

---

# **SDK Programming Guide**

**V3.4.7**

---

## Preface

### Purpose

Welcome to use NetSDK(SDK for short below) programming guide.

SDK is the software development kit used by software developer when developing DVR(digital video recorder), DVS(digital video server), IPC(IP camera), SD(speed dome), intelligent devices to monitor network application.


This document has described functions, interfaces and calling relationship between functions in SDK, and also provided example codes.

### Target reader

SDK software development engineers

### Symbols

The symbols shown in the sheet below may appear in the document, their meanings are shown below.

Symbol	Instruction
 instruction	Means: supplement to the body of the text

### Revise history

Time	Revised content
2016-01-15	Create
2016-08-16	Revise

# Catalogue

<b>PREFACE .....</b>	<b>2</b>
<b>CATALOGUE .....</b>	<b>3</b>
<b>1 INTRODUCTION .....</b>	<b>10</b>
1.1 ABSTRACT .....	10
1.2 APPLICABILITY .....	11
<b>2 MAJOR FUNCTIONS .....</b>	<b>12</b>
2.1 SDK INITIALIZATION MODULE .....	12
2.1.1 Module Intro .....	12
2.1.2 Interface Overview .....	12
2.1.3 SDK Initialization Instruction .....	13
2.1.4 Example Code .....	14
2.2 DEVICE LOGIN .....	17
2.2.1 Intro .....	17
2.2.2 Interface Overview .....	18
2.2.3 Process Instruction .....	18
2.2.4 Example Codes .....	19
2.3 REAL-TIME MONITORING .....	23
2.3.1 Intro .....	23
2.3.2 Interface Overview .....	23
2.3.3 Process Instruction .....	24
2.3.4 Example Code .....	27
2.4 RECORD PLAYBACK .....	38
2.4.1 Intro .....	38
2.4.2 Interface Overview .....	39
2.4.3 Process Instruction .....	39
2.4.4 Example Code .....	44

2.5	RECORD DOWNLOAD.....	59
2.5.1	Intro .....	59
2.5.2	Interface Overview .....	60
2.5.3	Process Instruction .....	60
2.5.4	Example Code.....	65
2.6	PTZ CONTROL.....	80
2.6.1	Intro .....	80
2.6.2	Interface Overview .....	80
2.6.3	Process Instruction .....	81
2.6.4	Example Code.....	83
2.7	AUDIO INTERCOM.....	95
2.7.1	Intro .....	95
2.7.2	Interface Overview .....	95
2.7.3	Process Instruction .....	96
2.7.4	Example Code.....	100
2.8	SNAPSHOT.....	129
2.8.1	Intro .....	129
2.8.2	Interface Overview .....	129
2.8.3	Process Instruction .....	129
2.8.4	Example code .....	130
2.9	REPORT ALARM.....	138
2.9.1	Intro .....	138
2.9.2	Interface Overview .....	138
2.9.3	Process Instruction .....	138
2.9.4	Example Code.....	140
2.10	DEVICE SEARCH.....	147
2.10.1	Intro .....	147
2.10.2	Interface Overview .....	148
2.10.3	Process Intruction .....	148
2.10.4	Example Code.....	150

2.11	INTELLIGENT EVENT REPORT AND SNAPSHOT.....	162
2.11.1	Intro .....	162
2.11.2	Interface Overview .....	162
2.11.3	Process Intruction .....	162
2.11.4	Example Code.....	164
<b>3</b>	<b>CALLBACK FUNCTION DEFINITION .....</b>	<b>174</b>
3.1	FDISCONNECT.....	174
3.2	FHAVERECONNECT.....	175
3.3	FREALDATACALLBACKEX.....	176
3.4	FDOWNLOADPOSCALLBACK.....	177
3.5	FDATACALLBACK .....	178
3.6	FTIMEDOWNLOADPOSCALLBACK.....	179
3.7	FMESSCALLBACK.....	180
3.8	FSEARCHDEVICESCB.....	182
3.9	FANALYZERDATACALLBACK .....	183
3.10	FSNAPREV .....	185
3.11	FREALPLAYDISCONNECT.....	186
3.12	PFAUDIODATACALLBACK.....	187
	<b>APPENDIX 1 STRUCT DEFINITION .....</b>	<b>189</b>
	NET_DEVICEINFO.....	189
	NET_IN_STARTLOGINEX.....	190
	NET_OUT_STARTLOGINEX .....	191
	NET_PARAM .....	191
	NET_DEVICEINFO_Ex .....	193
	TAGVIDEOFRAMEPARAM.....	194
	TAGCBPCMDATAPARAM .....	197
	NET_TIME .....	198
	NET_RECORDFILE_INFO .....	199
	CFG_PTZ_PROTOCOL_CAPS_INFO.....	200

CFG_PTZ_MOTION_RANGE .....	207
CFG_PTZ_LIGHTING_CONTROL.....	208
DHDEV_TALKFORMAT_LIST .....	208
DHDEV_TALKDECODE_INFO .....	209
DHDEV_SYSTEM_ATTR_CFG .....	210
NET_SPEAK_PARAM.....	213
NET_TALK_TRANSFER_PARAM .....	214
DEVICE_NET_INFO_EX .....	215
MANUAL_SNAP_PARAMETER .....	218
OPR_RIGHT_EX .....	218
OPR_RIGHT_NEW .....	219
NET_DEV_CHN_COUNT_INFO .....	219
NET_CHN_COUNT_INFO .....	220
NET_IN_SNAP_CFG_CAPS.....	221
NET_OUT_SNAP_CFG_CAPS.....	222
DH_RESOLUTION_INFO .....	223
DEOENC_VIDEOENC_OPT .....	223
CFG_VIDEO_FORMAT .....	225
CFG_AUDIO_ENCODE_FORMAT .....	229
CFG_VIDEO_COVER.....	231
CFG_COVER_INFO .....	232
CFG_RECT.....	233
CFG_RGBA .....	234
CFG_ENCODE_INFO .....	234
SNAP_PARAMS.....	237
DH_VERSION_INFO .....	238
DH_DSP_ENCODECAP.....	239
<b>APPENDIX 2 ENUM DEFINITION.....</b>	<b>243</b>
NET_DEVICE_TYPE .....	243
EM_LOGIN_SPAC_CAP_TYPE.....	245

DH_REALPLAYTYPE .....	246
EM_QUERY_RECORD_TYPE .....	247
EM_USEDEV_MODE .....	248
EM_SUPPORT_FOCUS_MODE .....	250
DH_PTZ_CONTROLTYPE .....	250
DH_EXTPTZ_CONTROLTYPE .....	251
DH_TALK_CODING_TYPE .....	254
CTRLTYPE .....	255
CFG_VIDEO_COMPRESSION .....	267
CFG_BITRATE_CONTROL .....	268
CFG_IMAGE_QUALITY .....	268
CFG_H264_PROFILE_RANK .....	268
CFG_AUDIO_FORMAT .....	269
EM_REALPLAY_DISCONNECT_EVENT_TYPE .....	269
<b>APPENDIX 3 INTERFACE DEFINITION .....</b>	<b>271</b>
CLIENT_INIT .....	271
CLIENT_CLEANUP .....	272
CLIENT_GETSDKVERSION .....	273
CLIENT_GETLASTERROR .....	273
CLIENT_SETAUTORECONNECT .....	274
CLIENT_SETCONNECTTIME .....	276
CLIENT_SETNETWORKPARAM .....	277
CLIENT_LOGINEX2 .....	277
CLIENT_LOGOUT .....	280
CLIENT_REALPLAYEX .....	281
CLIENT_STOPREALPLAYEX .....	282
CLIENT_SETREALDATACALLBACKEX .....	283
CLIENT_FINDFILE .....	286
CLIENT_FINDNEXTFILE .....	289
CLIENT_FINDCLOSE .....	290

---

CLIENT_PLAYBACKBYTIMEEX .....	291
CLIENT_STOPPLAYBACK .....	294
CLIENT_GETPLAYBACKOSD TIME .....	294
CLIENT_QUERYRECORDFILE.....	296
CLIENT_DOWNLOADBYTIMEEX.....	299
CLIENT_STOPDOWNLOAD.....	301
CLIENT_PLAYBACKBYRECORDFILEEX.....	302
CLIENT_PAUSEPLAYBACK .....	304
CLIENT_SEEKPLAYBACK .....	305
CLIENT_FASTPLAYBACK .....	306
CLIENT_SLOWPLAYBACK.....	307
CLIENT_NORMALPLAYBACK .....	308
CLIENT_DOWNLOADBYRECORDFILEEX.....	309
CLIENT_PARSEDATA .....	311
CLIENT_DHPTZCONTROLEX2.....	312
CLIENT_QUERYNEWSYSTEMINFO.....	314
CLIENT_SETDEVICEMODE.....	317
CLIENT_STARTSEARCHDEVICES.....	318
CLIENT_QUERYDEVSTATE .....	319
CLIENT_STARTTALKEX.....	321
CLIENT_STOPTALKEX .....	322
CLIENT_RECORDSTARTEX.....	322
CLIENT_RECORDSTOPEX.....	323
CLIENT_TALKSEND DATA .....	324
CLIENT_AUDIODEC EX.....	325
CLIENT_SETDVRMESSCALLBACK.....	326
CLIENT_STARTLISTEN EX .....	327
CLIENT_STOPLISTEN .....	328
CLIENT_STOPSEARCHDEVICES.....	329
CLIENT_SEARCHDEVICESBYIPS.....	330



CLIENT_REALLOADPICTUREEX .....	331
CLIENT_CONTROLDEVICEEX.....	333
CLIENT_STOPLOADPIC.....	335
CLIENT_GETDOWNLOADPOS.....	336
CLIENT_SETSNAPREVCALLBACK.....	337
CLIENT_SNAPPICTUREEX .....	338

# 1 Introduction

## 1.1 Abstract

SDK is the software development kit used by software developer when developing DVR(digital video recorder), DVS(digital video server), IPC(IP camere), SD(speed dome), intelligent devices to monitor network application.

The kit mainly includes the following functions:

Device login, real-time monitoring, record playback and playback control, record download, PTZ control, audio intercom, video snapshot, alarm report, device search, intelligent event report and picture snapshot, user management and some other functions (device restart, device upgrade, video image parameter setup, channel name setup, device network parameter setup and etc.) .

Files in development kit are shown below:

Library type	Library file name	Library description
Network library	dhnetSDK.h	Header file
	avglobal.h	Header file
	avnetSDK.dll	3 <sup>rd</sup> generation library
	dhconfigSDK.dll	Configure library
	dhconfigSDK.h	Header file
	dhconfigSDK.lib	Lib file
	dhnetSDK.lib	Lib file
	dhnetSDK.dll	Interface library
	Infra.dll	3 <sup>rd</sup> generation aux library
	json.dll	3 <sup>rd</sup> generation aux library
	NetFramework.dll	3 <sup>rd</sup> generation aux library
	Stream.dll	3 <sup>rd</sup> generation aux library
	StreamSvr.dll	3 <sup>rd</sup> generation aux library
AUX Library	dhplay.dll	Decoding aux library
	adpcmdecode.dll	Decoding aux library
	postproc.dll	Decoding aux library

	mpeg4enc.dll	Decoding aux library
	mpeg4dec.dll	Decoding aux library
	mp3dec.dll	Decoding aux library
	mjpegdec.dll	Decoding aux library
	h264dec.dll	Decoding aux library
	amrdec.dll	Decoding aux library
	aacdec.dll	Decoding aux library
	DhDecode.dll	mobile snapshotting decoding library
	IvsDrawer.dll	Intelligent drawing library

## 1.2Applicability

Recommended memory: unused memory should not below 256M.

