiseCmd,BYTE byCruiseRoute, BYTE byCruisePoint, WORD wInput)

Parameters:

[in] lRealHandle	The return value of NET_DVR_RealPlay_V30
[in] dwPTZCruiseCmd	The commands to control PTZ patrol, see to the

list table.

[in] byCruiseRoute — The number of patrol route, it supports maximum 32 routes, the number starts from 1

[in] byCruisePoint — The number of preset, it supports maximum 32 presets, the number starts from 1

[in] wInput — The value is different for different commands, preset(maximum is 128), dwell time (maximum is 255), Speed (maximum is 40)

Macro Definition	Value	Implication
FILL_PRE_SEQ	30	Add preset to the patrol sequence
SET_SEQ_DWELL	31	Set dwell time of the patrol point
SET_SEQ_SPEED	32	Set patrol speed
CLE_PRE_SEQ	33	Delete preset point from the patrol sequence
RUN_SEQ	37	Start running the patrol
STOP_SEQ	38	Stop running the patrol

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.
If decoder configuration of the current device doesn't match the PTZ device, it needs to re-configure the decoder parameter. If the PTZ doesn't support the parameter, it will not able to control PTZ.

Return to index

5.18.8 PTZ patrol operation: NET_DVR_PTZPCruise_Other

API: BOOL NET_DVR_PTZCruise_Other(LONG lUserID,LONG lChannel,DWORD dwPTZCruiseCmd,BYTE byCruiseRoute, BYTE byCruisePoint, WORD wInput)

Parameters:

[in] lUserID — The return value of NET_DVR_Login_V30

[in] lChannel — Channel number

[in] dwPTZCruiseCmd — The commands to control PTZ patrol, see to the list table.

[in] byCruiseRoute — The number of patrol route, it supports maximum 32 routes, the number starts from 1

[in] byCruisePoint — The number of preset, it supports maximum 32 presets, the number starts from 1

[in] wInput — The value is different for different commands, preset(maximum is 128), dwell time (maximum is 255), Speed (maximum is 40)

Macro Definition	Value	Implication
FILL_PRE_SEQ	30	Add preset into patrol sequence
SET_SEQ_DWELL	31	Set dwell time of the patrol point
SET_SEQ_SPEED	32	Set patrol speed
CLE_PRE_SEQ	33	Delete preset from the patrol sequence
RUN_SEQ	37	Start running the patrol
STOP_SEQ	38	Stop running the patrol

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.
If decoder configuration of the current device doesn't match the PTZ device, it needs to re-configure the decoder parameter. If the PTZ doesn't support the parameter, it will not able to control PTZ.

Return to index

PTZ pattern operation

5.18.9 PTZ pattern operation(requires starting live view firstly):

NET_DVR_PTZTrack

API: BOOL NET_DVR_PTZTrack(LONG lRealHandle, DWORD dwPTZTrackCmd)

Parameters: [in] lRealHandle The return value of NET_DVR_RealPlay_V30.
[in] dwPTZTrackCmd The command to control PTZ pattern, see to the list table below.

Macro Definition	Value	Implication
STA_MEM_CRUISE	34	Start recording pattern
STO_MEM_CRUISE	35	Stop recording pattern
RUN_CRUISE	36	Start running according to the pattern

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.
If decoder configuration of the current device doesn't match the PTZ device, it needs to re-configure the decoder parameter. If the PTZ doesn't support the parameter, it will not able to control PTZ.

Return to index

5.18.10 PTZ pattern operation: NET_DVR_PTZTrack_Other

API: BOOL NET_DVR_PTZTrack_Other(LONG lUserID, LONG lChannel, DWORD dwPTZTrackCmd)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30.
[in] lChannel	Channel number
[in] dwPTZTrackCmd	The command to control PTZ pattern, see to the list table below.

Macro Definition	Value	Implication
STA_MEM_CRUISE	34	Start recording pattern
STO_MEM_CRUISE	35	Stop recording pattern
RUN_CRUISE	36	Start running according to the pattern

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: Every operation command corresponds to the control code between the device and the PTZ, and the device will send control code to PTZ based on the current decoder type and address.
If decoder configuration of the current device doesn't match the PTZ device, it needs to re-configure the decoder parameter. If the PTZ doesn't support the parameter, it will not able to control PTZ.
If call NET_DVR_PTZTrack to control PTZ, after the device receive the command and PTZ runs according to the command, it will return success to client when PTZ runs normally, and return false when PTZ failed to run. While, if call NET_DVR_PTZTrack_Other, it will return success immediately after the device receive the command.

Return to index

Transparent PTZ Control

5.18.11 Tansparent PTZ control(requires starting live view firstly): NET_DVR_TransPTZ

API: BOOL NET_DVR_TransPTZ(LONG lRealHandle,char *pPTZCodeBuf,DWORD dwBufSize)

Parameters:

[in] lRealHandle	The return value of NET_DVR_RealPlay_V30
[in] pPTZCodeBuf	Pointer of the buffer to save PTZ control code
[in] dwBufSize	Length of PTZ control code

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It supports sending the control command code directly to the PTZ decoder through the device by calling this API, and it's not neccessary to configure the decoder parameter in the device.

Return to index

5.18.12 Tansparent PTZ control: NET_DVR_TransPTZ_Other

API: BOOL NET_DVR_TransPTZ(LONG lUserID,LONG lChannel,char *pPTZCodeBuf,DWORD dwBufSize)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] lChannel	Channel number
[in] pPTZCodeBuf	Pointer of the buffer to save PTZ control code
[in] dwBufSize	Length of PTZ control code

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It supports sending the control command code directly to the PTZ decoder through the device by calling this API, and it's not neccessary to configure the decoder parameter in the device.

Return to index

PTZ Region Zoom control

5.18.13 PTZ control to enlarge or narrow the selected image region (requires starting live view firstly): NET_DVR_PTZSelZoomIn

API: BOOL NET_DVR_PTZSelZoomIn(LONG lRealHandle, LPNET_DVR_POINT_FRAME pStruPointFrame);

Parameters:

[in] lRealHandle	The return value of NET_DVR_RealPlay_V30
[in] pStruPointFrame	Image region position

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API is used to realize 3D intelligent positioning function, and it will take effect just when the front-end device supports it.

If suppose the frame of current live view image is 352 * 288, the origin point is the upper left corner of the display box. The calculation method of coordinate value in parameter pStruPointFrame (here take X-axis as an example):

xTop = (upper left point of the region currently selected by mouse) * 255/352.

The zoom-in condition: xBottom - xTop > 2.
The zoom-out condition: xBottom - xTop > 0 and yBottom - yTop > 0.

Return to index

5.18.14 PTZ control to enlarge or narrow the selected image region:

NET_DVR_PTZSelZoomIn_Ex

API: BOOL NET_DVR_PTZSelZoomIn_EX(LONG lUserID, LONG lChannel, LPNET_DVR_POINT_FRAME pStruPointFrame)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] lChannel	Channel number
[in] pStruPointFrame	Image region position

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API is used to realize 3D intelligent positioning function, and it will take effect just when the front-end device supports it.
If suppose the frame of current live view image is 352 * 288, the origin point is the upper left corner of the display box. The calculation method of coordinate value in parameter pStruPointFrame (here take X-axis as an example):
xTop = (upper left point of the region currently selected by mouse) * 255/352.
The zoom-in condition: xBottom - xTop > 2.
The zoom-out condition: xBottom - xTop > 0 and yBottom - yTop > 0.

Return to index

Get patrol path of IP dome

5.18.15 Get patrol path of PTZ: NET_DVR_GetPTZCruise

API: BOOL NET_DVR_GetPTZCruise(LONG lUserID, LONG lChannel, LONG lCruiseRoute, LPNET_DVR_CRUISE_RET lpCruiseRet)

Parameters:

[in]lUserID	The return value of NET_DVR_Login_V30
[in]lChannel	Channel number
[in]lCruiseRoute	Path serial number
[out]dwInBufferSize	Patrol path

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.19 IPC remote control

5.19.1 Control one-key focus: NET_DVR_FocusOnePush

API: BOOL NET_DVR_FocusOnePush(LONG lUserID, LONG lChannel)

Parameters:

[in]lUserID	The return value of NET_DVR_Login_V30
[in]lChannel	Channel number

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

返回目录

5.19.2 Reset lens motor default location: NET_DVR_ResetLens

API: BOOL NET_DVR_ResetLens(LONG lUserID, LONG lChannel)

Parameters:

[in]lUserID	The return value of NET_DVR_Login_V30
[in]lChannel	Channel number

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

返回目录

5.19.3 Control the remote controller: NET_DVR_RemoteControl

API: BOOL NET_DVR_RemoteControl(LONG lUserID, DWORD dwCommand, LPVOID lpInBuffer, DWORD dwInBufferSize)

Parameters:

[in]lUserID	The return value of NET_DVR_Login_V30
[in]dwCommand	Control command, please kindly refer to the list below
[in]lpInBuffer	Buffer that saves the input parameters, the content is related to the control command, details listed below
[in]dwInBufferSize	Size of the buffer (unit: byte)

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: The lpInBuffer structures and command numbers are different according to the different control, and they are listed as below:

dwCommand Macro Definition	dwCommand Value	Control Function	lpInBuffer Structure
NET_DVR_REMOTECONTROL_ALARM	3205	Set the remote controller armed	NET_DVR_REMOTECONTROL_ALARM_PARAM

NET_DVR_REMOTECONTROL_DISALARM	3206	Set the remote controller disarmed	NET_DVR_REMOTECONTROL_ALARM_PARAM
NET_DVR_REMOTECONTROL_STUDY	3207	Set the remote controller study	NET_DVR_REMOTECONTROL_STUDY_PARAM
NET_DVR_WIRELESS_ALARM_STUDY	3208	Remotely contol wireless alarm study	NET_DVR_WIRELESS_ALARM_STUDY_PARAM

返回目录

5.20 Voice Talk, Forwarding and Broadcast

Voice talk

5.20.1 Start voice talk: NET_DVR_StartVoiceCom_V30

API: LONG NET_DVR_StartVoiceCom_V30(LONG lUserID, DWORD dwVoiceChan, BOOL bNeedCBNoEncData, fVoiceDataCallBack cbVoiceDataCallBack, void* pUser)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] dwVoiceChan	Audio channel number, starts from 1
[in] bNeedCBNoEncData	The audio type that you want to callback: 0- decoded audio data, 1- PCM original data before encoded
[in] fVoiceDataCallBack	Audio data callback function
[in] pUser	User data

typedef void(CALLBACK *fVoiceDataCallBack)(LONG lVoiceComHandle,char *pRecvDataBuffer,DWORD dwBufSize, BYTE byAudioFlag,void *pUser)

[out] lVoiceComHandle	The return value of NET_DVR_StartVoiceCom_V30
[out]pRecvDataBuffer	Pointer of the buffer to save the audio data
[out]dwBufSize	The size of audio data
[out]byAudioFlag	Audio data type: 0- collected by local PC, 1- sent from the device
[out]pUser	User data

Return: Return -1 if it is failed, and other values are as handle parameters of functions like NET_DVR_StopVoiceCom. Please call NET_DVR_GetLastError to get the error code.

Remarks: Under Windows 7 system, if no external audio devices ,this interface will return false.

Before calling this API, it supports to get the audio encoding format (NET_DVR_COMPRESSION_AUDIO) of the device, by calling NET_DVR_GetDVRConfig.