Supported system by SDK:

- Windows

Windows10/Windows8/Windows7/vista/XP/2000 and Windows Server 2008/2003.

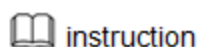
- Linux

CentOS/Red Hat/SUSE and other general Linux OS. GCC version not below 4.1.2.

Applicable devices for each functional module are shown below:

Function	Supported Device
Device Login	DVR, NVR, IPC and SD
real-time monitoring	DVR, NVR, IPC and SD
Record Playback and Playback Control	storage device as DVR and NVR
Record Download	storage device DVR and NVR
PTZ Control	SD
audio intercom	DVR, NVR, IPC and SD
Video snapshot	DVR, NVR, IPC and SD
Alarm Report	DVR, NVR, IPC and SD
Device Search	DVR, NVR, IPC and SD
Intelligent Event Report and Snapshot	IVS, mobile and smart SD

## 2 Major Functions



All the example codes are tested under Windows OS only.

### 2.1 SDK Initialization Module

#### 2.1.1 Module Intro

Initialization is the first step of SDK business. The module itself does not include surveillance business, interfaces in this chapter will not make any interaction with Dahua device, but the module is responsible for SDK initialization, and setting properties which influence business.

##### Warnings:

1. The initialization module of SDK will occupy a certain memory .
2. Within the same process, only the first initialization is valid.
3. You need to call cleanup interface to release resources after using SDK completely.

#### 2.1.2 Interface Overview

SDK initialization module involves the following interfaces:

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for SDK initialization
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources
<a href="#">CLIENT_GetSDKVersion</a>	Interface for getting SDK version information
<a href="#">CLIENT_GetLastError</a>	Interface for getting the function execution failure code
<a href="#">CLIENT_SetAutoReconnect</a>	Interface for setting reconnection callback function after disconnection
<a href="#">CLIENT_SetConnectTime</a>	Interface for setting device connection timeout value and trial times
<a href="#">CLIENT_SetNetworkParam</a>	Interface for setting log in network environment

### 2.1.3 SDK Initialization Instruction

a) The SDK initialization process is shown below in Figure 2-1.

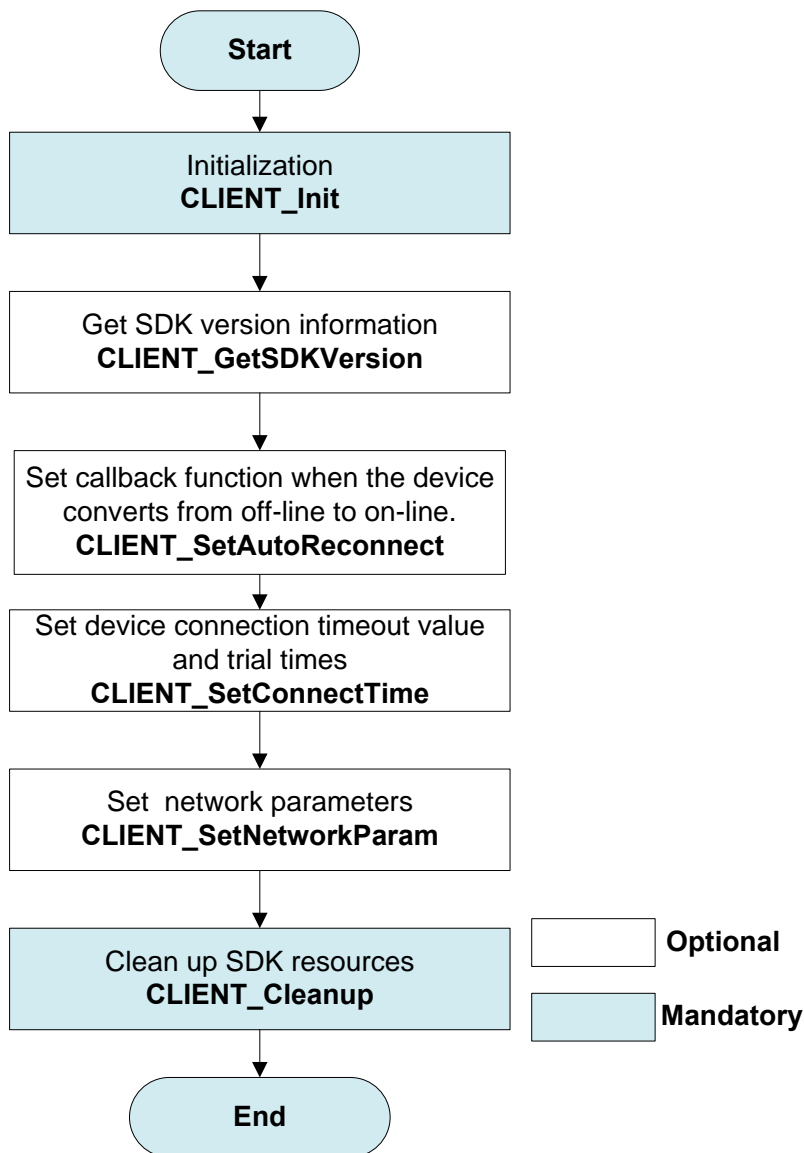


Figure 2-1 SDK initialization process

#### Process description:

1. Call CLIENT\_Init to initialize SDK.
2. Call CLIENT\_GetSDKVersion to get SDK version information (optional) .
3. Call CLIENT\_SetAutoReconnect to set offline reconnection callback function (optional,suggest calling) , Internal SDK auto connect again when the device disconnect.
4. Call CLIENT\_SetConnectTime to set device connection timeout value and trial times

(optional) .

5. Call CLIENT\_SetNetworkParam to set network parameters, parameters include login timeout value and trial times (optional) .
6. Call CLIENT\_Cleanup to clean up SDK resources After using SDK function.

## 2.1.4 Example Code

```
#include<windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;

//*****
// commonly used callback set declaration.

// callback function used when device disconnected.
// It is not recommended to call SDK interface in the SDK callback function.
//The callback function set by CLINET_Init,When the device is offline,SDK will call this callback
function.
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// Callback function set after the device reconnected successfully.
// It is not recommended to call SDK interface in the SDK callback function.
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function.
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);
//*****
void InitTest()
{
    // SDK initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
}
```

```
}
else
{
    printf("Initialize client SDK done; \n");
}

// This operation is optional.
//Get the SDK version info.
DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
printf("NetSDK version is [%d]", dwNetSdkVersion);

// Set CLIENT_SetAutoReconnect interface.Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally
//This operation is optional,but recommended.
CLIENT_SetAutoReconnect(&HaveReConnect, 0);

// Set device connection timeout value and trial times.
//This is an optional.
int nWaitTime = 5000;    // timeout value is 5 seconds
int nTryTimes = 3;      //If timeout,it will try to log in three times.
CLIENT_SetConnectTime(nWaitTime, nTryTimes);

//Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
//This operation is optional.

NET_PARAM stuNetParm = {0};

stuNetParm.nConnectTime = 3000; //when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

// The first time logging to device, some data is needed to be initialized to enable normal
business function. It is recommended to wait for a while after login, the waiting time varies by
devices.
    Sleep(1000);
    printf("\n");
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
}
```

```
    }
    // task realizing operation
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();

    //Logout device operation.
    //Cleaning up initialization resources.
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//*****
//commonly used callback functions definition
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
```



```
{  
    printf("Call HaveReConnect\n");  
    printf("ILoginID[0x%x]", ILoginID);  
    if (NULL != pchDVRIP)  
    {  
        printf("pchDVRIP[%s]\n", pchDVRIP);  
    }  
    printf("nDVRPort[%d]\n", nDVRPort);  
    printf("dwUser[%p]\n", dwUser);  
    printf("\n");  
}
```

## 2.2 Device Login

### 2.2.1 Intro

#### Precondition

Before logging to device, successfully initialization should be done.

#### Overview

Device login, as device registration is the precondition of other businesses.

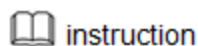
When SDK initialization is complete, user need to login to Dahua device first. Only when the sole valid login ID is generated, can we operate other businesses. Login ID is the unique sign to recognize the login, other function SDK follows will require this login ID.

#### Login Method

It has three login methods according to function and use scene: sync login, async login and auto registration login. Please refer to Ch 2.2.3.1, 2.2.3.2 and 2.2.3.3 to learn about them.

#### Reconnection

SDK can set device reconnection function. When encounter some special conditions (offline, outage) which makes device become offline, it will try to reconnect to device continuously within SDK until being online.



- Among the three login methods, auto registration login don't support reconnection.
- User can call SDK self-carried reconnection function, as well as can call login and logout interface at application layer to manually control reconnection business.

#### NOTE

1. The provided login operation is for Dahua devices only, not for other manufacturers' devices. Please use the login operation carefully, otherwise the device will not be able to login successfully.
2. Login and logout should be used as a pair. In case of resource leak, you must call logout interface to logout user and release SDK resource.

### 2.2.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for initialization
<a href="#">CLIENT_Cleanup</a>	Interface for SDK clearance
<a href="#">CLIENT_LoginEx2</a>	Extensive interface 2 for sync login
<a href="#">CLIENT_Logout</a>	Interface for logout device
<a href="#">CLIENT_GetLastError</a>	Interface for getting error code after failed calling interface

### 2.2.3 Process Instruction

When client with SDK has fluent connection to Dahua device, you can start the login operation. When the login interface return a valid login ID, your login is successful.

Login business process flow sheet is shown in Figure 2-2.