If current encoding format is OggVorbis, audio data sampling frequency is 16000, 16 bytes sampling and monophonic. Audio playing format should be defined as following:

```
const int SAMPLES_PER_SECOND = 16000;
const int CHANNEL = 1;
const int BITS_PER_SAMPLE = 16;
WAVEFORMATEX m_wavFormatEx;
m_wavFormatEx.cbSize = sizeof(m_wavFormatEx);
m_wavFormatEx.nBlockAlign = CHANNEL * BITS_PER_SAMPLE / 8;
m_wavFormatEx.nChannels = CHANNEL;
m_wavFormatEx.nSamplesPerSec = SAMPLES_PER_SECOND;
m_wavFormatEx.wBitsPerSample = BITS_PER_SAMPLE;
m_wavFormatEx.nAvgBytesPerSec =
SAMPLES_PER_SECOND*m_wavFormatEx.nBlockAlign
```

If current encoding format is G711 or G726, the audio data sampling frequency is 8000, 16 bytes sampling and monophonic. Audio playing format should be defined as following:

```
const int SAMPLES_PER_SECOND_G711_MU = 8000;
const int CHANNEL = 1;
const int BITS_PER_SAMPLE = 16;
WAVEFORMATEX m_wavFormatEx;
m_wavFormatEx.cbSize = sizeof(m_wavFormatEx);
m_wavFormatEx.nBlockAlign = CHANNEL * BITS_PER_SAMPLE / 8;
m_wavFormatEx.nChannels = CHANNEL;
m_wavFormatEx.nSamplesPerSec = SAMPLES_PER_SECOND_G711_MU;
m_wavFormatEx.wBitsPerSample = BITS_PER_SAMPLE;
m_wavFormatEx.nAvgBytesPerSec = SAMPLES_PER_SECOND_G711_MU*
m_wavFormatEx.nBlockAlign;
```

Return to index

5.20.2 Set the client volume of voice talk:

NET_DVR_SetVoiceComClientVolume

API: BOOL NET_DVR_SetVoiceComClientVolume(LONG lVoiceComHandle, WORD wVolume)

Parameters:

[in] lVoiceComHandle	The return value of NET_DVR_StartVoiceCom_V30
[in] wVolume	The volume value to set, value range: [0,0xffff]

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.20.3 Stop voice talk: NET_DVR_StopVoiceCom

API: BOOL NET_DVR_StopVoiceCom(LONG lVoiceComHandle)

Parameters:

[in]	lVoiceComHandle	The return value of NET_DVR_StartVoiceCom_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

Voice fowarding

5.20.4 Start voice forwarding, to get the encoded audio data: NET_DVR_StartVoiceCom_MR_V30

API: LONG NET_DVR_StartVoiceCom_MR_V30(LONG lUserID, DWORD dwVoiceChan, fVoiceDataCallBack cbVoiceDataCallBack, void* pUser)

Parameters:

[in]	lUserID	The return value of NET_DVR_Login_V30
[in]	dwVoiceChan	Audio channel number, starts from 1
[in]	fVoiceDataCallBack	Callback function of audio data, the obtained data is encoded, and requires to call the audio decoding APIs (refer to Audio Encoding & Decoding chapter) to get PCM data
[in]	pUser	User data

typedef void(CALLBACK *fVoiceDataCallBack)(LONG lVoiceComHandle,char *pRecvDataBuffer, DWORD dwBufSize,BYTE byAudioFlag,void*pUser)

[out]	lVoiceComHandle	The return value of NET_DVR_StartVoiceCom_MR_V30
[out]	pRecvDataBuffer	Pointer of the buffer to save the audio data
[out]	dwBufSize	The size of audio data
[out]	byAudioFlag	Audio data type: 1- audio data sent from the device
[out]	pUser	User data

Return: Return -1 if it is failed, and other values are as handle parameters of functions like NET_DVR_StopVoiceCom. Please call NET_DVR_GetLastError to get the error code.

Remarks: Under Windows 7 system, if no external audio devices ,this interface will return false.

Before calling this API, it supports to get the audio encoding format (NET_DVR_COMPRESSION_AUDIO) of the device, by calling NET_DVR_GetDVRConfig.

If current encoding format is OggVorbis, audio data sampling frequency is 16000, 16 bytes sampling and monophonic. Audio playing format should be defined as following:

```
const int SAMPLES_PER_SECOND = 16000;
const int CHANNEL = 1;
const int BITS_PER_SAMPLE = 16;
WAVEFORMATEX m_wavFormatEx;
m_wavFormatEx.cbSize = sizeof(m_wavFormatEx);
m_wavFormatEx.nBlockAlign = CHANNEL * BITS_PER_SAMPLE / 8;
m_wavFormatEx.nChannels = CHANNEL;
m_wavFormatEx.nSamplesPerSec = SAMPLES_PER_SECOND;
m_wavFormatEx.wBitsPerSample = BITS_PER_SAMPLE;
m_wavFormatEx.nAvgBytesPerSec =
SAMPLES_PER_SECOND*m_wavFormatEx.nBlockAlign
```

If current encoding format is G711 or G726, the audio data sampling frequency is 8000, 16 bytes sampling and monophonic. Audio playing format should be defined as following:

```
const int SAMPLES_PER_SECOND_G711_MU = 8000;
const int CHANNEL = 1;
const int BITS_PER_SAMPLE = 16;
WAVEFORMATEX m_wavFormatEx;
m_wavFormatEx.cbSize = sizeof(m_wavFormatEx);
m_wavFormatEx.nBlockAlign = CHANNEL * BITS_PER_SAMPLE / 8;
m_wavFormatEx.nChannels = CHANNEL;
m_wavFormatEx.nSamplesPerSec = SAMPLES_PER_SECOND_G711_MU;
m_wavFormatEx.wBitsPerSample = BITS_PER_SAMPLE;
m_wavFormatEx.nAvgBytesPerSec = SAMPLES_PER_SECOND_G711_MU*
m_wavFormatEx.nBlockAlign;
```

Return to index

5.20.5 Forward audio data to the device:

NET_DVR_VoiceComSendData

API: BOOL NET_DVR_VoiceComSendData(LONG lVoiceComHandle, char *pSendBuf, DWORD dwBufSize)

Parameters:

[in] lVoiceComHandle	The return value of NET_DVR_StartVoiceCom_MR_V30
[in] pSendBuf	Pointer of voice data buffer

	[in] dwBufSize	Size of voice data, which is 80 bytes if the audio format is OggVorbis, or 160 bytes if the audio format is G711.

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API is used to send encoded audio data to the device. If the audio data is original PCM format, please call NET_DVR_EncodeG722Frame (for OggVorbis) or NET_DVR_EncodeG711Frame (for G711) to encode the data and then send to the device.

Return to index

5.20.6 Stop voice forwarding: NET_DVR_StopVoiceCom

API: BOOL NET_DVR_StopVoiceCom (LONG lVoiceComHandle)

Parameters:	[in] lVoiceComHandle	The return value of NET_DVR_StartVoiceCom_MR_V30

Return: Return TRUE on success, FALSE on failure.

Remarks:

Return to index

Voice broadcast

5.20.7 Start to collect audio data in PC-end for voice broadcast: NET_DVR_ClientAudioStart_V30

API: BOOL NET_DVR_ClientAudioStart_V30(fVoiceDataCallBack cbVoiceDataCallBack, void *pUser)

Parameters:	[in] fVoiceDataCallBack	Callback function of audio data
	[in] pUser	User data
	typedef void(CALLBACK *fVoiceDataCallBack)(char *pRecvDataBuffer,DWORD dwBufSize, void *pUser)	
	[out] pRecvDataBuffer	Pointer of the buffer to save the audio data collected from local PC.
	[out]dwBufSize	The size of audio data
	[out] pUser	User data

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: On Windows 7 OS, the API will return FALSE if there is no peripheral audio device in the PC. To achieve audio broadcast, should call firstly

NET_DVR_ClientAudioStart_V30 to collect audio data form local PC, and call NET_DVR_AddDVR_V30 to add device one by one, and then it will transfer the collected data to the addes devices.

Return to index

5.20.8 Add one voice channel of the device to the broadcast group:

NET_DVR_AddDVR_V30

API: LONG NET_DVR_AddDVR_V30(LONG lUserID, DWORD dwVoiceChan)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

[in] dwVoiceChan The voice channel number, starts from 1

Return: Return -1 if it is failed, and other values could be used as a parameter of NET_DVR_DelDVR_V30. Please call NET_DVR_GetLastError to get the error code.

Remarks: To achieve voice broadcast, please call NET_DVR_ClientAudioStart_V30 firstly to start collecting audio data of local PC, and then call NET_DVR_AddDVR_V30 to add device one by one, and transfer the collected audio data to the added devices in the meantime.

It supports to add max 512 devices to the broadcast group by the SDK.

Return to index

5.20.9 Delete the voice channel of the device from the broadcast group:

NET_DVR_DelDVR_V30

API: LONG NET_DVR_DelDVR_V30(LONG lUserID)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.20.10 Stop collecting audio data in PC-end for the broadcast:

NET_DVR_ClientAudioStop

API: BOOL NET_DVR_ClientAudioStop()

Parameters: None

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

Encode or decode the audio data

Encode or decode the OggVorbis audio

5.20.11 Initialize the audio encoding resource:

NET_DVR_InitG722Encoder

API: void* NET_DVR_InitG722Encoder()

Parameters: None

Return: Return -1 if it is failed, and the other is used as the handle of audio encoding. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.20.12 Encode the PCM audio to G722 format:

NET_DVR_EncodeG722Frame

API: BOOL NET_DVR_EncodeG722Frame(void *pEncodeHandle,unsigned char* pInBuffer, unsigned char* pOutBuffer)

Parameters:

[in] pEncodeHandle	Audio encoding handle, the return value of NET_DVR_InitG722Encoder
[in] InBuffer	Input buffer, PCM data is 16000 sample rate, 16 bit, Mono, and the size of input data should be 1280 bytes
[out] pOutBuffer	Output buffer, the size of output encoded data is 80 bytes

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It is used mainly for voice talking and voice forwarding.
When you want to transfer the original audio data from client to the device, please call this API to encode the data and then transfer the encoded data to the device. When the client get the encoded voice stream from device, you

can call NET_DVR_DecodeG722Frame to decode the data. Before calling the encoding and decoding functions, it requires initial operation (NET_DVR_InitG722Encoder or NET_DVR_InitG722Decoder), and after calling them, please release the resource by calling NET_DVR_ReleaseG722Encoder or NET_DVR_ReleaseG722Decoder.

Return to index

5.20.13 Release the audio encoding resource:

NET_DVR_ReleaseG722Encoder

API:	void NET_DVR_ReleaseG722Encoder(void *pEncodeHandle)	
Parameters:	[in] pEncodeHandle	Audio encoding handle, the return value of NET_DVR_InitG722Encoder
Return:	None. Please call NET_DVR_GetLastError to get the error code.	
Remarks:		

Return to index

5.20.14 Initialize the audio decoding resource:

NET_DVR_InitG722Decoder

API:	void* NET_DVR_InitG722Decoder(int nBitrate = 16000)	
Parameters:	[in] nBitrate	The sample rate, it should be 16000
Return:	Return -1 if it is failed, and other return values could be used as handle of audio decoding. Please call NET_DVR_GetLastError to get the error code.	
Remarks:		

Return to index

5.20.15 Decode G722 audio to PCM: NET_DVR_DecodeG722Frame

API:	BOOL NET_DVR_DecodeG722Frame(void *pDecHandle, unsigned char* pInBuffer, unsigned char* pOutBuffer)	
Parameters:	[in] pDecHandle	Audio decoding handle, the return value of NET_DVR_InitG722Decoder
	[in] pInBuffer	Input buffer which size is 80 bytes
	[out] pOutBuffer	Output buffer, the sample rate of PCM data is 16000, 16 bit, Mono, and the size of output data is 1280 bytes.

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It is used mainly for voice talking and voice forwarding.
When you want to transfer the original audio data from client to the device, please call NET_DVR_EncodeG722Frame to encode the data and then transfer the encoded data to the device. When the client get the encoded voice stream from device, you can call this API to decode the data. Before calling the encoding and decoding functions, it requires initial operation (NET_DVR_InitG722Encoder or NET_DVR_InitG722Decoder), and after calling them, please release the resource by calling NET_DVR_ReleaseG722Encoder or NET_DVR_ReleaseG722Decoder.

Return to index

5.20.16 Release the audio decoding resource:

NET_DVR_ReleaseG722Decoder

API: void NET_DVR_ReleaseG722Decoder(void *pDecHandle)

Parameters:

[in] pDecHandle	Audio decoding handle, the return value of NET_DVR_InitG722Decoder

Return: None. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

Encode or decode the G711 audio

5.20.17 Encode the PCM audio to G711 format:

NET_DVR_EncodeG711Frame

API: BOOL NET_DVR_EncodeG711Frame(unsigned int iType, unsigned char *pInBuffer, unsigned char *pOutBuffer)

Parameters:

[in] iType	Encoding type: 0- Mu law, none 0- A law
[in] pInBuffer	Input buffer, PCM data is 8000 sample rate, 16 bit, Mono, and the size of input data should be 320 bytes
[out] pOutBuffer	Output buffer, the size of output encoded data is 160 bytes

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It is used mainly for voice talking and voice forwarding.
When you want to transfer the original audio data from client to the device, please call this API to encode the data and then transfer the encoded data to the device. When the client get the encoded voice stream from device, you can call NET_DVR_DecodeG711Frame to decode the data. Before calling the encoding and decoding functions, it doesn't require initial operation.

Return to index

5.20.18 Decode G711 audio to PCM: NET_DVR_DecodeG711Frame

API: BOOL NET_DVR_DecodeG711Frame(unsigned int iType, unsigned char *pInBuffer, unsigned char *pOutBuffer)

Parameters:

[in] iType	Encoding type: 0- Mu law, none 0- A law
[in] pInBuffer	Input buffer which size should be 160 bytes
[out] pOutBuffer	Output buffer. PCM data is 8000 sample rate, 16 bit, Mono, and the size of output data is 320 bytes.

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It is used mainly for voice talking and voice forwarding.
When you want to transfer the original audio data from client to the device, please call NET_DVR_EncodeG711Frame to encode the data and then transfer the encoded data to the device. When the client get the encoded voice stream from device, you can call this API to decode the data. Before calling the encoding and decoding functions, it doesn't require initial operation.

Return to index

Encode or decode the G726 audio

5.20.19 Initialize the audio encoding resource:

NET_DVR_InitG726Encoder

API: void* NET_DVR_InitG726Encoder(void **pEncMoudle)

Parameters:

[our] pEncMoudle	Encoding module handle, used as the input parameter when encoding the audio data

Return: Return -1 if it is failed, and the other is used as the handle of audio encoding. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.20.20 Encode the PCM audio to G726 format:

NET_DVR_EncodeG726Frame

API: BOOL NET_DVR_EncodeG726Frame(void *pEncMoudle, unsigned char *pInBuffer, unsigned char *pOutBuffer, BYTE byReset)

Parameters:

[in] pEncMoudle	Audio encoding handle, the output parameter value of NET_DVR_InitG726Encoder
[in] pInBuffer	Input buffer, PCM data is 8000 sample rate, 16 bit, Mono, and the size of input data should be 640 bytes
[out] pOutBuffer	Output buffer, the size of output encoded data is 80 bytes
[in] byReset	Whether to reset or not: 0- no, 1- yes, the first frame requires to be reset

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It is used mainly for voice talking and voice forwarding.
When you want to transfer the original audio data from client to the device, please call this API to encode the data and then transfer the encoded data to the device. When the client get the encoded voice stream from device, you can call NET_DVR_DecodeG726Frame to decode the data. Before calling the encoding and decoding functions, it requires initial operation (NET_DVR_InitG726Encoder or NET_DVR_InitG726Decoder), and after calling them, please release the resource by calling NET_DVR_ReleaseG726Encoder or NET_DVR_ReleaseG726Decoder.

Return to index

5.20.21 Release the audio encoding resource:

NET_DVR_ReleaseG726Encoder

API: void NET_DVR_ReleaseG726Encoder(void *pEncHandle)

Parameters:

[in] pEncHandle	Audio encoding handle, the return value of NET_DVR_InitG726Encoder

Return: None. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.20.22 Initialize the audio decoding resource:

NET_DVR_InitG726Decoder

API: void* NET_DVR_InitG726Decoder(void **pDecMoudle)

Parameters:

[out] pDecMoudle	Decoding module handle, used as the input parameter when decoding the audio data

Return: Return -1 if it is failed, and other return values could be used as handle of audio decoding. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.20.23 Decode G726 audio to PCM: NET_DVR_DecodeG726Frame

API: BOOL NET_DVR_DecodeG726Frame(void *pDecMoudle,unsigned char *pInBuffer, unsigned char *pOutBuffer, BYTE byReset)

Parameters:

[in] pDecMoudle	Audio decoding handle, the output parameter value of NET_DVR_InitG726Decoder
[in] pInBuffer	Input buffer which size is 80 bytes
[out] pOutBuffer	Output buffer, the sample rate of PCM data is 8000, 16 bit, Mono, and the size of output data is 640 bytes.
[in] byReset	Whether to reset or not: 0- no, 1- yes, the first frame requires to be reset

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: It is used mainly for voice talking and voice forwarding.

When you want to transfer the original audio data from client to the device, please call NET_DVR_EncodeG726Frame to encode the data and then transfer the encoded data to the device. When the client get the encoded voice stream from device, you can call this API to decode the data. Before calling the encoding and decoding functions, it requires initial operation (NET_DVR_InitG726Encoder or NET_DVR_InitG726Decoder), and after calling them, please release the resource by calling NET_DVR_ReleaseG726Encoder or NET_DVR_ReleaseG726Decoder.

Return to index

5.20.24 Release the audio decoding resource:

NET_DVR_ReleaseG726Decoder

API:	void NET_DVR_ReleaseG726Decoder(void *pDecHandle)	
Parameters:	[in] pDecHandle	Audio decoding handle, the return value of NET_DVR_InitG726Decoder
Return:	None. Please call NET_DVR_GetLastError to get the error code.	
Remarks:		

Return to index

5.21 Transparent Channel

5.21.1 Setup the transparent channel: NET_DVR_SerialStart

API:	LONG NET_DVR_SerialStart(LONG lUserID,LONG lSerialPort, fSerialDataCallBack cbSerialDataCallBack,DWORD dwUser)	
Parameters:	[in] lUserID	The return value of NET_DVR_Login_V30
	[in] lSerialPort	Serial port number: 1- 232 port, 2- 485 port
	[in] fSerialDataCallBack	Callback function, used to receive the data form the device's serial port.
	[in] dwUser	User data
	typedef void(CALLBACK *fSerialDataCallBack)(LONG lSerialHandle,char *pRecvDataBuffer, DWORD dwBufSize, DWORD dwUser)	
	[out] lSerialHandle	The serial handle, the return value of NET_DVR_SerialStart
	[out] pRecvDataBuffer	Pointer of the buffer to save data
	[out] dwBufSize	The size of data buffer
	[out] dwUser	User data
Return:	Return -1 if it is failed, and other values are as handle parameter of APIs like NET_DVR_SerialSend. Please call NET_DVR_GetLastError to get the error code.	
Remarks:	The decoder that connects with the serial port should support data postback, otherwise the callback will not get the data.	

Return to index

5.21.2 Send data to the serial port of the device by transparent channel: NET_DVR_SerialSend

API: BOOL NET_DVR_SerialSend(LONG lSerialHandle, LONG lChannel, char *pSendBuf,DWORD dwBufSize)

Parameters:

[in] lSerialHandle	The serial handle, the return value of NET_DVR_SerialStart.
[in] lChannel	Valid when using 485 serial port, begin with 1, set value to 0 when using RS232.
[in] pSendBuf	Buffer pointer of the data to be sent.
[in] dwBufSize	The size of data buffer, max 1016 bytes.

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.21.3 Close the transparent channel: NET_DVR_SerialStop

API: BOOL NET_DVR_SerialStop (LONG lSerialHandle)

Parameters:

[in] lSerialHandle	The return value of NET_DVR_SerialStart

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.22 Send data to the serial port directly

5.22.1 Send data to the serial port directly, and it dosn not require to setup transparent channel: NET_DVR_SendToSerialPort

API: BOOL NET_DVR_SendToSerialPort(LONG lUserID, DWORD dwSerialPort, DWORD dwSerialIndex, char *pSendBuf, DWORD dwBufSize)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] dwSerialPort	Serial port type: 1- 232, 2- 485
[in] dwSerialIndex	Means the number of 232 or 485, starting from 1
[in] pSendBuf	Pointer of the buffer to save the data
[in] dwBufSize	Buffer size, max 1016 bytes

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.22.2 Send data to RS232 directly and it doesn't require to setup transparent channel: NET_DVR_SendTo232Port

API: BOOL NET_DVR_SendTo232Port(LONG lUserID, char *pSendBuf, DWORD dwBufSize)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] pSendBuf	Pointer of the buffer to save the data
[in] dwBufSize	Buffer size, max 1016 bytes

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.23 Hard Disk Management

5.23.1 Remotely format hard disk of the device: NET_DVR_FormatDisk

API: LONG NET_DVR_FormatDisk(LONG lUserID,LONG lDiskNumber)

Parameters:

[in] lUserID	The return value of NET_DVR_Login_V30
[in] lDiskNumber	Hard disk number, begins from 0, and 0xff means all disk(don't include read-only disk)

Return: Return -1 if it is failed, and other values could be used as a parameter of NET_DVR_CloseFormatHandle. Please call NET_DVR_GetLastError to get the error code.

Remarks: If network breaks down during formatting, the device will continue to format, but the client can't receive the state.

Return to index

5.23.2 Get the format progress: NET_DVR_GetFormatProgress

API: BOOL NET_DVR_GetFormatProgress(LONG lFormatHandle, LONG *pCurrentFormatDisk,LONG *pCurrentDiskPos,LONG *pFormatStatic)

Parameters:

[in] lFormatHandle	Handle of formatting, the return value of

	NET_DVR_FormatDisk
[out] pCurrentFormatDisk	The pointer of the hard disk number which is formatted currently, the hard disk number starts from 0, and -1 is the initial state
[out] pCurrentDiskPos	The pointer of formatting progress of current hard disk, and the progress value range: 0~100
[out] pFormatStatic	The pointer of hard disk formatting state: *0- it is being formatted* *1- the formatting of hard disk has finished* *2- there is exception when formatting, and the progress is stopped. It will appear in both local and network disk* *3- exception in network that leads to to the loss of network disk, and it will not able to start formatting*

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.23.3 Close the formatting handle, and release the resource:

NET_DVR_CloseFormatHandle

API:	BOOL NET_DVR_CloseFormatHandle(LONG lFormatHandle)	
Parameters:	[in] lFormatHandle	The formatting handle, the return value of NET_DVR_FormatDisk
Return:	Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.	
Remarks:		

Return to index

5.24 Device Maintenance Management

Get device work state

5.24.1 Get work state of the device: NET_DVR_GetDVRWorkState_V30

API: BOOL NET_DVR_GetDVRWorkState_V30(LONG lUserID,

LPNET_DVR_WORKSTATE_V30 lpWorkState)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

[out] lpWorkState Pointer to the structure of work state

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API is used to get device state, including the state of channel, alarm input, alarm output, voice channel, etc.