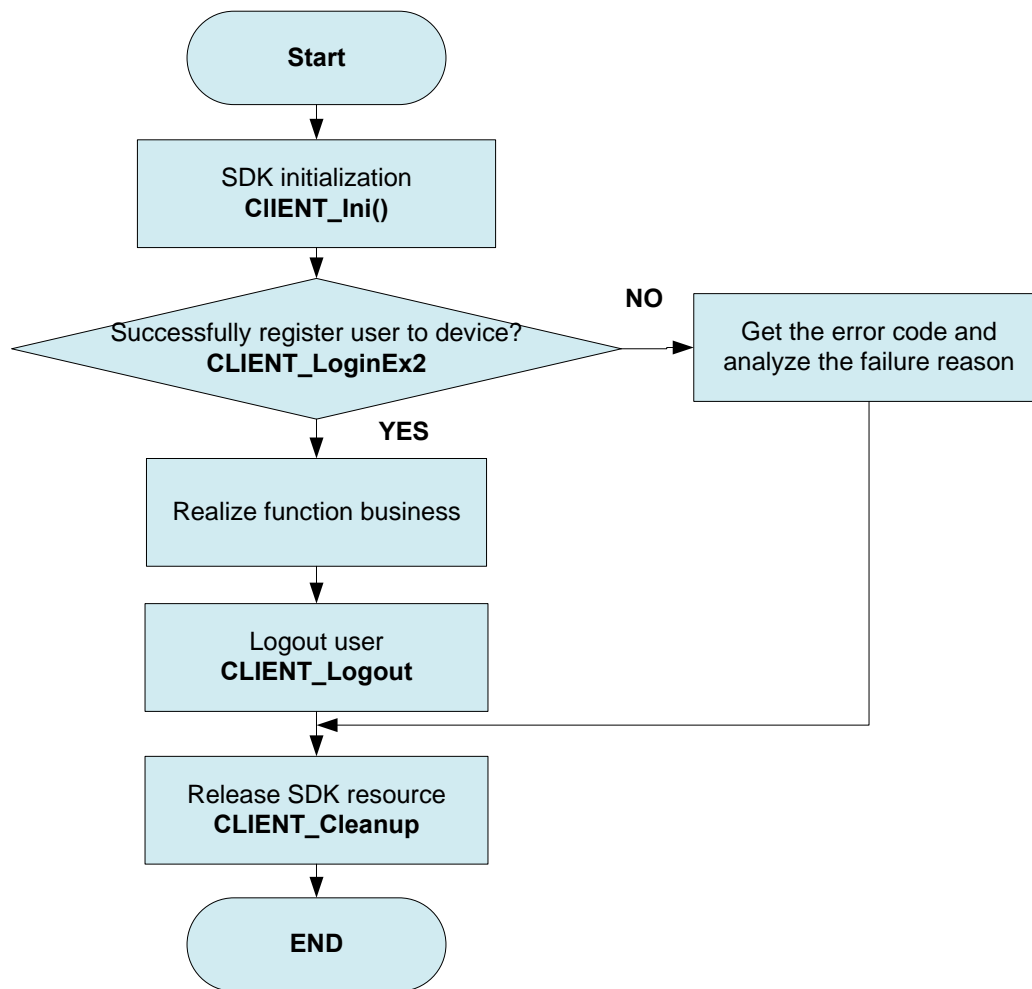


Figure 2-2 login business process flow sheet

**Process description:**

1. SDK initialization.
2. After successful initialization, call CLIENT\_LoginEx2 to login device.
3. After successful login, users can realize business as needed.
4. After realizing needed business, call CLIENT\_Logout to logout device.
5. Last but not the least, call CLIENT\_Cleanup to release SDK resource.

## 2.2.4 Example Codes

```
#include <windows.h>
#include <stdio.h>
#include " dhnetsdk.h "
```

```
#pragma comment(lib , " dhnetSDK.h ")

static LLONG gILoginHandle = 0L;
static char g_szDevIp[32] = "172.32.4.25";
static WORD g_nPort = 37777;
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
static BOOL g_bSDKInitFlag = FALSE;

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisconnectFunc(ULONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)

// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)

//*****
void InitTest()
{
    // SDK Initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    //Get the SDK version information.
    // This operation is optional.
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
```

```
printf("NetSDK version is [%d]\n", dwNetSdkVersion);

// set CLIENT_SetAutoReconnect interface.Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally
//This operation is optional,but recommended.
CLIENT_SetAutoReconnect(&HaveReConnect, 0);

// Set device connection timeout value and trial times.
//This operation is optional
int nWaitTime = 5000; // timeout value is 5 seconds
int nTryTimes = 3; //If timeout,it will try to log in three times.
CLIENT_SetConnectTime(nWaitTime, nTryTimes);

//Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
//This operation is optional.
NET_PARAM stuNetParm = {0};

stuNetParm.nConnectTime = 3000; //when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

CLIENT_SetNetworkParam(&stuNetParm);

NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == gILoginHandle)
{
    //Login device
    gILoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if (0 == gILoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhnetsdk.h.
        It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        //Example: #define NET_NOT_SUPPORTED_EC(23) //Now SDK does not
support this function, error code is 0x80000017, Decimal number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
    }
    else
```

```
        {
            printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
        }
        // The first time logging to device, some data is needed to be initialized to enable
normal business function. It is recommended to wait for a while after login, the waiting time varies
by devices.
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    // business realization
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // logout device
    if (0 == g_lLoginHandle)
    {
        if (FALSE == CLIENT_Logout(g_lLoginHandle))
        {
            printf("CLIENT_Logout Failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            {
                g_lLoginHandle = 0;
            }
        }
    }
    // clean up resource
    if (TRUE == g_bSDKInitFlag)
    {
        {
            CLIENT_Cleanup();
            g_bSDKInitFlag = FALSE;
        }
    }
    return;
}

int main()
{
```

```
    InitTest();
    RunTest();
    EndTest();
    return 0;
}
//*****
// commonly used callback set definition

void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

## 2.3 Real-time Monitoring

### 2.3.1 Intro

Real-time monitoring which is to get real time stream from storage device or front-end device is an important part of a surveillance system.

After login to device, SDK can obtain main stream, aux stream and the third stream from device.

- Support: Config front-end device bit stream resolution, encode, bit rate and etc.
- Support: Setup of image saturation, contrast, exposure and etc.
- Support: User conveys window handle, SDK analyzes stream and play directly.
- Support: Callback real-time stream data to user, let user process by himself.
- Support: Save real-time record to specific folder, user can save callback stream to achieve it or call SDK interface to realize it.

### 2.3.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for SDK Initialization.
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources.
<a href="#">CLIENT_LoginEx2</a>	Extensive interface 2 for Sync login.
<a href="#">CLIENT_RealPlayEx</a>	Extensive Interface for starting real-time monitoring
<a href="#">CLIENT_StopRealPlayEx</a>	Extensive interface for stopping real-time monitoring
<a href="#">CLIENT_SetRealDataCallBackEx</a>	Callback function Extensive interface for setting real-time monitoring data
<a href="#">CLIENT_Logout</a>	Interface for logout
<a href="#">CLIENT_GetLastError</a>	Interface for getting the function execution error code

### 2.3.3 Process Instruction

Real-time surveillance business includes the following two methods:

- SDK decoding and playing

SDK realizes real-time play by calling playsdk library in aux library.

- The third party decoding and playing

SDK only callback real-time monitoring data stream to user, user decodes and plays with a third-party library.

#### 2.3.3.1 SDK Decoding and Playing Process

SDK decoding and playing process flow sheet is shown below in Figure 2-3.



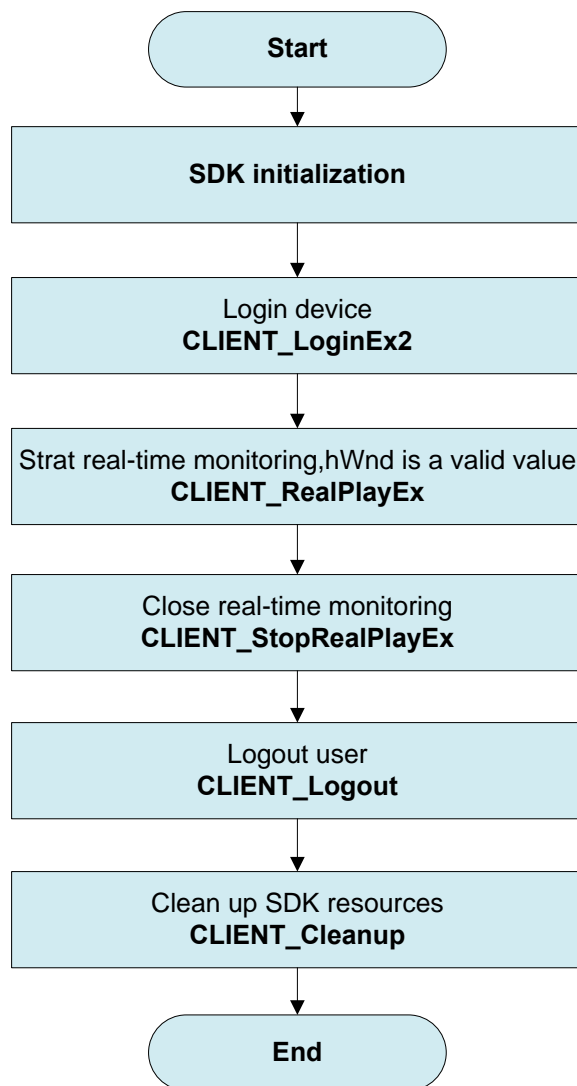


Figure 2-3 SDK Decoding and Playing Process

**Process description:**

1. SDK initialization
2. After successful initialization, call **CLIENT\_LoginEx2** to login device.
3. After successful login, call **CLIENT\_RealPlayEx** to start real-time monitoring, parameter hWnd is a valid window handle.
4. After completing real-time monitoring use, call **CLIENT\_StopRealPlayEx** to stop real-time monitoring.
5. After realizing needed business, call **CLIENT\_Logout** to logout device.
6. Last but not the least, call **CLIENT\_Cleanup** to release SDK resources.

### 2.3.3.2 The third Party Decoding and Playing Process

The third party decoding and playing process is shown below in Figure 2-4.

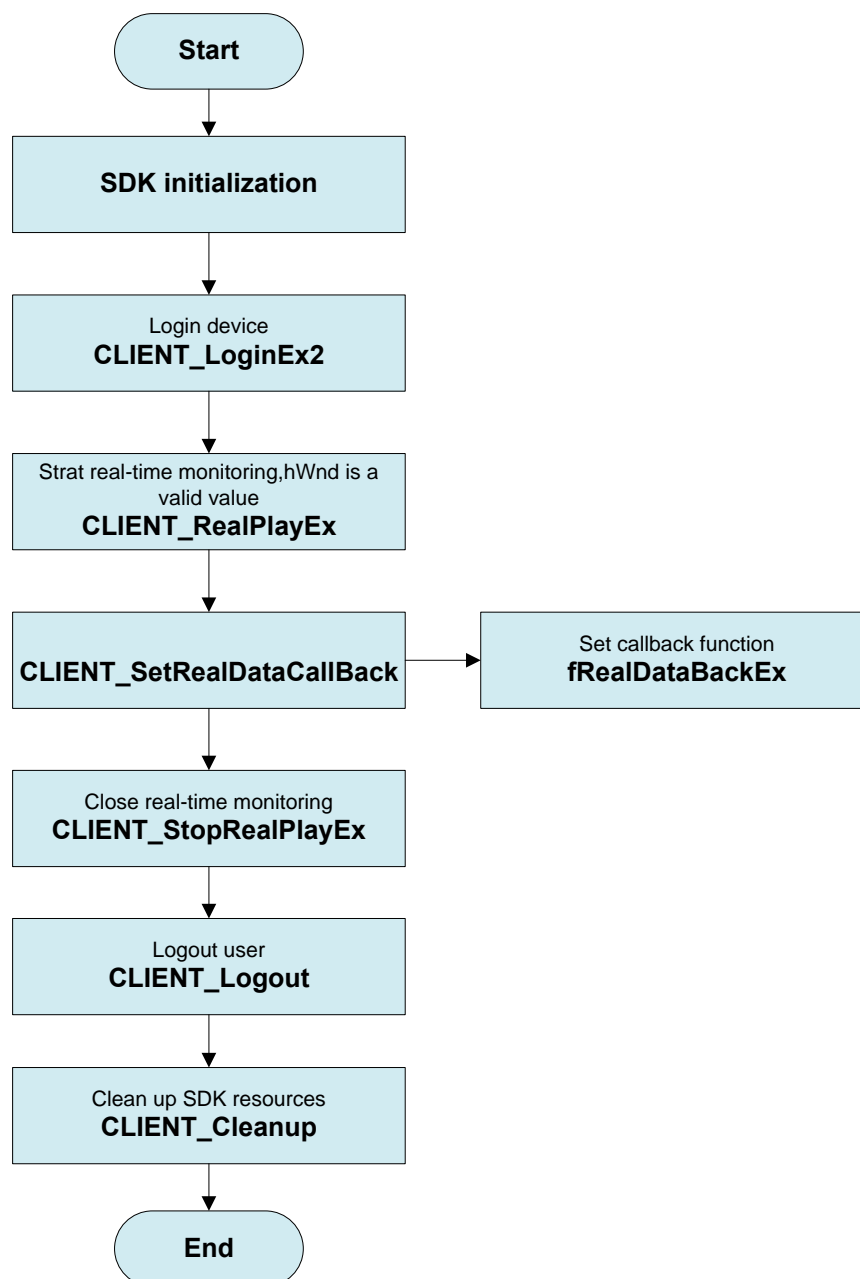


Figure 2-4 The third party decoding and playing process

#### Process description:

1. SDK initialization.
2. After successful initialization, call **CLIENT\_LoginEx2** to login device.
3. After successful login, call **CLIENT\_RealPlayEx** to start real-time monitoring, hWnd is a

valid window handle.

4. Set real-time data callback function in CLIENT\_SetRealDataCallBackEx.
5. Save real-time data in callback function for further use. It is not recommended to do any work but data stored and data delivered, or the performance will be affected when there are too many monitoring channels.
6. After completing real-time monitoring, call CLIENT\_StopRealPlayEx to stop real-time monitoring.
7. After realizing needed business, call CLIENT\_Logout to logout device.
8. Last but not the least, call CLIENT\_Cleanup to release SDK resources.

## 2.3.4 Example Code

### 2.3.4.1 SDK Decoding and Playing

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

typedef HWND (WINAPI *PROCGETCONSOLEWINDOW)();
PROCGETCONSOLEWINDOW GetConsoleWindow;

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lRealHandle = 0;
static char g_szDevIp[32] = "172.11.1.88";
static WORD g_nPort = 37777;          // Before tcp connects port,the port need to keep with tcp
port config in expected login
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
```

```
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);
//*****
void InitTest()
{
    // SDK Initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    //Get the SDK version info
    // This operation is optional.
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // set CLIENT_SetAutoReconnect interface.Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally
    //This operation is optional, but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

    // Set device connection timeout value and trial times.
    //This operation is optional.
    int nWaitTime = 5000;    // timeout value is 5 seconds
    int nTryTimes = 3;        //if timeout,it will try to log in three times.
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    //Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
    //This operation is optional.
```

```
NET_PARAM stuNetParm = {0};

stuNetParm.nConnectTime = 3000; //when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);


NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_LoginHandle)
{
    //Login device
    g_LoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if(0 == g_LoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhnetsdk.h.
        It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        //Example: #define NET_NOT_SUPPORTED_EC(23) //Now SDK does not
        support this function, error code is 0x80000017, Decimal number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
    }
    else
    {
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
    }
    // The first time logging to device, some data is needed to be initialized to enable
    normal business function. It is recommended to wait for a while after login, the waiting time varies
    by devices.
    Sleep(1000);
    printf("\n");
}

void RunTest()
{
    // judge whether initialization is successful or not
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
    //judge whether login device or not
```

```
    if (0 == g_ILoginHandle)
    {
        return;
    }

    // realize real-time monitoring
    //Get the console window handle
    HMODULE hKernel32 = GetModuleHandle("kernel32");
    GetConsoleWindow =
(PROCGETCONSOLEWINDOW)GetProcAddress(hKernel32,"GetConsoleWindow");
    HWND hWnd = GetConsoleWindow();

    printf("user can input any key to quit during real play!\n");
    Sleep(1000);

    //start real-time monitoring
    int nChannelID = 0;    // preview channel NO.
    DH_RealPlayType emRealPlayType = DH_RType_Realplay;    // real-time monitoring
    g_IRealHandle = CLIENT_RealPlayEx(g_ILoginHandle, nChannelID, hWnd,
emRealPlayType);
    if (0 == g_IRealHandle)
    {
        printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // Close preview
    if (0 != g_IRealHandle)
    {
        if(FALSE == CLIENT_StopRealPlayEx(g_IRealHandle))
        {
            printf("CLIENT_StopRealPlayEx Failed!Last Error[%x]\n",
CLIENT_GetLastError());
        }
        else
        {
            g_IRealHandle = 0;
        }
    }
}
```

```
//Exit device
if (0 != g_ILLoginHandle)
{
    if(FALSE == CLIENT_Logout(g_ILLoginHandle))
    {
        printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
    }
    else
    {
        g_ILLoginHandle = 0;
    }
}
// Clean up initialization resources
if (TRUE == g_bNetSDKInitFlag)
{
    CLIENT_Cleanup();
    g_bNetSDKInitFlag = FALSE;
}
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//*****
//commonly used callback functions definition
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

```
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

#### 2.3.4.2 The third party decoding and playing

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_ILoginHandle = 0L;
static LLONG g_IRealHandle = 0;
static char g_szDevIp[32] = "172.11.1.88";
static WORD g_nPort = 37777; // Before tcp connects port,the port need to keep with tcp
port config in expected login
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);
```



```
// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

// Real-time monitoring data callback function
// Not recommended to call SDK interface in SDK callback function.
// The callback function is set by CLIENT_SetRealDataCallBackEx, When real-time data is
received, SDK will call function.
//Recommend user only save data in this callback, do not encode or decode data directly.
//In other words: Copy data to your storage and encode or decode data after leaving callback.
void CALLBACK RealDataCallBackEx(LLONG IRealHandle, DWORD dwDataType, BYTE
*pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser);

//*****
void InitTest()
{
    // SDK Initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    //Get the SDK version information.
    // This operation is optional.
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // set CLIENT_SetAutoReconnect interface.Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally
    //This operation is optional,but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

    // Set device connection timeout value and trial times.
```

```
//This operation is optional
int nWaitTime = 5000;    // timeout value is 5 seconds
int nTryTimes = 3;       //If timeout,it will try to log in three times.
CLIENT_SetConnectTime(nWaitTime, nTryTimes);

//Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
//This operation is optional.

NET_PARAM stuNetParm = {0};

stuNetParm.nConnectTime = 3000; //when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

CLIENT_SetNetworkParam(&stuNetParm);

NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_LoginHandle)
{
    //Login device
    g_LoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if (0 == g_LoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhnetsdk.h.
        It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        //Example: #define NET_NOT_SUPPORTED_EC(23)    //Now SDK does not
support this function, error code is 0x80000017, Decimal number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp , g_nPort ,
CLIENT_GetLastError());
    }
    else
    {
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp , g_nPort);
    }
    // The first time logging to device, some data is needed to be initialized to enable
normal business function. It is recommended to wait for a while after login, the waiting time varies
by devices.
    Sleep(1000);
    printf("\n");
}
```

```
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_ILoginHandle)
    {
        return;
    }

    // Real-time monitoring realization operation
    printf("user can input any key to quit during real play data callback!\n");
    Sleep(1000);

    //start real-time monitoring
    int nChannelID = 0;    // preview channel number
    DH_RealPlayType emRealPlayType = DH_RType_Realplay;    // real-time monitoring
    g_IRealHandle = CLIENT_RealPlayEx(g_ILoginHandle, nChannelID, NULL, emRealPlayType);
    if (0 == g_IRealHandle)
    {
        printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
        return;
    }
    else
    {
        DWORD dwFlag = 0x00000001;
        if (FALSE == CLIENT_SetRealDataCallBackEx(g_IRealHandle,
&RealDataCallBackEx, NULL, dwFlag))
        {
            printf("CLIENT_SetRealDataCallBackEx: failed! Error code: %x.\n",
CLIENT_GetLastError());
            return;
        }
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
}
```

```
    getchar();
    //Close preview
    if (0 != g_IRealHandle)
    {
        if (FALSE == CLIENT_StopRealPlayEx(g_IRealHandle))
        {
            printf("CLIENT_StopRealPlayEx Failed, g_IRealHandle[%x]! Last Error[%x]\n" ,
g_IRealHandle, CLIENT_GetLastError());
        }
        else
        {
            g_IRealHandle = 0;
        }
    }
    //Exit device
    if (0 != g_ILLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_ILLoginHandle))
        {
            printf("CLIENT_Logout Failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_ILLoginHandle = 0;
        }
    }
    // Clean up initialization resources.
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//*****
```

```
//commonly used callback functions definition
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK RealDataCallBackEx(LLONG IRealHandle, DWORD dwDataType, BYTE
*pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser)
{
    // if more than one real-time monitoring use same data callback, we can judge by
parameter IRealHandle.
    if (IRealHandle == g_IRealHandle)
    {
        switch(dwDataType)
        {
            case 0:
                //original A/V hybrid data
                printf("receive real data, param: IRealHandle[%p], dwDataType[%d],
pBuffer[%p], dwBufSize[%d], param[%p], dwUser[%p]\n", IRealHandle, dwDataType, pBuffer,
dwBufSize, param, dwUser);
```