Return to index

Remote upgrade

5.24.2 Set the networt environment of remote upgrade:

NET_DVR_SetNetworkEnvironment

API: BOOL NET_DVR_SetNetworkEnvironment(DWORD dwEnvironmentLevel)

Parameters: [in] dwEnvironmentLevel Network environment level:

```
enum{
    LOCAL_AREA_NETWORK = 0,//LAN
    WIDE_AREA_NETWORK      //WAN
}
```

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: There're two network environment levels:
LOCAL_AREA_NETWORK means local area network environment (fine network, and smooth communication),
WIDE_AREA_NETWORK means wide area network environment (poor network, and communication easy to be blocked).
Before calling NET_DVR_Upgrade to upgrade the device, please call this API to adjust the different upgrading environment.

Return to index

5.24.3 Remote upgrade: NET_DVR_Upgrade

API: LONG NET_DVR_Upgrade(LONG lUserID, char *sFileName)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

[in] sFileName Upgrade file path (including the file name). The path length is related to the OS, and SDK has no limit for it. For Windows system, the default length is less than or equal to 256

bytes(including the file name).

Return: Return -1 if it is failed, and the other value is used to be parameter of NET_DVR_GetUpgradeState. Please call NET_DVR_GetLastError to get the error code.

Remarks: This API is used to upgrade the device remotely

Return to index

5.24.4 Get the progress of the remote upgrade:

NET_DVR_GetUpgradeProgress

API: Int NET_DVR_GetUpgradeProgress(LONG lUpgradeHandle)

Parameters: [in] lUpgradeHandle The return value of NET_DVR_Upgrade

Return: Return -1 if it is failed. 0 ~100 means the progress of upgrade. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.24.5 Get the state of the remote upgrade:

NET_DVR_GetUpgradeState

API: Int NET_DVR_GetUpgradeState(LONG lUpgradeHandle)

Parameters: [in] lUpgradeHandle The return value of NET_DVR_Upgrade

Return: -1- the calling of the API is failed
1 - the upgrade has been successful
2 - it is being upgrading
3 - the upgrade is failed
4 - network has disconnected, and the state is unknown
5 - language version not match
Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.24.6 Get the step information of the remote upgrade:

NET_DVR_GetUpgradeStep

API: LONG NET_DVR_GetUpgradeStep(LONG lUpgradeHandle, LONG

*pSubProgress)

Parameters: [in] lUpgradeHandle The return value of NET_DVR_Upgrade
[in] pSubProgress Step sub progress of the upgrade

Return: Return -1 if it is failed. Other value is defined as below:

Macro Definition	Value	Implication
STEP_RECV_DATA	1	Receive the upgrade package data
STEP_UPGRADE	2	Upgrade the device system
STEP_BACKUP	3	Backup the device system
STEP_SEARCH	255	The devcie is being searching upgrade file

Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.24.7 Close the upgrade handle, and release the resource: NET_DVR_CloseUpgradeHandle

API: BOOL NET_DVR_CloseUpgradeHandle(LONG lUpgradeHandle)
Parameters: [in] lUpgradeHandle The return value of NET_DVR_Upgrade
Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.
Remarks:

Return to index

Log Query

5.24.8 Query the log information of the device (supports to search log with S.M.A.R.T information): NET_DVR_FindDVRLog_V30

API: LONG NET_DVR_FindDVRLog_V30(LONG lUserID, LONG lSelectMode, DWORD dwMajorType,DWORD dwMinorType, LPNET_DVR_TIME lpStartTime, LPNET_DVR_TIME lpStopTime, BOOL bOnlySmart = FALSE)
Parameters: [in] lUserID The return value of NET_DVR_Login_V30
[in] lSelectMode Query mode: 0- all, 1- by type, 2- by time, 3- by time and type
[in] dwMajorType Major type (it is invalid when search in S.M.A.R.T), 0 means all types, and the other

	types are listed below: **dwMajorType List**.
[in] dwMinorType	Minor type (it is invalid when search in S.M.A.R.T), 0 means all types, and the other types based on major types are listed below: **dwMinorType List**
[in] lpStartTime	Starting time
[in] lpStopTime	End time
[in] bOnlySmart	Whether to search log with S.M.A.R.T information only

dwMajorType List:

Macro Definition	Value	Implication
MAJOR_ALARM	0x1	Alarm
MAJOR_EXCEPTION	0x2	Exception
MAJOR_OPERATION	0x3	Operation
MAJOR_INFORMATION	0x4	Additional information of log

dwMinorType List:

Macro Definition of Major Type	Value	Implication
MAJOR_ALARM	0x1	Alarm
Macro Definition of Minor Type	**Value**	**Implication**
MINOR_ALARM_IN	0x1	Input of alarm
MINOR_ALARM_OUT	0x2	Output of alarm
MINOR_MOTDET_START	0x3	Start motion detection alarm
MINOR_MOTDET_STOP	0x4	Stop motion detection alarm
MINOR_HIDE_ALARM_START	0x5	Start tampering alarm
MINOR_HIDE_ALARM_STOP	0x6	Stop tampering alarm
MINOR_VCA_ALARM_START	0x7	Start intelligent alarm
MINOR_VCA_ALARM_STOP	0x8	Stop intelligent alarm
MINOR_ITS_ALARM_START	0x9	Start intelligent traffic alarm
MINOR_ITS_ALARM_STOP	0xa	Stop intelligent traffic alarm
MINOR_NETALARM_START	0xb	Start network alarm resume
MINOR_NETALARM_STOP	0xc	Stop network alarm resume
MINOR_NETALARM_RESUME	0xd	Network alarm resume

Macro Definition of Major Type	Value	Implication
MAJOR_EXCEPTION	0x2	Exception
Macro Definition of Minor Type	**Value**	**Implication**

MINOR_RAID_ERROR	0x20	RAID exception
MINOR_VI_LOST	0x21	Lose video signal
MINOR_ILLEGAL_ACCESS	0x22	Illegal access
MINOR_HD_FULL	0x23	Hard disk full
MINOR_HD_ERROR	0x24	Hard disk error
MINOR_DCD_LOST	0x25	MODEM off-line(reserved)
MINOR_IP_CONFLICT	0x26	IP conflict
MINOR_NET_BROKEN	0x27	Network not connected
MINOR_REC_ERROR	0x28	Recoding error
MINOR_IPC_NO_LINK	0x29	IPC connection failed
MINOR_VI_EXCEPTION	0x2a	Exception of video input (only for analog channels)
MINOR_IPC_IP_CONFLICT	0x2b	IP conflict of IPC
MINOR_SENCE_EXCEPTION	0x2c	Sence exception
MINOR_PIC_REC_ERROR	0x2d	Failed to get picture file, capture error
MINOR_VI_MISMATCH	0x2e	Video format mismatch
MINOR_RESOLUTION_MISMATCH	0x2f	Encoding resolution is not matching with the front-end resolution
MINOR_SCREEN_SUBSYSTEM_ABNORMALREBOOT	0x3c	Sub-board abnormal startup
MINOR_SCREEN_SUBSYSTEM_ABNORMALINSERT	0x3d	Sub-board inserted
MINOR_SCREEN_SUBSYSTEM_ABNORMALPULLOUT	0x3e	Sub-board pulled out
MINOR_SCREEN_ABNARMALTEMPERATURE	0x3f	Temperature abnormal

Macro Definition of Major Type	**Value**	**Implication**
MAJOR_OPERATION	0x3	Operation
Macro Definition of Minor Type	**Value**	**Implication**
MINOR_START_DVR	0x41	Start DVR
MINOR_STOP_DVR	0x42	Close DVR
MINOR_STOP_ABNORMAL	0x43	Stop abnormal
MINOR_REBOOT_DVR	0x44	reboot DVR (local)
MINOR_LOCAL_LOGIN	0x50	Login (local)
MINOR_LOCAL_LOGOUT	0x51	Logout (local)

MINOR_LOCAL_CFG_PARM	0x52	Local configuration
MINOR_LOCAL_PLAYBYFILE	0x53	Playback or download (local)
MINOR_LOCAL_PLAYBYTIME	0x54	Playback or download by time (local)
MINOR_LOCAL_START_REC	0x55	start recoding (local)
MINOR_LOCAL_STOP_REC	0x56	Stop recoding (local)
MINOR_LOCAL_PTZCTRL	0x57	Local PTZ control
MINOR_LOCAL_PREVIEW	0x58	Local preview(reserved)
MINOR_LOCAL_MODIFY_TIME	0x59	Modify time (local, reserved)
MINOR_LOCAL_UPGRADE	0x5a	Upgrade (local)
MINOR_LOCAL_RECFILE_OUTPUT	0x5b	Backup (local)
MINOR_LOCAL_FORMAT_HDD	0x5c	HD format (local)
MINOR_LOCAL_CFGFILE_OUTPUT	0x5d	Export configuration (local)
MINOR_LOCAL_CFGFILE_INPUT	0x5e	Import configuration (local)
MINOR_LOCAL_COPYFILE	0x5f	Backup file (local)
MINOR_LOCAL_LOCKFILE	0x60	Lockup file (local)
MINOR_LOCAL_UNLOCKFILE	0x61	Unlock file (local)
MINOR_LOCAL_DVR_ALARM	0x62	Clear/Trigger alarm (local)
MINOR_IPC_ADD	0x63	Add IPC (local)
MINOR_IPC_DEL	0x64	Delete IPC (local)
MINOR_IPC_SET	0x65	Set IPC (local)
MINOR_LOCAL_START_BACKUP	0x66	Start local backup
MINOR_LOCAL_STOP_BACKUP	0x67	Stop local backup
MINOR_LOCAL_COPYFILE_START_TIME	0x68	Start time of local backup
MINOR_LOCAL_COPYFILE_END_TIME	0x69	End time of local backup
MINOR_LOCAL_ADD_NAS	0x6a	Add network disk locally
MINOR_LOCAL_DEL_NAS	0x6b	Delete network disk locally
MINOR_LOCAL_SET_NAS	0x6c	Set NAS locally
MINOR_REMOTE_LOGIN	0x70	Login (remote)
MINOR_REMOTE_LOGOUT	0x71	Logout (remote)
MINOR_REMOTE_START_REC	0x72	Start record (remote)
MINOR_REMOTE_STOP_REC	0x73	Stop record (remote)
MINOR_START_TRANS_CHAN	0x74	Start transparent channel
MINOR_STOP_TRANS_CHAN	0x75	Stop transparent channel
MINOR_REMOTE_GET_PARM	0x76	Get parameter remotely
MINOR_REMOTE_CFG_PARM	0x77	Remote configuration