```
        break;
    case 1:
        //standard video data

        break;
    case 2:
        //yuv data

        break;
    case 3:
        //pcm audio data

        break;
    case 4:
        //original audio data

        break;
    default:
        break;
    }
}
```

## 2.4 Record Playback

### 2.4.1 Intro

#### Overview

Record playback is to playback record of certain channels during specific periods, in order to locate target video for research from a large quantity of videos.

Playback function includes several operations, such as play, pause, quick play, slow play, dragging play and etc.

#### Record Playback Method

According to the different decoding method selected by users, record playback have two methods: SDK decoding playback and third-party decoding playback. please refer to Ch 2.4.3 for details.

## 2.4.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for SDK initialization
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources
<a href="#">CLIENT_LoginEx2</a>	Extensive interface 2 for Sync login
<a href="#">CLIENT_PlayBackByTimeEx</a>	Extensive interface for playback by time
<a href="#">CLIENT_SetDeviceMode</a>	Interface for setting work mode such as audio intercom, playback, right and etc.
<a href="#">CLIENT_StopPlayBack</a>	Interface for stopping record playback
<a href="#">CLIENT_GetPlayBackOsdTime</a>	Interface for getting playback OSD time
<a href="#">CLIENT_PausePlayBack</a>	Interface for pause or restoring playback
<a href="#">CLIENT_FastPlayBack</a>	Interface for fast play. Increasing frame rate by 1x
<a href="#">CLIENT_SlowPlayBack</a>	Interface for slow play. Decreasing frame rate by 1x
<a href="#">CLIENT_NormalPlayBack</a>	Interface for restoring normal play speed
<a href="#">CLIENT_SeekPlayBack</a>	Interface for positioning record playback start point
<a href="#">CLIENT_Logout</a>	Interface for logout
<a href="#">CLIENT_GetLastError</a>	Interface for getting error code after failed calling interface

## 2.4.3 Process Instruction

According to the different decoding method selected by users, record playback have the following two methods:

- SDK decoding and playing back

Firstly user inputs start time, end time and valid window handle of record, then SDK will call corresponding decoding library to analyze stream and show the video in display window.

- The third-party decoding and playing back

Firstly user inputs start time, end time and valid window handle(window handle is set to NULL in this method) and valid playback stream callback function of record. After SDK receives playback stream data, the data is called back to user for saving by playback stream callback function. After leaving callback function, user calls a third-party library to analyze and display the saved stream data.

#### **2.4.3.1 SDK Decoding and Playing Back**

SDK Decoding and Playing back process flow sheet is shown below in Figure 2-5.



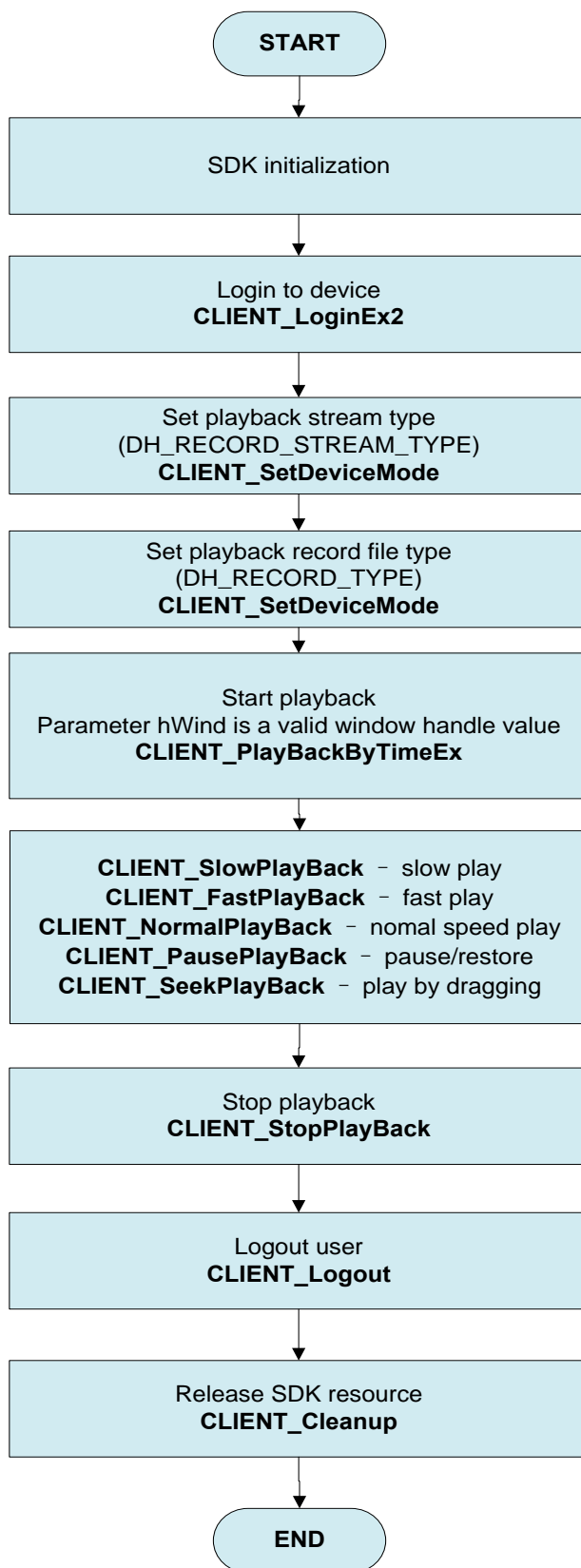


Figure 2-5 SDK Decoding and Playing back process flow sheet

**Process description:**

1. SDK initialization.
2. After successful initialization, call CLIENT\_LoginEx2 to login device.
3. After successful login, call CLIENT\_SetDeviceMode twice to separately set playback stream type and playback record file type.
4. Call CLIENT\_PlayBackByTimeEx to start playback, parameter hWnd is set to valid window handle value.
5. During playback, user can call CLIENT\_SlowPlayBack to slowly play, CLIENT\_FastPlayBack to fast play, CLIENT\_NormalPlayBack to play at normal speed, CLIENT\_PausePlayBack to pause/store play, CLIENT\_SeekPlayBack to play by dragging.
6. After playback is done, call CLIENT\_StopPlayBack to stop playback.
7. After realizing needed business, call CLIENT\_Logout to logout device.
8. Last but not the least, call CLIENT\_Cleanup to release SDK resource.

**2.4.3.2 The Third-party Decoding and Playing Back**

The third-party library decoding and playing back flow sheet is shown in Figure 2-6.

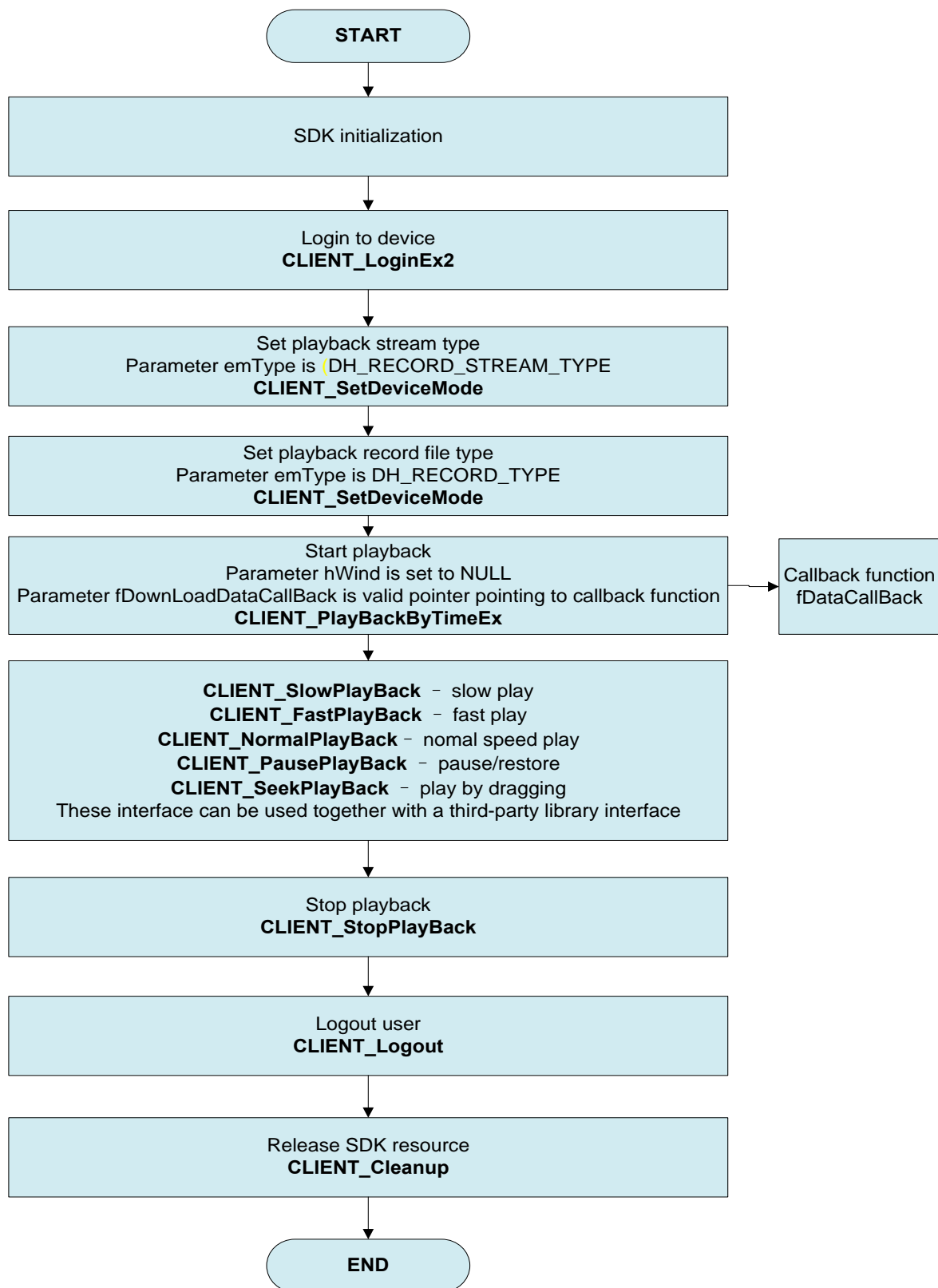


Figure 2-6 The third-party library decoding and playing back flow sheet

**Process description:**

1. SDK initialization.
2. After successful initialization, call CLIENT\_LoginEx2 to login device.
3. After successful login, call CLIENT\_SetDeviceMode twice to separately set playback stream type and playback record file type.
4. After successful login, call CLIENT\_PlayBackByTimeEx to start playback, parameter hWnd is set to NULL, parameter fDownloadDataCallBack is a valid pointer pointing to a callback function.
5. After SDK receives playback stream data, the data is called back to user for saving by playback stream callback function fDownloadDataCallBack. After leaving callback function, user calls a third-party library to analyze and display the saved stream data.
6. During playback, user can call CLIENT\_SlowPlayBack to slowly play, CLIENT\_FastPlayBack to fast play, CLIENT\_NormalPlayBack to play at normal speed, CLIENT\_PausePlayBack to pause/store play, CLIENT\_SeekPlayBack to play by dragging together with the third-party interfaces.
7. After playback is done, call CLIENT\_StopPlayBack to stop playback.
8. After realizing needed business, call CLIENT\_Logout to logout device.
9. Last but not the least, call CLIENT\_Cleanup to release SDK resource.

## 2.4.4 Example Code

### 2.4.4.1 SDK Decoding and Playing Back

```
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"
#pragma comment(lib, "dhnetsdk.lib")

extern "C" HWND WINAPI GetConsoleWindow();

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lPlayHandle = 0L;
static char g_szDevIp[32] = "172.11.1.13";
static WORD g_nPort = 37777;
static char g_szUserName[64] = "admin";
```

```
static char g_szPasswd[64] = "admin";

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisconnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);
//*****
void InitTest()
{
    // SDK Initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    //Get the SDK version info
    // This operation is optional.
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // set CLIENT_SetAutoReconnect interface. Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally
    //This operation is optional, but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

    // Set device connection timeout value and trial times.
```

```
//This operation is optional.
int nWaitTime = 5000;    // timeout value is 5 seconds
int nTryTimes = 3;       //if timeout,it will try to log in three times.
CLIENT_SetConnectTime(nWaitTime, nTryTimes);

//Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
//This operation is optional.
NET_PARAM stuNetParm = {0};
stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_ILoginHandle)
{
    //Login device
    g_ILoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if(0 == g_ILoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhnetsdk.h.
        It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        //Example: #define NET_NOT_SUPPORTED_EC(23)
        //Now SDK does not support this function, error code is 0x80000017, Decimal
        number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
    }
    else
    {
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
    }
    // The first time logging to device, some data is needed to be initialized to enable
    normal business function. It is recommended to wait for a while after login, the waiting time varies
    by devices.
    Sleep(1000);
    printf("\n");
}
}
```

```
void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_ILoginHandle)
    {
        return;
    }

    // record playback function
    // get console window handle
    HWND hWnd = GetConsoleWindow();

    // set playback stream type
    int nStreamType = 0; // 0-main/sub stream,1-main stream,2-sub stream
    CLIENT_SetDeviceMode(g_ILoginHandle, DH_RECORD_STREAM_TYPE,
&nStreamType);

    // set playback record file type
    NET_RECORD_TYPE emFileType = NET_RECORD_TYPE_ALL; // all records
    CLIENT_SetDeviceMode(g_ILoginHandle, DH_RECORD_TYPE, &emFileType);

    //start record playback
    int nChannelID = 0; // channel NO.
    NET_TIME stuStartTime = {0};
    stuStartTime.dwYear = 2015;
    stuStartTime.dwMonth = 11;
    stuStartTime.dwDay = 20;

    NET_TIME stuStopTime = {0};
    stuStopTime.dwYear = 2015;
    stuStopTime.dwMonth = 11;
    stuStopTime.dwDay = 21;

    g_IPlayHandle = CLIENT_PlayBackByTimeEx(g_ILoginHandle, nChannelID,
&stuStartTime, &stuStopTime, hWnd, NULL, NULL, NULL, NULL);
    if (0 == g_IPlayHandle)
    {
        printf("CLIENT_PlayBackByTimeEx: failed! Error code: %x.\n",
CLIENT_GetLastError());
    }
}
```

```
}

// user can control palyback according to needs
// due to it is a console demo, we cannot show user record playback and playback control
at the same time, here are some examples for reference.

// CLIENT_SlowPlayBack to slowly play
/* example code
if (FALSE == CLIENT_SlowPlayBack (g_IPlayHandle))
{
    printf("CLIENT_SlowPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
*/

// CLIENT_FastPlayBack to fast play
/* example code
if (FALSE == CLIENT_FastPlayBack (g_IPlayHandle))
{
    printf("CLIENT_FastPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
*/

// CLIENT_NormalPlayBack to play at normal speed
/* example code
if (FALSE == CLIENT_NormalPlayBack (g_IPlayHandle))
{
    printf("CLIENT_NormalPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
*/

// CLIENT_PausePlayBack to pause/restore play
/* example code
if (FALSE == CLIENT_PausePlayBack (g_IPlayHandle, TRUE))
{
    printf("CLIENT_PausePlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
*/

// CLIENT_SeekPlayBack to play by dragging
```



```
/* example code
int nOffsetSeconds = 2 * 60 * 60; // drag to 2*60*60s after stuStartTime to start play
if (FALSE == CLIENT_SeekPlayBack (g_IPlayHandle, nOffsetSeconds, 0))
{
    printf("CLIENT_SeekPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
*/

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // close playback
    if (0 != g_IPlayHandle)
    {
        if (FALSE == CLIENT_StopPlayBack(g_IPlayHandle))
        {
            printf("CLIENT_StopPlayBack Failed, g_IRealHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
        }
        else
        {
            g_IPlayHandle = 0;
        }
    }
    // logout device
    if (0 != gILoginHandle)
    {
        if(FALSE == CLIENT_Logout(gILoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            gILoginHandle = 0;
        }
    }
    // cleanup initialization resource
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
    }
}
```

```
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//*****
// commonly used callback set definition

void CALLBACK DisConnectFunc(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

## 2.4.4.2 The Third-party Decoding and Playing Back

```
#include <windows.h>
#include <stdio.h>
#include "dhnetSDK.h"
#pragma comment(lib, "dhnetSDK.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_ILoginHandle = 0L;
static LLONG g_IPlayHandle = 0L;
static char g_szDevIp[32] = "172.11.1.6";
static WORD g_nPort = 37777;
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

// playback progress callback function
// It's not recommended to call SDK interfaces in this callback function
// set this callback function in CLIENT_PlayBackByTimeEx. when playback data is received,
SDK will call this function.
void CALLBACK DownLoadPosCallBack(LLONG IPlayHandle, DWORD dwTotalSize, DWORD
dwDownloadSize, LDWORD dwUser);

// playback data callback function
// It's not recommended to call SDK interfaces in this callback function
//when parameter hWnd is NULL, the function return 0: means this callback failed, next
callback will return the same data. 1:means this callback is successful, next callback will return
follow-up data.
```

// when parameter hWnd isn't NULL, it will be treated as callback successful no matter what the return value is, and next time callback will return follow-up data

// set this callback function in CLIENT\_PlayBackByTimeEx. when playback data is received, SDK will call this function.

```
int CALLBACK DataCallBack(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer,
DWORD dwBufSize, LDWORD dwUser);
```

```
//*****
```

```
void InitTest()
```

```
{
```

```
    // SDK Initialization
```

```
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
```

```
    if (FALSE == g_bNetSDKInitFlag)
```

```
    {
```

```
        printf("Initialize client SDK fail; \n");
```

```
        return;
```

```
    }
```

```
    else
```

```
    {
```

```
        printf("Initialize client SDK done; \n");
```

```
    }
```

```
    //Get the SDK version info
```

```
    // This operation is optional.
```

```
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
```

```
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);
```

// set CLIENT\_SetAutoReconnect interface. Once HaveReConnect is set, when device gets offline, SDK will reconnect internally

```
    //This operation is optional, but recommended.
```

```
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);
```

```
    // Set device connection timeout value and trial times.
```

```
    //This operation is optional.
```

```
    int nWaitTime = 5000;    // timeout value is 5 seconds
```

```
    int nTryTimes = 3;        //if timeout, it will try to log in three times.
```

```
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);
```

//Set more network parameter, nWaittime and nConnectTryNum in NET\_PARAM has the same meaning with the timeout time and trial time set in interface CLIENT\_SetConnectTime.

```
    //This operation is optional.
```

```
    NET_PARAM stuNetParm = {0};
```

```
    stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

    NET_DEVICEINFO_Ex stDevInfoEx = {0};
    int nError = 0;
    while(0 == g_ILoginHandle)
    {
        //Login device
        g_ILoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

        if(0 == g_ILoginHandle)
        {
            // according to error code, you can find corresponding explanation in dhnetsdk.h.
            It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion
            //Example: #define NET_NOT_SUPPORTED_EC(23)
            //Now SDK does not support this function, error code is 0x80000017, Decimal
            number 23 is hexadecimal 0x17.
            printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
        }
        // The first time logging to device, some data is needed to be initialized to enable
        normal business function. It is recommended to wait for a while after login, the waiting time varies
        by devices.
        Sleep(1000);
        printf("\n");
    }

    void RunTest()
    {
        if (FALSE == g_bNetSDKInitFlag)
        {
            return;
        }

        if (0 == g_ILoginHandle)
        {
            return;
        }
    }
}
```

```
}

// record playback function

// set playback stream type
int nStreamType = 0; // 0-main/sub stream,1-main stream,2-sub stream
CLIENT_SetDeviceMode(g_ILoginHandle, DH_RECORD_STREAM_TYPE,
&nStreamType);

// set playback record file type
NET_RECORD_TYPE emFileType = NET_RECORD_TYPE_ALL; // all records
CLIENT_SetDeviceMode(g_ILoginHandle, DH_RECORD_TYPE, &emFileType);

//start record playback
int nChannelID = 0; // channel NO.
NET_TIME stuStartTime = {0};
stuStartTime.dwYear = 2015;
stuStartTime.dwMonth = 11;
stuStartTime.dwDay = 20;

NET_TIME stuStopTime = {0};
stuStopTime.dwYear = 2015;
stuStopTime.dwMonth = 11;
stuStopTime.dwDay = 21;

// parameter hWnd need to be NULL
// parameter fDownloadDataCallBack need to be a valid pointer pointing to a callback
function
g_IPlayHandle = CLIENT_PlayBackByTimeEx(g_ILoginHandle, nChannelID,
&stuStartTime, &stuStopTime, NULL, &DownloadPosCallBack, NULL, &DataCallBack, NULL);
if (g_IPlayHandle == 0)
{
    printf("CLIENT_PlayBackByTimeEx: failed! Error code: %x.\n",
CLIENT_GetLastError());
}

// user control playback according to need
// due to it is third-party library decoding, when a user calls SDK playback control interface,
he needs to call control interface of third-party library at the same time
// due to it is a console demo, we cannot show user record playback and playback control
at the same time, here are some examples for reference.
// CLIENT_SlowPlayBack to slowly play
/* example code
```

```
if (FALSE == CLIENT_SlowPlayBack (g_IPlayHandle))
{
    printf("CLIENT_SlowPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
//the third-party interface call

*/

// CLIENT_FastPlayBack to fast play
/* example code
if (FALSE == CLIENT_FastPlayBack (g_IPlayHandle))
{
    printf("CLIENT_FastPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
// the third-party interface call

*/

// CLIENT_NormalPlayBack to play at normal speed
/* example code
if (FALSE == CLIENT_NormalPlayBack (g_IPlayHandle))
{
    printf("CLIENT_NormalPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
//the third-party interface call

*/

// CLIENT_PausePlayBack to pause/restore play
/* example code
if (FALSE == CLIENT_PausePlayBack (g_IPlayHandle, TRUE))
{
    printf("CLIENT_PausePlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
// the third-party interface call

*/

// CLIENT_SeekPlayBack to play by dragging
```

```
/* example code
int nOffsetSeconds = 2 * 60 * 60; // drag to 2*60*60s after stuStartTime to start play
if (FALSE == CLIENT_SeekPlayBack (g_IPlayHandle, nOffsetSeconds, 0))
{
    printf("CLIENT_SeekPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
}
//the third-party interface call

*/

}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // close playback
    if (0 != g_IPlayHandle)
    {
        if (FALSE == CLIENT_StopPlayBack(g_IPlayHandle))
        {
            printf("CLIENT_StopPlayBack Failed, g_IRealHandle[%x]!Last Error[%x]\n" ,
g_IPlayHandle, CLIENT_GetLastError());
        }
        else
        {
            g_IPlayHandle = 0;
        }
    }
    // logout device
    if (0 != gILoginHandle)
    {
        if(FALSE == CLIENT_Logout(gILoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            gILoginHandle = 0;
        }
    }
    // cleanup initialization resource
    if (TRUE == g_bNetSDKInitFlag)
```



```
{
    CLIENT_Cleanup();
    g_bNetSDKInitFlag = FALSE;
}
return;
}

int main()
{
    InitTest();

    RunTest();

    EndTest();

    return 0;
}

//*****
// commonly used callback set definition

void CALLBACK DisConnectFunc(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
}
```

```
    }  
    printf("nDVRPort[%d]\n", nDVRPort);  
    printf("dwUser[%p]\n", dwUser);  
    printf("\n");  
}
```

```
void CALLBACK DownLoadPosCallBack(LLONG IPlayHandle, DWORD dwTotalSize, DWORD  
dwDownLoadSize, LDWORD dwUser)
```

```
{  
    // If different playback/download uses the same process callback function, we can judge by  
parameter IPlayHandle
```

```
    if (IPlayHandle == g_IPlayHandle)  
    {  
        printf("IPlayHandle[%p]\n", IPlayHandle);  
        printf("dwTotalSize[%d]\n", dwTotalSize);  
        printf("dwDownLoadSize[%d]\n", dwDownLoadSize);  
        printf("dwUser[%p]\n", dwUser);  
        printf("\n");  
    }  
}
```

```
int CALLBACK DataCallBack(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer,  
DWORD dwBufSize, LDWORD dwUser)
```

```
{  
    int nRet = 0;  
    printf("call DataCallBack\n");  
    // If different playback/download uses the same callback function, we can judge by  
parameter IRealHandle  
    if(IRealHandle == g_IPlayHandle)  
    {  
        BOOL bSuccess = TRUE;  
        // the following printing will cause frequently screen refresh, please be careful  
        printf("IPlayHandle[%p]\n", IRealHandle);  
        printf("dwDataType[%d]\n", dwDataType);  
        printf("pBuffer[%p]\n", pBuffer);  
        printf("dwBufSize[%d]\n", dwBufSize);  
        printf("dwUser[%p]\n", dwUser);  
        printf("\n");  
        switch(dwDataType)  
        {  
            case 0:  
                //Original data  
                // users can save stream data here for further process such as decoding and
```

transferring after getting out of callback function

```
        nRet = 1;

        break;
    case 1:
        //Standard video data

        break;
    case 2:
        //yuv data

        break;
    case 3:
        //pcm audio data

        break;
    case 4:
        //Original audio data

        break;
    default:
        break;
    }
}
return nRet;
}
```

## 2.5 Record Download

### 2.5.1 Intro

Video monitoring system is widely used in safe city, airport, subway, bank, factory and etc. When something happened, it's needed to download records to leaders、police and media as evidence for further use. Therefore, record download is a commonly used application module.

Record download process is that users get the records from storage device and save them to local. It allows user to download records in currently selected channel, and export the records to local hard disk or U disk.

## Record Download Methods

Record download have two methods: download by file and download by time, please refer to Ch 2.5.3.

## 2.5.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for SDK initialization
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources
<a href="#">CLIENT_LoginEx2</a>	Extensive interface 2 for Sync login
<a href="#">CLIENT_SetDeviceMode</a>	Interface for setting work mode of device audio intercom, playback and right
<a href="#">CLIENT_QueryRecordFile</a>	Interface for searching all files in a specified time period
<a href="#">CLIENT_FindFile</a>	Interface for opening record search handle
<a href="#">CLIENT_FindNextFile</a>	Interface for searching record file
<a href="#">CLIENT_FindClose</a>	Interface for closing record search handle
<a href="#">CLIENT_DownloadByRecordFileEx</a>	Extensive interface for downloading record by file
<a href="#">CLIENT_DownloadByTimeEx</a>	Extensive interface for downloading record by time
<a href="#">CLIENT_GetDownloadPos</a>	Interface for searching record download process
<a href="#">CLIENT_StopDownload</a>	Interface for stopping record download
<a href="#">CLIENT_Logout</a>	Interface for logout
<a href="#">CLIENT_GetLastError</a>	Interface for getting error code after failed calling interface

## 2.5.3 Process Instruction

Record download includes the following two methods:

- Download by file

Users need to point the downloaded record file's information and SDK can download the specified file and save it to a specified file. At the same time, user can also provide a callback function pointer, so that SDK send the downloaded file info to users for further use by callback

function.

- Download by time

User will need to point the start time and end time of the download file, SDK can download the specified file in a specified time period and save it to a specified file. At the same time, user can also provide a callback function pointer, so that SDK send the downloaded file info to users for further use by callback function.

### **2.5.3.1 Download by File**

Download by file process flow sheet is shown below in Figure 2-7.

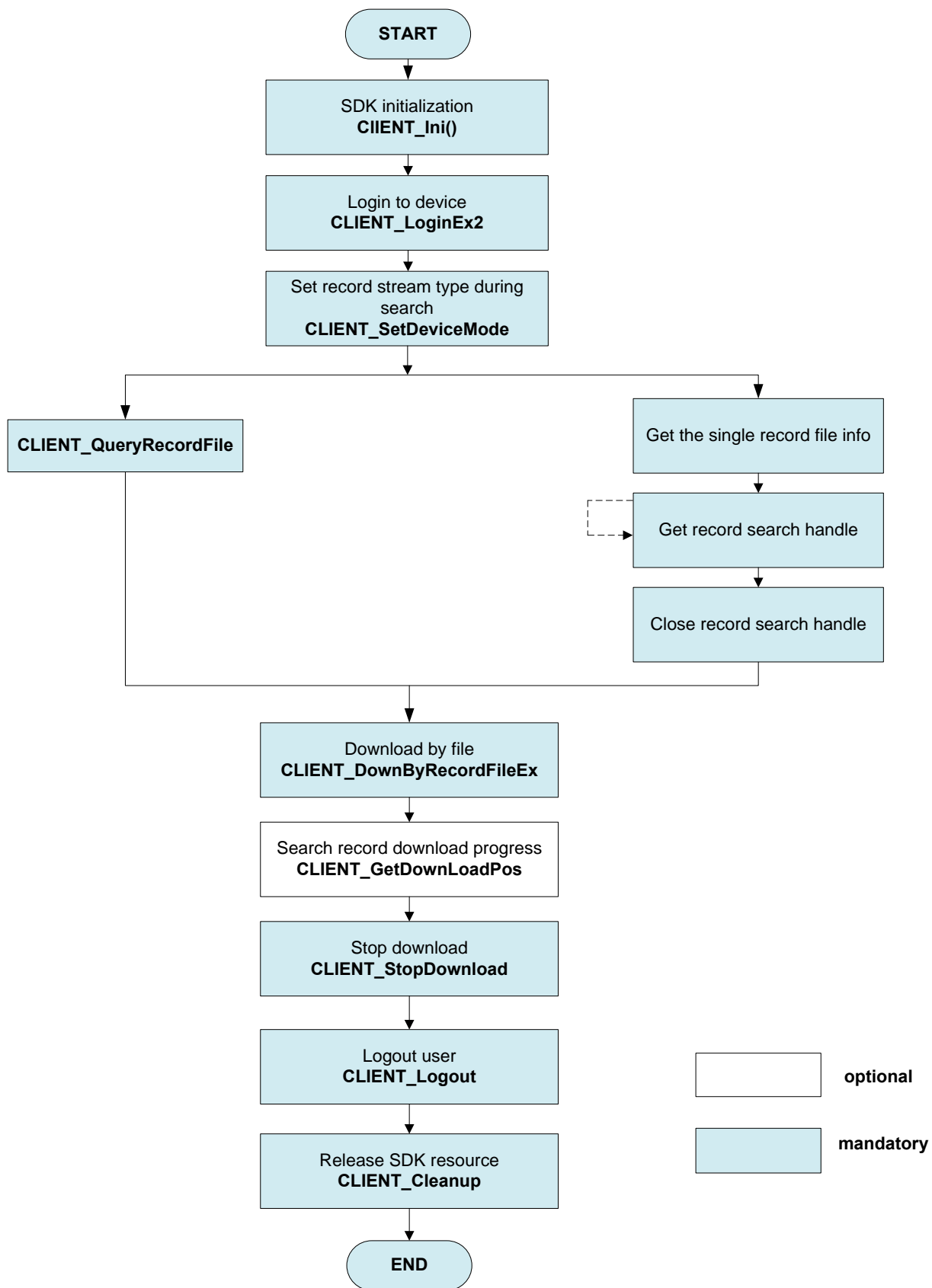


Figure 2-7 Download by file process flow sheet

**Process description:**

1. SDK initialization.
2. After successful initialization, call CLIENT\_LoginEx2 to login device.
3. Set record stream type in Call CLIENT\_SetDeviceMode. Parameter EmType is DH\_RECORD\_STREAM\_TYPE which is recommended to set to "0-main/sub stream", or some devices will not return a result. If you only need main stream record, you can filter out sub stream record info in result. Please refer to Appendix 2 "EM\_USEDEV\_MODE" enumeration note.
4. You can search record file with the following two methods:
  - Call CLIENT\_FindFile to get record search handle, and then call CLIENT\_FindNextFile to progressively get record file info in cycle, finally call CLIENT\_FindClose to close record search handle.
  - Call CLIENT\_QueryRecordFile to get all record files info in a specified period.
5. After you get record file info, call CLIENT\_DownloadByRecordFileEx to download record file. At least one of the input parameters sSavedFileName and fDownloadDataCallBack is valid.
6. During the download, call CLIENT\_GetDownloadPos to search record download progress as needed.
7. After download is finished, call CLIENT\_StopDownload to stop download.
8. After business is finished, call CLIENT\_Logout to logout device.
9. Last but not the least, call CLIENT\_Cleanup to release SDK resource.

**2.5.3.2 Download by Time**

Download by time process flow sheet is shown in Figure 2-8.

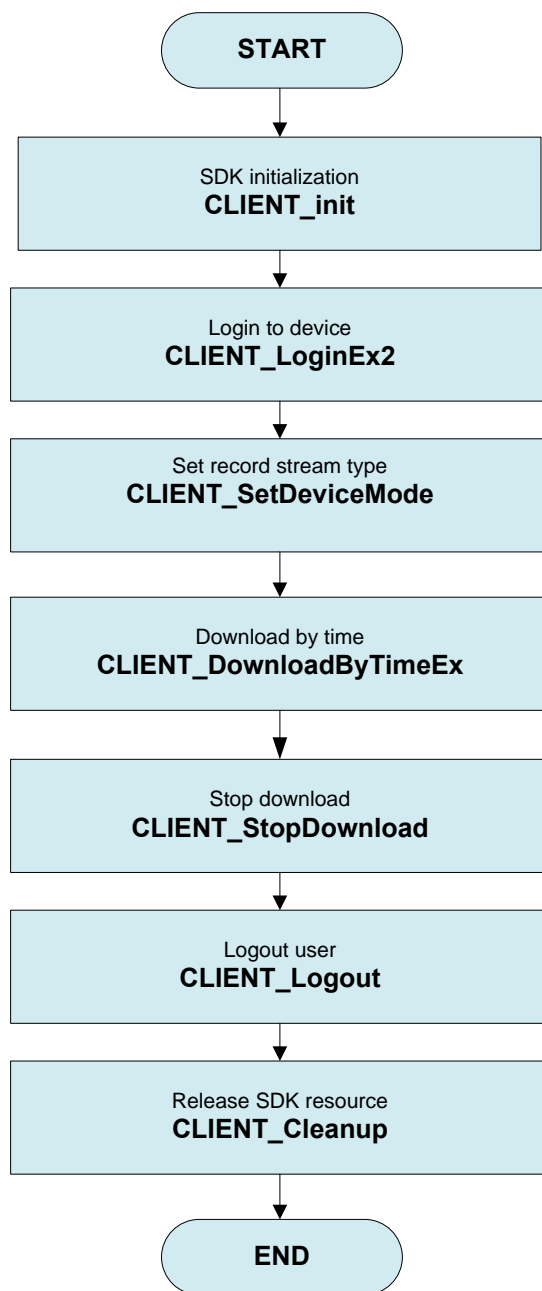


Figure 2-8 Download by time process flow sheet

**Process description:**

1. SDK initialization
2. After successful initialization, call CLIENT\_LoginEx2 to login device.
3. Set record stream type in Call CLIENT\_SetDeviceMode. Parameter EmType is DH\_RECORD\_STREAM\_TYPE, Please refer to Appendix 2 “EM\_USEDEV\_MODE” enumeration note.