MINOR_REMOTE_GET_STATUS	0x78	Get status remotely
MINOR_REMOTE_ARM	0x79	On guard (remote)
MINOR_REMOTE_DISARM	0x7a	Disarm remotely
MINOR_REMOTE_REBOOT	0x7b	Reboot remotely
MINOR_START_VT	0x7c	Start voice talk
MINOR_STOP_VT	0x7d	Stop voice talk
MINOR_REMOTE_UPGRADE	0x7e	Upgrade remotely
MINOR_REMOTE_PLAYBYFILE	0x7f	Playback by file name remotely
MINOR_REMOTE_PLAYBYTIME	0x80	Playback by time remotely
MINOR_REMOTE_PTZCTRL	0x81	Remote PTZ control
MINOR_REMOTE_FORMAT_HDD	0x82	Format hard disk remotely
MINOR_REMOTE_STOP	0x83	Shut down remotely
MINOR_REMOTE_LOCKFILE	0x84	Lockup file remotely
MINOR_REMOTE_UNLOCKFILE	0x85	Unlock file remotely
MINOR_REMOTE_CFGFILE_OUTPUT	0x86	Export configuration remotely
MINOR_REMOTE_CFGFILE_INTPUT	0x87	Import configuration remotely
MINOR_REMOTE_RECFILE_OUTPUT	0x88	Backup recording files remotely
MINOR_REMOTE_DVR_ALARM	0x89	Trigger/clear alarm remotely
MINOR_REMOTE_IPC_ADD	0x8a	Add IPC remotely
MINOR_REMOTE_IPC_DEL	0x8b	Delete IPC remotely
MINOR_REMOTE_IPC_SET	0x8c	Set IPC remotely
MINOR_REBOOT_VCA_LIB	0x8d	Restart VCA library
MINOR_REMOTE_ADD_NAS	0x8e	Add NAS remotely
MINOR_REMOTE_DEL_NAS	0x8f	Delete NAS remotely
MINOR_REMOTE_SET_NAS	0x90	Set NAS remotely
MINOR_LOCAL_CONF_REB_RAID	0x101	Rebuild local configuraion automatically
MINOR_LOCAL_CONF_SPARE	0x102	Local configuration spare
MINOR_LOCAL_ADD_RAID	0x103	Create RAID locally
MINOR_LOCAL_DEL_RAID	0x104	Delete RAID localy
MINOR_LOCAL_MIG_RAID	0x105	Migrate RAID locally
MINOR_LOCAL_REB_RAID	0x106	Rebuild RAID manually and locally
MINOR_LOCAL_QUICK_CONF_RAID	0x107	Local one-key configuration
MINOR_LOCAL_ADD_VD	0x108	Create virtual disk locally
MINOR_LOCAL_DEL_VD	0x109	Delete virtual disk locally

MINOR_LOCAL_RP_VD	0x10a	Repair virtual disk locally
MINOR_LOCAL_FORMAT_EXPANDVD	0x10b	Expand virtual disk locally
MINOR_LOCAL_RAID_UPGRADE	0x10c	Local RAID card upgrade
MINOR_LOCAL_STOP_RAID	0x10d	Stop RAID operation(pull out disk safely) locally
MINOR_REMOTE_CONF_REB_RAID	0x111	Remotely configure auto rebuilding
MINOR_REMOTE_CONF_SPARE	0x112	Remotely configure spare
MINOR_REMOTE_ADD_RAID	0x113	Create RAID remotely
MINOR_REMOTE_DEL_RAID	0x114	Delete RAID remotely
MINOR_REMOTE_MIG_RAID	0x115	Migrate RAID remotely
MINOR_REMOTE_REB_RAID	0x116	Rebuild RAID manually and remotely
MINOR_REMOTE_QUICK_CONF_RAID	0x117	remote one-key configuration
MINOR_REMOTE_ADD_VD	0x118	Create virtual disk remotely
MINOR_REMOTE_DEL_VD	0x119	Delete virtual disk remotely
MINOR_REMOTE_RP_VD	0x11a	Repair virtual disk remotely
MINOR_REMOTE_FORMAT_EXPANDVD	0x11b	Expand virtual disk remotely
MINOR_REMOTE_RAID_UPGRADE	0x11c	Remote RAID card upgrade
MINOR_REMOTE_STOP_RAID	0x11d	Stop RAID operation(pull out disk safely) remotely
MINOR_LOCAL_START_PIC_REC	0x121	Start capturing picture locally
MINOR_LOCAL_STOP_PIC_REC	0x122	Stop capturing picture locally
MINOR_LOCAL_SET_SNMP	0x125	Configure SNMP locally
MINOR_LOCAL_TAG_OPT	0x126	Local label operation
MINOR_REMOTE_START_PIC_REC	0x131	Start capturing picture remotely
MINOR_REMOTE_STOP_PIC_REC	0x132	Stop capturing picture remotely
MINOR_REMOTE_SET_SNMP	0x135	Remote SNMP configuration
MINOR_REMOTE_TAG_OPT	0x136	Remote label operation

Macro Definition of Major Type	**Value**	**Implication**
MAJOR_INFORMATION	0x4	Additional information
Macro Definition of Minor Type	**Value**	**Implication**
MINOR_HDD_INFO	0xa1	HD information
MINOR_SMART_INFO	0xa2	S.M.A.R.T information
MINOR_REC_START	0xa3	Start recording
MINOR_REC_STOP	0xa4	Stop recording
MINOR_REC_OVERDUE	0xa5	Record overdue

MINOR_LINK_START	0xa6	Connect to front-end device
MINOR_LINK_STOP	0xa7	Disconnect front-end device
MINOR_NET_DISK_INFO	0xa8	Network disk information
MINOR_RAID_INFO	0xa9	RAID information
MINOR_LINK_START	0xb3	Start capturing picture
MINOR_PIC_REC_STOP	0xb4	Stop capturing picture
MINOR_PIC_REC_OVERDUE	0xb5	Delete expired picture

Return: Return -1 if it is failed, and the other values could be used as a parameter of NET_DVR_FindNextLog_V30. Please call NET_DVR_GetLastError to get the error code.

Remarks: For general devices, such as DS-81xx and DS-80xx series, it supports to query up to 2000 normal logs; for DS-90xx series(v2.0 or higher), supports up to 4000 logs; For DS-81xxHF-ST,it supports up to 1000 logs.If to query S.M.A.R.T logs, it supports max 500 logs at one time.
If S.M.A.R.T information is not needed, we can search all logs by setting bOnlySmart to FALSE.
S.M.A.R.T information: HD working record.

Return to index

5.24.9 Get the log one by one: NET_DVR_FindNextLog_V30

API: LONG NET_DVR_FindNextLog_V30(LONG lLogHandle, LPNET_DVR_LOG_V30 lpLogData)

Parameters: [in] lLogHandle Handle of file searching, return value of NET_DVR_FindDVRLog_V30
[out] lpLogData Pointer for saving the log information

Return: Return -1 if it is failed, and other values stand for current status or other information, details listed below. Please call NET_DVR_GetLastError to get the error code.

Macro Definition	Value	Implication
NET_DVR_FILE_SUCCESS	1000	Get the log information successfully
NET_DVR_FILE_NOFIND	1001	No log found
NET_DVR_ISFINDING	1002	Being searching, please wait
NET_DVR_NOMOREFILE	1003	No more log found, search is finished
NET_DVR_FILE_EXCEPTION	1004	Exception when search log

Remarks: Before calling this API, please call NET_DVR_FindDVRLog_V30 to get current searching handle firstly.

Return to index

5.24.10 Stop querying the log and release the resource:

NET_DVR_FindLogClose_V30

API: BOOL NET_DVR_FindLogClose_V30(LONG lLogHandle)

Parameters:

[in] lLogHandle	Handle of log query, the return value of NET_DVR_FindDVRLog_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

Remote backup

5.24.11 Backup record files, pictures, or log information:

NET_DVR_Backup

API: DWORD NET_DVR_Backup(long lUserID, DWORD dwBackupType,void* lpBackupBuff, DWORD dwBackupBuffSize)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[in] dwBackupType	Backup type: 1- backup record files by file name, 2- backup record files by time, 3- backup pictures, 4- backup the event that resume inquest, 5- backup log information
[in] lpBackupBuff	The backup paramter, related with dwBackupType, see to the list below
[in] dwBackupBuffSize	The size of backup paramter

Return: The size of backup paramter. Please call NET_DVR_GetLastError to get the error code.

Remarks: The relation between dwBackupType and lpBackupBuff is listed below:

dwBackupType	Implication	lpBackupBuff
1	Backup record files by file name	NET_DVR_BACKUP_NAME_PARAM
2	Backup record files by time	NET_DVR_BACKUP_TIME_PARAM
3	Backup pictures	NET_DVR_BACKUP_PICTURE_PARAM
5	Backup log information	NET_DVR_BACKUP_LOG_PARAM

Return to index

Restore device default configuration

5.24.12 Restore device default configuration: NET_DVR_RestoreConfig

API: BOOL NET_DVR_RestoreConfig(LONG lUserID)

Parameters: [in] lUserID User ID, the return value of NET_DVR_Login_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

Import or export configuration file

5.24.13 Export the configuration file from the device:

NET_DVR_GetConfigFile_V30

API: BOOL NET_DVR_GetConfigFile_V30(LONG lUserID, char *sOutBuffer, DWORD dwOutSize, DWORD *pReturnSize)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[out]sOutBuffer	The buffer to save configuration parameters
[in] dwOutSize	The buffer size
[out] pReturnSize	The size of the returned buffer

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: When sOutBuffer = NULL, dwOutSize = 0 and pReturnSize != NULL, it is used to get the required size of the buffer to save the configuration file.
When sOutBuffer != NULL and dwOutSize != 0, it is used to get the buffer content which is the configuration file.

Return to index

5.24.14 Export the configuration file from the device:

NET_DVR_GetConfigFile

API: BOOL NET_DVR_GetConfigFile(LONG lUserID, char *sFileName)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[in] sFileName	The file path to save the configuration file (binary file)

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError

to get the error code.

Remarks:

Return to index

5.24.15 Import the configuration file to the device:

NET_DVR_SetConfigFile_EX

API: BOOL NET_DVR_SetConfigFile_EX(LONG lUserID, char *sInBuffer, DWORD dwInSize)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[in] sInBuffer	The buffer that saves the configuration parameters
[in] dwInSize	The buffer size

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.24.16 Import the configuration file to the device:

NET_DVR_SetConfigFile

API: BOOL NET_DVR_SetConfigFile(LONG lUserID, char *sFileName)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[in] sFileName	The file path that saves the configuration file (binary file)

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.25 Shutdown and Reboot

5.25.1 Reboot the device: NET_DVR_RebootDVR

API: BOOL NET_DVR_RebootDVR(LONG lUserID)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.25.2 Shutdown the device: NET_DVR_ShutDownDVR

API: BOOL NET_DVR_ShutDownDVR(LONG lUserID)

Parameters: [in] lUserID User ID, the return value of NET_DVR_Login_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.26 Remote Parameter Configuration

General parameter configuration

5.26.1 Get configuration of the device: NET_DVR_GetDVRConfig

API: BOOL NET_DVR_GetDVRConfig(LONG lUserID, DWORD dwCommand,LONG lChannel, LPVOID lpOutBuffer, DWORD dwOutBufferSize, LPDWORD lpBytesReturned)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[in] dwCommand	Configuration command, please kindly refer to **the DwCommand type definition** below
[in] lChannel	Channel number, if the channel parameter is not required, lChannel is invalid, and set it as 0xFFFFFFFF
[out] lpOutBuffer	The buffer to save the received data
[in] dwOutBufferSize	The size of the buffer (unit: byte), it can't be 0
[out] lpBytesReturned	The size of the returned buffer, it can't be NULL

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: The structures and command numbers are different according to the various getting functions, and they are listed as below:

The relationship between dwCommand and lpOutBuffer

Macro Definition of dwCommand	Description	lChannel	lpOutBuffer	Value
NET_DVR_GET_TIMECFG	Get time parameters	invalid	NET_DVR_TIME	118
NET_DVR_GET_ZONEANDDST	Get time zone & DST parameters	invalid	NET_DVR_ZONEANDDST	128

NET_DVR_GET_NETAPPCFG	Get network application parameters(NTP/DDNS)	invalid	NET_DVR_NETAPPCFG	222
NET_DVR_GET_NTPCFG	Get network application parameters(NTP)	invalid	NET_DVR_NTPPARA	224
NET_DVR_GET_NFSCFG	Get NFS (Network File System) configuration	invalid	NET_DVR_NFSCFG	230
NET_DVR_GET_PTZPOS	Get PTZ parameters of IP speed dome	valid	NET_DVR_PTZPOS	293
NET_DVR_GET_PTZSCOPE	Get PTZ scope of IP speed dome	valid	NET_DVR_PTZSCOPE	294
NET_DVR_GET_AP_INFO_LIST	Get wireless network resource parameters	invalid	NET_DVR_AP_INFO_LIST	305
NET_DVR_GET_WIFI_CFG	Get wireless configuration of IP device	invalid	NET_DVR_WIFI_CFG	307
NET_DVR_GET_WIFI_WORKMODE	Get network adapter mode of IP device	invalid	NET_DVR_WIFI_WORKMODE	309
NET_DVR_GET_NETCFG_V30	Get network parameters	invalid	NET_DVR_NETCFG_V30	1000
NET_DVR_GET_PICCFG_V30	Get image parameters	valid	NET_DVR_PICCFG_V30	1002
NET_DVR_GET_RECORDCFG_V30	Get record parameters	valid	NET_DVR_RECORD_V30	1004
NET_DVR_GET_USERCFG_V30	Get user parameters	invalid	NET_DVR_USER_V30	1006
NET_DVR_GET_DDNSCFG_V30	Get network application parameters(DDNS)	invalid	NET_DVR_DDNSPARA_V30	1010
NET_DVR_GET_EMAILCFG_V30	Get network application parameters(EMAIL)	invalid	NET_DVR_EMAILCFG_V30	1012
NET_DVR_GET_CRUISE	Get PTZ cruise parameters	valid	NET_DVR_CRUISE_PARA	1020
NET_DVR_GET_ALARMINCFG_V30	Get alarm input parameters	valid	NET_DVR_ALARMINCFG_V30	1024
NET_DVR_GET_ALARMOUTCFG_V30	Get alarm output parameters	valid	NET_DVR_ALARMOUTCFG_V30	1026
NET_DVR_GET_SHOWSTRING_V30	Get OSD parameters	valid	NET_DVR_SHOWSTRING_V30	1030
NET_DVR_GET_EXCEPTIONCFG_V30	Get exception parameters	invalid	NET_DVR_EXCEPTION_V30	1034
NET_DVR_GET_RS232CFG_V30	Get 232 parameters	invalid	NET_DVR_RS232CFG_V30	1036
NET_DVR_GET_NET_DISKCFG	Get network disk configuration	invalid	NET_DVR_NET_DISKCFG	1038
NET_DVR_GET_COMPRESSCFG_V30	Get compression parameters	valid	NET_DVR_COMPRESSIONCFG_V30	1040

NET_DVR_GET_DECODERCFG_V30	Get (PTZ) decoder parameters	valid	NET_DVR_DECODERCFG_V30	1042
NET_DVR_GET_HDCFG	Get hard disk management parameters	invalid	NET_DVR_HDCFG	1054
NET_DVR_GET_COMPRESSCFG_AUD	Get audio parameters of voice talk	invalid	NET_DVR_COMPRESSION_AUDIO	1058
NET_DVR_GET_CCDPARAMCFG	Get front-end parameters	invalid	NET_DVR_CAMERAPARAMCFG	1067
NET_DVR_GET_DEVICECFG_V40	Get device parameters (extended)	invalid	NET_DVR_DEVICECFG_V40	1100
NET_DVR_GET_AUDIO_INPUT	Get audio input parameter	valid	NET_DVR_AUDIO_INPUT_PARAM	3201
NET_DVR_GET_CAMERA_DEHAZE_CFG	Get the de-haze parameter	valid	NET_DVR_CAMERA_DEHAZE_CFG	3203
NET_IPC_GET_AUX_ALARMCFG	Get aux alarm parameter	valid	NET_IPC_AUX_ALARMCFG	3209

Return to index

5.26.2 Set the parameters of the device: NET_DVR_SetDVRConfig

API: BOOL NET_DVR_SetDVRConfig(LONG lUserID, DWORD dwCommand,LONG lChannel, LPVOID lpInBuffer, DWORD dwInBufferSize)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[in] dwCommand	Parameter type. Please kindly refer to the **DwCommand Type Definition** below.
[in] lChannel	Channel number, if it is not the channel parameter, do not use lChannel, and set it as 0xFFFFFFFF
[in] lpInBuffer	Buffer that saves the output parameters
[in] dwInBufferSize	The buffer size (unit: byte)

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks: The structures and command numbers are different according to the various setting functions, and they are listed as below:

The relationship between dwCommand and lpInBuffer

Macro Definition of dwCommand	Description	lChannel	lpInBuffer	Value
NET_DVR_SET_TIMECFG	Set time parameters	invalid	NET_DVR_TIME	119
NET_DVR_SET_ZONEANDDST	Set time zone and DST	invalid	NET_DVR_ZONEANDDST	129

	parameters			
NET_DVR_SET_NETAPPCFG	Set network application parameters(NTP/DDNS)	invalid	NET_DVR_NETAPPCFG	223
NET_DVR_SET_NTPCFG	Set network application parameters(NTP)	invalid	NET_DVR_NTPPARA	225
NET_DVR_SET_NFSCFG	Set NFS (Network File System) parameters	invalid	NET_DVR_NFSCFG	231
NET_DVR_SET_PTZPOS	Set PTZ parameters of IP speed dome	valid	NET_DVR_PTZPOS	292
NET_DVR_SET_WIFI_CFG	Set wireless configuration of IP device	invalid	NET_DVR_WIFI_CFG	306
NET_DVR_SET_WIFI_WORKMODE	Set network adapter mode of IP device	invalid	NET_DVR_WIFI_WORKMODE	308
NET_DVR_SET_NETCFG_V30	Set network parameters	invalid	NET_DVR_NETCFG_V30	1001
NET_DVR_SET_PICCFG_V30	Set image parameters	valid	NET_DVR_PICCFG_V30	1003
NET_DVR_SET_RECORDCFG_V30	Set record parameters	valid	NET_DVR_RECORD_V30	1005
NET_DVR_SET_USERCFG_V30	Set user parameters	invalid	NET_DVR_USER_V30	1007
NET_DVR_SET_DDNSCFG_V30	Set network application parameters(DDNS)	invalid	NET_DVR_DDNSPARA_V30	1011
NET_DVR_SET_EMAILCFG_V30	Set network application parameters(EMAIL)	invalid	NET_DVR_EMAILCFG_V30	1013
NET_DVR_SET_CRUISE	Set cruise parameters	valid	NET_DVR_CRUISE_PARA	1021
NET_DVR_SET_ALARMINCFG_V30	Set alarm input parameters	valid	NET_DVR_ALARMINCFG_V30	1025
NET_DVR_SET_ALARMOUTCFG_V30	Set alarm output parameters	valid	NET_DVR_ALARMOUTCFG_V30	1027
NET_DVR_SET_SHOWSTRING_V30	Set OSD parameters	valid	NET_DVR_SHOWSTRING_V30	1031
NET_DVR_SET_EXCEPTIONCFG_V30	Set exception parameters	invalid	NET_DVR_EXCEPTION_V30	1035
NET_DVR_SET_RS232CFG_V30	Set 232 serial port parameters	invalid	NET_DVR_RS232CFG_V30	1037
NET_DVR_SET_NET_DISKCFG	Set network disk access parameters	invalid	NET_DVR_NET_DISKCFG	1039
NET_DVR_SET_COMPRESSCFG_V30	Set compression parameters	valid	NET_DVR_COMPRESSIONCFG_V30	1041
NET_DVR_SET_DECODERCFG_V30	Set PTZ decoder parameters	valid	NET_DVR_DECODERCFG_V30	1043

NET_DVR_SET_HDCFG	Set hard disk management parameters	invalid	NET_DVR_HDCFG	1055
NET_DVR_SET_COMPRESSCFG_AUD	Set audio parameters of voice talk	invalid	NET_DVR_COMPRESSION_AUDIO	1059
NET_DVR_SET_CCDPARAMCFG	Set front-end parameters	invalid	NET_DVR_CAMERAPARAMCFG	1068
NET_DVR_SET_DEVICECFG_V40	Set device parameters (extended)	invalid	NET_DVR_DEVICECFG_V40	1101
NET_DVR_SET_AUDIO_INPUT	Set audio input parameter	valid	NET_DVR_AUDIO_INPUT_PARAM	3202
NET_DVR_SET_CAMERA_DEHAZE_CFG	Set the de-haze parameter	valid	NET_DVR_CAMERA_DEHAZE_CFG	3204
NET_IPC_SET_AUX_ALARMCFG	Set aux alarm parameter	valid	NET_IPC_AUX_ALARMCFG	3210

Return to index

Alarm output configuration

5.26.3 Get the state of the alarm output: NET_DVR_GetAlarmOut_V30

API: BOOL NET_DVR_GetAlarmOut_V30(LONG lUserID, LPNET_DVR_ALARMOUTSTATUS_V30 lpAlarmOutState)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[out] lpAlarmOutState	The state of the alarm output

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.26.4 Set the alarm output port: NET_DVR_SetAlarmOut

API: BOOL NET_DVR_SetAlarmOut(LONG lUserID, LONG lAlarmOutPort,LONG lAlarmOutStatic)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[in] lAlarmOutPort	Alarm output port: The output port number begins with 0, 0x00ff means all analog output, 0xff00 means all IP output. DS-90xx devices support both analog and IP alarm output, and 32-95 are IP alarm ports.
[in] lAlarmOutStatic	The state of alarm output port: 0- stop output, 1

output

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

RTSP parameter configuration

5.26.5 Get the RTSP parameter: NET_DVR_GetRtspConfig

API: BOOL NET_DVR_GetRtspConfig(LONG lUserID, DWORD dwCommand, LPNET_DVR_RTSPCFG lpOutBuffer, DWORD dwOutBufferSize)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[in] dwCommand	Reserved, please set to 0
[out] lpOutBuffer	Output buffer
[in] dwOutBufferSize	The size of output buffer

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.26.6 Set the RTSP parameter: NET_DVR_SetRtspConfig

API: BOOL NET_DVR_SetRtspConfig(LONG lUserID, DWORD dwCommand, LPNET_DVR_RTSPCFG lpInBuffer, DWORD dwInBufferSize)

Parameters:

[in] lUserID	User ID, the return value of NET_DVR_Login_V30
[in] dwCommand	Reserved, please set to 0
[in] lpInBuffer	The buffer that saves the input parameters
[in] dwOutBufferSize	The size of the buffer, the value is the size of the structure NET_DVR_RTSPCFG

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

Scale parameters settings of video output

5.26.7 Get the scale information of the video output:

NET_DVR_GetScaleCFG_V30

API: BOOL NET_DVR_GetScaleCFG(LONG lUserID, LPNET_DVR_SCALECFG pScalecfg)

Parameters: [in] lUserID User ID, the return value of NET_DVR_Login_V30

[out] pScalecfg Scale parameter

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.26.8 Set the scale parameter of the video output:

NET_DVR_SetScaleCFG_V30

API: BOOL NET_DVR_SetScaleCFG_V30(LONG lUserID, LPNET_DVR_SCALECFG pScalecfg)

Parameters: [in] lUserID User ID, the return value of NET_DVR_Login_V30

[in] pScalecfg Scale parameter

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.27 E-mail test

5.27.1 Test according to the configured EMAIL parameter to see whether it can receive and send e-mail successfully:

NET_DVR_StartEmailTest

API: LONG NET_DVR_StartEmailTest(LONG lUserID)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

Return: -1 means false, other values are as parameters of

NET_DVR_GetEmailTestProcess and NET_DVR_StopEmailTest. Please call NET_DVR_GetLastError to get the error code.

Remarks: Before calling to this API to test, please configure the EMAIL parameter firstly, refer to NET_DVR_GetDVRConfig and NET_DVR_SetDVRConfig (command: NET_DVR_GET_EMAILCFG_V30 and NET_DVR_SET_EMAILCFG_V30).

Return to index

5.27.2 Get the progress of the e-mail test:

NET_DVR_GetEmailTestProgress

API: BOOL NET_DVR_GetEmailTestProgress(LONG lEmailTestHandle, DWORD* pState)

Parameters: [in] lEmailTestHandle The return value of NET_DVR_StartEmailTest

[out] pState E-mail test progress, range: (0,100), the other values out of this range is defined as below

Macro Definition	Value	Implication
PROCESSING	0	Being processing
PROCESS_SUCCESS	100	Test finished
PROCESS_EXCEPTION	400	Test abnormal
PROCESS_FAILED	500	Test failed

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Return to index

5.27.3 Stop E-mail test: NET_DVR_StopEmailTest

API: BOOL NET_DVR_StopEmailTest(LONG lEmailTestHandle)

Parameters: [in] lEmailTestHandle The return value of NET_DVR_StartEmailTest

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.28 Thermal network camera

5.28.1 Set manual shutter compensation:

NET_DVR_ShutterCompensation

API: BOOL NET_DVR_ShutterCompensation(LONG lUserID)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Remarks:

Return to index

5.28.2 Correct dead pixel: NET_DVR_CorrectDeadPixel

API: BOOL NET_DVR_CorrectDeadPixel(LONG lUserID, LONG lChannel, LPNET_DVR_CORRECT_DEADPIXEL_PARAM lpInParam)

Parameters: [in] lUserID The return value of NET_DVR_Login_V30

[in] lChannel Channel number

[in] lpInParam Dead pixel correction parameter

Return: Return TRUE on success, FALSE on failure. Please call NET_DVR_GetLastError to get the error code.

Return to index

6 Macro Definition of Error Code

6.1 Error code of network communication library

Error	Value	Message
NET_DVR_NOERROR	0	No error.
NET_DVR_PASSWORD_ERROR	1	User name or password error.
NET_DVR_NOENOUGHPRI	2	Not authorized to do this operation.
NET_DVR_NOINIT	3	SDK is not initialized.
NET_DVR_CHANNEL_ERROR	4	Channel number error. There is no corresponding channel number on the device.
NET_DVR_OVER_MAXLINK	5	The number of clients connected to the device has exceeded the max limit.
NET_DVR_VERSIONNOMATCH	6	Version mismatch. SDK version is not matching with the device.
NET_DVR_NETWORK_FAIL_CONNECT	7	Failed to connect to the device. The device is off-line, or connection timeout caused by network.
NET_DVR_NETWORK_SEND_ERROR	8	Failed to send data to the device.
NET_DVR_NETWORK_RECV_ERROR	9	Failed to receive data from the device.
NET_DVR_NETWORK_RECV_TIMEOUT	10	Timeout when receiving the data from the device.
NET_DVR_NETWORK_ERRORDATA	11	The data sent to the device is illegal, or the data received from the device error. E.g. The input data is not supported by the device for remote configuration.
NET_DVR_ORDER_ERROR	12	API calling order error.
NET_DVR_OPERNOPERMIT	13	Not authorized for this operation.
NET_DVR_COMMANDTIMEOUT	14	Executing command on the device is timeout.
NET_DVR_ERRORSERIALPORT	15	Serial port number error. The assigned serial port does not exist on the device.
NET_DVR_ERRORALARMPORT	16	Alarm port number error.
NET_DVR_PARAMETER_ERROR	17	Parameter error. Input or output parameter in the SDK API is NULL.
NET_DVR_CHAN_EXCEPTION	18	Device channel is in exception status.
NET_DVR_NODISK	19	No hard disk on the device, and the operation of recording and hard disk configuration will fail.
NET_DVR_ERRORDISKNUM	20	Hard disk number error. The assigned hard disk number does not exist during hard disk management.
NET_DVR_DISK_FULL	21	Device hark disk is full.
NET_DVR_DISK_ERROR	22	Device hard disk error.
NET_DVR_NOSUPPORT	23	Device does not support this function.
NET_DVR_BUSY	24	Device is busy.
NET_DVR_MODIFY_FAIL	25	Failed to modify device parameters.

NET_DVR_PASSWORD_FORMAT_ERROR	26	The inputting password format is not correct.
NET_DVR_DISK_FORMATING	27	Hard disk is formatting, and the operation cannot be done.
NET_DVR_DVRNORESOURCE	28	Not enough resource on the device.
NET_DVR_DVROPRATEFAILED	29	Device operation failed.
NET_DVR_OPENHOSTSOUND_FAIL	30	Failed to collect local audio data or to open audio output during voice talk / broadcasting.
NET_DVR_DVRVOICEOPENED	31	Voice talk channel on the device has been occupied.
NET_DVR_TIMEINPUTERROR	32	Time input is not correct.
NET_DVR_NOSPECFILE	33	There is no selected file for playback.
NET_DVR_CREATEFILE_ERROR	34	Failed to create a file, during local recording, saving picture, getting configuration file or downloading record file.
NET_DVR_FILEOPENFAIL	35	Failed to open a file, when importing configuration file, upgrading device or uploading inquest file.
NET_DVR_OPERNOTFINISH	36	The last operation has not been completed.
NET_DVR_GETPLAYTIMEFAIL	37	Failed to get the current played time.
NET_DVR_PLAYFAIL	38	Failed to start playback.
NET_DVR_FILEFORMAT_ERROR	39	The file format is not correct.
NET_DVR_DIR_ERROR	40	File directory error.
NET_DVR_ALLOC_RESOURCE_ERROR	41	Resource allocation error.
NET_DVR_AUDIO_MODE_ERROR	42	Sound adapter mode error. Currently opened sound playing mode does not match with the set mode.
NET_DVR_NOENOUGH_BUF	43	Buffer is not enough.
NET_DVR_CREATESOCKET_ERROR	44	Create SOCKET error.
NET_DVR_SETSOCKET_ERROR	45	Set SOCKET error.
NET_DVR_MAX_NUM	46	The number of login or preview connections has exceeded the SDK limitation.
NET_DVR_USERNOTEXIST	47	User doest not exist. The user ID has been logged out or unavailable.
NET_DVR_WRITEFLASHERROR	48	Writing FLASH error. Failed to write FLASH during device upgrade.
NET_DVR_UPGRADEFAIL	49	Failed to upgrade device. It is caused by network problem or the language mismatch between the device and the upgrade file.
NET_DVR_CARDHAVEINIT	50	The decode card has already been initialed.
NET_DVR_PLAYERFAILED	51	Failed to call API of player SDK.
NET_DVR_MAX_USERNUM	52	The number of login user has reached the maximum limit.
NET_DVR_GETLOCALIPANDMACFAIL	53	Failed to get the IP address or physical address of local PC.
NET_DVR_NOENCODEING	54	This channel hasn't started encoding.
NET_DVR_IPMISMATCH	55	IP address not match.
NET_DVR_MACMISMATCH	56	MAC address not match.
NET_DVR_UPGRADELANGMISMATCH	57	The language of upgrading file does not match the language of the device.

NET_DVR_MAX_PLAYERPORT	58	The number of player ports has reached the maximum limit.
NET_DVR_NOSPACEBACKUP	59	No enough space to backup file in backup device.
NET_DVR_NODEVICEBACKUP	60	No backup device.
NET_DVR_PICTURE_BITS_ERROR	61	The color quality setting of the picture does not match the requirement, and it should be limited to 24.
NET_DVR_PICTURE_DIMENSION_ERROR	62	The dimension is over 128x256.
NET_DVR_PICTURE_SIZ_ERROR	63	The size of picture is over 100K.
NET_DVR_LOADPLAYERSDKFAILED	64	Failed to load the player SDK.
NET_DVR_LOADPLAYERSDKPROC_ERROR	65	Can not find the function in player SDK.
NET_DVR_LOADDSSDKFAILED	66	Failed to load the library file-"DsSdk".
NET_DVR_LOADDSSDKPROC_ERROR	67	Can not find the API in "DsSdk".
NET_DVR_DSSDK_ERROR	68	Failed to call the API in "DsSdk".
NET_DVR_VOICEMONOPOLIZE	69	Sound adapter has been monopolized.
NET_DVR_JOINMULTICASTFAILED	70	Failed to join to multicast group.
NET_DVR_CREATEDIR_ERROR	71	Failed to create log file directory.
NET_DVR_BINDSOCKET_ERROR	72	Failed to bind socket.
NET_DVR_SOCKETCLOSE_ERROR	73	Socket disconnected. It is caused by network disconnection or destination unreachable.
NET_DVR_USERID_ISUSING	74	The user ID is operating when logout.
NET_DVR_SOCKETLISTEN_ERROR	75	Failed to listen.
NET_DVR_PROGRAM_EXCEPTION	76	SDK program exception.
NET_DVR_WRITEFILE_FAILED	77	Failed to write file, during local recording, saving picture or downloading record file.
NET_DVR_FORMAT_READONLY	78	Failed to format read-only HD.
NET_DVR_WITHSAMEUSERNAME	79	This user name already exists in the user configuration structure.
NET_DVR_DEVICETYPE_ERROR	80	Device type does not match when import configuration.
NET_DVR_LANGUAGE_ERROR	81	Language does not match when import configuration.
NET_DVR_PARAVERSION_ERROR	82	Software version does not match when import configuration.
NET_DVR_IPCHAN_NOTALIVE	83	IP channel is not on-line when previewing.
NET_DVR_RTSP_SDK_ERROR	84	Load StreamTransClient.dll failed.
NET_DVR_CONVERT_SDK_ERROR	85	Load SystemTransform.dll failed.
NET_DVR_IPC_COUNT_OVERFLOW	86	Exceeds maximum number of connected IP channels.
NET_DVR_MAX_ADD_NUM	87	Exceeds maximum number of supported record labels or other operations.
NET_DVR_PARAMMODE_ERROR	88	Image intensifier, parameter mode error. This error may occur when client sets software or hardware parameters.
NET_DVR_CODESPITTER_OFFLINE	89	Code splitter is offline.
NET_DVR_BACKUP_COPYING	90	Device is backing up.
NET_DVR_CHAN_NOTSUPPORT	91	Channel not support.
NET_DVR_CALLINEINVALID	92	The height line location is too concentrated, or the length line is not inclined enough.

NET_DVR_CALCANCELCONFLICT	93	Cancel calibration conflict, if the rule and overall actual size filter have been set.
NET_DVR_CALPOINTOUTRANGE	94	Calibration point exceeds the range.
NET_DVR_FILTERRECTINVALID	95	The size filter does not meet the requirement.
NET_DVR_DDNS_DEVOFFLINE	96	Device has not registered to DDNS.
NET_DVR_DDNS_INTER_ERROR	97	DDNS inner error.
NET_DVR_ALIAS_DUPLICATE	150	Alias is duplicate (for EasyDDNS)
NET_DVR_DEV_NET_OVERFLOW	800	Network traffic is over device ability limit.
NET_DVR_STATUS_RECORDFILE_WRITING_NOT_LOCK	801	The video file is recording and can't be locked.
NET_DVR_STATUS_CANT_FORMAT_LITTLE_DISK	802	The hard disk capacity is too small and can not be formatted.
Error code of RAID		
NET_DVR_NAME_NOT_ONLY	200	This user name already exists.
NET_DVR_OVER_MAX_ARRAY	201	The array exceeds the limitation.
NET_DVR_OVER_MAX_VD	202	The virtual disk exceeds the limitation.
NET_DVR_VD_SLOT_EXCEED	203	The virtual disk slots are full.
NET_DVR_PD_STATUS_INVALID	204	Physical disk used to rebuild RAID is in error state.
NET_DVR_PD_BE_DEDICATE_SPARE	205	Physical disk used to rebuild RAID is assigned as spare disk.
NET_DVR_PD_NOT_FREE	206	Physical disk used to rebuild RAID is not free.
NET_DVR_CANNOT_MIG2NEWMODE	207	Can not migrate from current RAID type to the new type.
NET_DVR_MIG_PAUSE	208	Migration has been paused.
NET_DVR_MIG_ABOUTED	209	Migration has been aborted.
NET_DVR_EXIST_VD	210	There is virtual disk in the array, and the array can not been deleted.
NET_DVR_TARGET_IN_LD_FUNCTIONAL	211	Target physical disk is part of the virtual disk and is functional.
NET_DVR_HD_IS_ASSIGNED_ALREADY	212	Specified physical disk is assigned as a virtual disk.
NET_DVR_INVALID_HD_COUNT	213	Number of physical disks doesn't fit the specified RAID level.
NET_DVR_LD_IS_FUNCTIONAL	214	Specified virtual disk is functional and it can not be rebuilt.
NET_DVR_BGA_RUNNING	215	BGA is running.
NET_DVR_LD_NO_ATAPI	216	Can not create virtual disk with ATAPI drive.
NET_DVR_MIGRATION_NOT_NEED	217	Migration is not necessary.
NET_DVR_HD_TYPE_MISMATCH	218	Physical disks are not of the same type.
NET_DVR_NO_LD_IN_DG	219	No virtual disk exists on the specified array.
NET_DVR_NO_ROOM_FOR_SPARE	220	Disk space is too small to be assigned as spare drive.
NET_DVR_SPARE_IS_IN_MULTI_DG	221	Disk is already assigned as a spare drive for an array.
NET_DVR_DG_HAS_MISSING_PD	222	Disk is missing from an array.

Error code of intelligent device		
NET_DVR_ID_ERROR	300	Configuration ID is illegal.
NET_DVR_POLYGON_ERROR	301	Polygon does not match requirement.
NET_DVR_RULE_PARAM_ERROR	302	Rule parameter is illegal.
NET_DVR_RULE_CFG_CONFLICT	303	Configuration conflict.
NET_DVR_CALIBRATE_NOT_READY	304	Calibration not ready.
NET_DVR_CAMERA_DATA_ERROR	305	Camera parameter is illegal.
NET_DVR_CALIBRATE_DATA_UNFIT	306	Not inclined enough, not fit to calibrate.
NET_DVR_CALIBRATE_DATA_CONFILICT	307	Calibration error.
NET_DVR_CALIBRATE_CALC_FAIL	308	Failed to calculate camera calibration parameter.
NET_DVR_CALIBRATE_LINE_OUT_RECT	309	The input calibrating line exceeds the external rectangle sample.
NET_DVR_ENTER_RULE_NOT_READY	310	Enter rule not ready.
NET_DVR_AID_RULE_NO_INCLUDE_LANE	311	It does not include lane in the traffic event rule (especial for traffic jam or driving against the traffic).
NET_DVR_LANE_NOT_READY	312	Lane not ready.
NET_DVR_RULE_INCLUDE_TWO_WAY	313	There are two different directions in event rule.
NET_DVR_LANE_TPS_RULE_CONFLICT	314	The lane conflicts with the data rule.
NET_DVR_NOT_SUPPORT_EVENT_TYPE	315	The event type is not supported by the device.
NET_DVR_LANE_NO_WAY	316	The lane has no direction.
NET_DVR_SIZE_FILTER_ERROR	317	The size of filter is illegal.
NET_DVR_LIB_FFL_NO_FACE	318	There is no face when feature point positioning.
NET_DVR_LIB_FFL_IMG_TOO_SMALL	319	The input image is too small when feature point positioning.
NET_DVR_LIB_FD_IMG_NO_FACE	320	The input image has no face when detecting face in single image.
NET_DVR_LIB_FACE_TOO_SMALL	321	Face is too small when building model.
NET_DVR_LIB_FACE_QUALITY_TOO_BAD	322	Face image is of poor quality when building model.
NET_DVR_KEY_PARAM_ERR	323	Advanced parameter setting error.
NET_DVR_CALIBRATE_DATA_ERR	324	Calibration sample size error, or data value error, or sample points beyond the horizon
NET_DVR_CALIBRATE_DISABLE_FAIL	325	The configured rules do not allow to cancel calibration.

6.2 Error code of RTSP communication library

Error	Value	Message
NET_DVR_RTSP_GETPORTFAILED	407	RTSP port getting error.
NET_DVR_RTSP_DESCRIBESENDTIMEOUT	411	Sending "RTSP DECRIBE" is timeout.
NET_DVR_RTSP_DESCRIBESENDERROR	412	Failed to send "RTSP DECRIBE".
NET_DVR_RTSP_DESCRIBERECVTIMEOUT	413	Receiving "RTSP DECRIBE" is timeout.
NET_DVR_RTSP_DESCRIBERECVDATALOST	414	Receiving data of "RTSP DECRIBE" error.
NET_DVR_RTSP_DESCRIBERECVERROR	415	Failed to receive "RTSP DECRIBE".

NET_DVR_RTSP_DESCRIBESERVERERR	416	"RTSP DECRIBE" device returns the error that values 401 or 501.
NET_DVR_RTSP_SETUPSENDTIMEOUT	421	Sending "RTSP SETUP" is timeout.
NET_DVR_RTSP_SETUPSENDERROR	422	Sending "RTSP SETUP" error.
NET_DVR_RTSP_SETUPRECVTIMEOUT	423	Receiving "RTSP SETUP" is timeout.
NET_DVR_RTSP_SETUPRECVDATALOST	424	Receiving data of "RTSP SETUP" error.
NET_DVR_RTSP_SETUPRECVERROR	425	Failed to receive "RTSP SETUP".
NET_DVR_RTSP_OVER_MAX_CHAN	426	"RTSP SETUP" device returns the error that values 401 or 501. It exceeds the max connection number.
NET_DVR_RTSP_PLAYSENDTIMEOUT	431	Sending "RTSP PLAY" is timeout.
NET_DVR_RTSP_PLAYSENDERROR	432	Sending "RTSP PLAY" error.
NET_DVR_RTSP_PLAYRECVTIMEOUT	433	Receiving "RTSP PLAY" is timeout.
NET_DVR_RTSP_PLAYRECVDATALOST	434	Receiving data of "RTSP PLAY" error.
NET_DVR_RTSP_PLAYRECVERROR	435	Failed to receive "RTSP PLAY".
NET_DVR_RTSP_PLAYSERVERERR	436	"RTSP PLAY" device returns the error that values 401 or 501.
NET_DVR_RTSP_TEARDOWNSENDTIMEOUT	441	Sending "RTSP TEARDOWN" is timeout.
NET_DVR_RTSP_TEARDOWNSENDERROR	442	Sending "RTSP TEARDOWN" error.
NET_DVR_RTSP_TEARDOWNRECVTIMEOUT	443	Receiving "RTSP TEARDOWN" is timeout.
NET_DVR_RTSP_TEARDOWNRECVDATALOST	444	Receiving data of "RTSP TEARDOWN" error.
NET_DVR_RTSP_TEARDOWNRECVERROR	445	Failed to receive "RTSP TEARDOWN".
NET_DVR_RTSP_TEARDOWNSERVERERR	446	"RTSP TEARDOWN" device returns the error that values 401 or 501.

6.3 Error code of software decoding library

Error	Value	Message
NET_PLAYM4_NOERROR	500	No error.
NET_PLAYM4_PARA_OVER	501	Input parameter is invalid.
NET_PLAYM4_ORDER_ERROR	502	API calling order error.
NET_PLAYM4_TIMER_ERROR	503	Failed to create multimedia clock.
NET_PLAYM4_DEC_VIDEO_ERROR	504	Failed to decode video data.
NET_PLAYM4_DEC_AUDIO_ERROR	505	Failed to decode audio data.
NET_PLAYM4_ALLOC_MEMORY_ERROR	506	Failed to allocate memory.
NET_PLAYM4_OPEN_FILE_ERROR	507	Failed to open the file.
NET_PLAYM4_CREATE_OBJ_ERROR	508	Failed to create thread event.
NET_PLAYM4_CREATE_DDRAW_ERROR	509	Failed to create DirectDraw object.
NET_PLAYM4_CREATE_OFFSCREEN_ERROR	510	Failed to create backstage cache for OFFSCREEN

		mode.
NET_PLAYM4_BUF_OVER	511	Buffer overflow, failed to input stream.
NET_PLAYM4_CREATE_SOUND_ERROR	512	Failed to create audio equipment.
NET_PLAYM4_SET_VOLUME_ERROR	513	Failed to set the volume.
NET_PLAYM4_SUPPORT_FILE_ONLY	514	This API can be called only for file playback mode.
NET_PLAYM4_SUPPORT_STREAM_ONLY	515	This API can be called only when playing stream.
NET_PLAYM4_SYS_NOT_SUPPORT	516	Not support by the system. Decoder can only work on the system above Pentium 3.
NET_PLAYM4_FILEHEADER_UNKNOWN	517	There is no file header.
NET_PLAYM4_VERSION_INCORRECT	518	The version mismatch between decoder and encoder.
NET_PLAYM4_INIT_DECODER_ERROR	519	Failed to initialize the decoder.
NET_PLAYM4_CHECK_FILE_ERROR	520	The file is too short, or the stream data is unknown.
NET_PLAYM4_INIT_TIMER_ERROR	521	Failed to initialize multimedia clock.
NET_PLAYM4_BLT_ERROR	522	BLT failure.
NET_PLAYM4_UPDATE_ERROR	523	Failed to update overlay surface
NET_PLAYM4_OPEN_FILE_ERROR_MULTI	524	Failed to open video & audio stream file.
NET_PLAYM4_OPEN_FILE_ERROR_VIDEO	525	Failed to open video stream file.
NET_PLAYM4_JPEG_COMPRESS_ERROR	526	JPEG compression error.
NET_PLAYM4_EXTRACT_NOT_SUPPORT	527	Don't support the version of this file.
NET_PLAYM4_EXTRACT_DATA_ERROR	528	Extract video data failed.