4. Call CLIENT\_DownloadByTimeEx to start to download by time. At least one input parameter sSavedFileName and fDownloadDataCallBack is valid.
5. After download is finished, call CLIENT\_StopDownload to stop download. We can stop download when download is finished or when download is still in progress as needed.
6. After business is finished, call CLIENT\_Logout to logout device.
7. Last but not the least, call CLIENT\_Cleanup to release SDK resource.

## 2.5.4 Example Code

### 2.5.4.1 Download by File

```
#include <windows.h>
#include <stdio.h>
#include <vector>
#include "dhnetsdk.h"

#pragma comment(lib, "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lDownloadHandle = 0L;
static char g_szDevIp[32] = "172.11.1.30";
static WORD g_nPort = 37777;
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
static const int g_nMaxRecordFileCount = 5000;
//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
```

```
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

// playback/download progress callback function
// It's not recommended to call SDK interfaces in this callback function
// dwDownloadSize: -1 means playback/download finished, -2 means failed to write file, other
value means valid data
// Set this callback function in CLIENT_DownloadByRecordFileEx. When SDK receives
playback/downloaded data, SDK will call this function.
void CALLBACK DownLoadPosCallBack(LLONG IPlayHandle, DWORD dwTotalSize, DWORD
dwDownloadSize, LDWORD dwUser);

// playback/download data callback function
// It's not recommended to call SDK interfaces in this callback function
// playback: return value: 0 means this playback failed, next callback will return the same data,
1 means this callback successful, next callback will return the following data
// Download: no matter what return from the callback function, it will be treated as callback is
successful, next callback will return the following data
// Set this callback function in CLIENT_DownloadByRecordFileEx. When SDK receives
playback/downloaded data, SDK will call this function.
int CALLBACK DataCallBack(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer,
DWORD dwBufSize, LDWORD dwUser);

//*****
void InitTest()
{
    // SDK initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // get SDK version info
    // this operation is optional
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);
}
```

```
// set CLIENT_SetAutoReconnect interface.Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally.
// this operation is optional, but recommended
CLIENT_SetAutoReconnect(&HaveReConnect, 0);

// set connecting device timeout time and attempts
// this operation is optional
int nWaitTime = 5000;    // timeout time is set to 5s
int nTryTimes = 3;       // if timeout, you have 3 attempts
CLIENT_SetConnectTime(nWaitTime, nTryTimes);

//Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
//This operation is optional.
NET_PARAM stuNetParm = {0};
stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_LoginHandle)
{
    // login device
    g_LoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if (0 == g_LoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhnetsdk.h.
        // It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        // example:
        // #define NET_NOT_SUPPORTED_EC(23)
        // current SDK does not support the function, corresponding error code is
        // 0x80000017. Decimal number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
    }
    else
    {
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
    }
}
// The first time logging to device, some data is needed to be initialized to enable
```

normal business function. It is recommended to wait for a while after login, the waiting time varies by devices.

```
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_ILoginHandle)
    {
        return;
    }

    // record file search
    // set record stream type
    int nStreamType = 0; // 0-main stream/sub stream,1-main stream,2-sub stream
    CLIENT_SetDeviceMode(g_ILoginHandle, DH_RECORD_STREAM_TYPE,
&nStreamType);
    // there are two methods to search files: 1.take all record files in the specified time period
once; 2, take all records in the specified time period in several times.
    // here is the second method, the first method can refer to CLIENT_QueryRecordFile
interface
    int nChannelID = 0; // channel no.
    NET_TIME stuStartTime = {0};
    stuStartTime.dwYear = 2015;
    stuStartTime.dwMonth = 9;
    stuStartTime.dwDay = 20;

    NET_TIME stuStopTime = {0};
    stuStopTime.dwYear = 2015;
    stuStopTime.dwMonth = 9;
    stuStopTime.dwDay = 30;
    int IFindHandle = CLIENT_FindFile(g_ILoginHandle, nChannelID, 0, NULL, &stuStartTime,
&stuStopTime, FALSE, 5000);
    if (0 == IFindHandle)
    {
        printf("CLIENT_FindFile Failed!Last Error[%x]\n",CLIENT_GetLastError());
        return;
    }
}
```

```
}
// demo
std::vector<NET_RECORDFILE_INFO> bufFileInfo(g_nMaxRecordFileCount);

for (int nFileIndex = 0; nFileIndex < g_nMaxRecordFileCount; ++nFileIndex)
{
    int result = CLIENT_FindNextFile(IFindHandle, &bufFileInfo[nFileIndex]);
    if (0 == result)// finish taking record files info    {
        break;
    }
    else if (1 != result)// parameter error
    {
        printf("CLIENT_FindNextFile Failed!Last Error[%x]\n",CLIENT_GetLastError());
        break;
    }
}

//stop search
if(0 != IFindHandle)
{
    CLIENT_FindClose(IFindHandle);
}
// set the first file queried as download file
NET_RECORDFILE_INFO stuNetFileInfo;
if (nFileIndex > 0)
{
    memcpy(&stuNetFileInfo, (void *)&bufFileInfo[0], sizeof(stuNetFileInfo));
}
else
{
    printf("no record, return\n");
    return;
}

// record file download
// start record download
// at least one of the two parameters sSavedFileName and fDownloadDataCallBack is
valid
// the downloaded data is save to parameter sSavedFileName or save to callback function
as needed
g_IDownloadHandle = CLIENT_DownloadByRecordFileEx(gILoginHandle,
&stuNetFileInfo, "test.dav", DownloadPosCallBack, NULL, DataCallBack, NULL);
if (0 == g_IDownloadHandle)
```

```
{
    printf("CLIENT_DownloadByRecordFileEx: failed! Error code: %x.\n",
CLIENT_GetLastError());
}
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // stop download,we can call this interface after download ends or during download.
    if (0 != g_IDownloadHandle)
    {
        if (FALSE == CLIENT_StopDownload(g_IDownloadHandle))
        {
            printf("CLIENT_StopDownload Failed, g_IDownloadHandle[%x]!Last
Error[%x]\n", g_IDownloadHandle, CLIENT_GetLastError());
        }
        else
        {
            g_IDownloadHandle = 0;
        }
    }
    // logout device
    if (0 != gILoginHandle)
    {
        if(FALSE == CLIENT_Logout(gILoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            gILoginHandle = 0;
        }
    }
    // cleanup initialization resource
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }

    return;
}
```

```
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//*****
// commonly used callback set definition
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK DownLoadPosCallBack(LLONG IPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, LDWORD dwUser)
{

```

```
// If different playback/download uses the same process callback function, we can judge by
parameter 1PlayHandle
if (IPlayHandle == g_IDownloadHandle)
{
    printf("IPlayHandle[%p]\n", IPlayHandle);
    printf("dwTotalSize[%d]\n", dwTotalSize);
    printf("dwDownLoadSize[%d]\n", dwDownLoadSize);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
}

int CALLBACK DataCallBack(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer,
DWORD dwBufSize, LDWORD dwUser)
{
    int nRet = 0;
    printf("call DataCallBack\n");
    // If different playback/download uses the same data callback function, we can judge by
parameter 1RealHandle
if(IRealHandle == g_IDownloadHandle)
{
    printf("IPlayHandle[%p]\n", IRealHandle);
    printf("dwDataType[%d]\n", dwDataType);
    printf("pBuffer[%p]\n", pBuffer);
    printf("dwBufSize[%d]\n", dwBufSize);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
    switch(dwDataType)
    {
    case 0:
        //Original data
        // users can save stream data here for further process such as decoding and
transferring after getting out of callback function
        nRet = 1;

        break;
    case 1:
        //Standard video data

        break;
    case 2:
        //yuv data
```



```
        break;
    case 3:
        //pcm audio data

        break;
    case 4:
        //Original audio data

        break;
    default:
        break;
    }
}
return nRet;
}
```

#### 2.5.4.2 Download by Time

```
#include <windows.h>
#include <stdio.h>
#include "dhnetSDK.h"

#pragma comment(lib, "dhnetSDK.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lDownloadHandle = 0L;
static char g_szDevIp[32] = "172.11.1.221";
static WORD g_nPort = 37777;
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
```

```
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

// playback by time progress callback function
// It's not recommended to call SDK interfaces in this callback function
// dwDownloadSize: -1 means playback/download finished, -2 means failed to write file, other
value means valid data
// Set this callback function in CLIENT_DownloadByTimeEx. When SDK receives
playback/downloaded data, SDK will call this function.
void CALLBACK TimeDownloadPosCallBack(LLONG IPlayHandle, DWORD dwTotalSize,
DWORD dwDownloadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD
dwUser);

// playback/download data callback function
// It's not recommended to call SDK interfaces in this callback function
// playback: return value: 0 means this playback failed, next callback will return the same data,
1 means this callback successful, next callback will return the following data
// Download: no matter what return from the callback function, it will be treated as callback is
successful, next callback will return the following data
// Set this callback function in CLIENT_DownloadByTimeEx. When SDK receives
playback/downloaded data, SDK will call this function.
int CALLBACK DataCallBack(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer,
DWORD dwBufSize, LDWORD dwUser);

//*****
void InitTest()
{
    // SDK initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // get SDK version info
    // this operation is optional
```

```
DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
printf("NetSDK version is [%d]\n", dwNetSdkVersion);

// set CLIENT_SetAutoReconnect interface. Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally
// this operation is optional, but recommended
CLIENT_SetAutoReconnect(&HaveReConnect, 0);

// set connecting device timeout time and attempts
// this operation is optional
int nWaitTime = 5000;    // timeout time is set to 5s
int nTryTimes = 3;       // if timeout, you have 3 attempts
CLIENT_SetConnectTime(nWaitTime, nTryTimes);

//Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
//This operation is optional.
NET_PARAM stuNetParm = {0};
stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_ILLoginHandle)
{
    // login device
    g_ILLoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if (0 == g_ILLoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhnetsdk.h.
        // It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        // example:
        // #define NET_NOT_SUPPORTED_EC(23)
        // current SDK does not support the function, corresponding error code is
        // 0x80000017. Decimal number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
    }
    else
    {

```

```
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
    }
    // The first time logging to device, some data is needed to be initialized to enable
normal business function. It is recommended to wait for a while after login, the waiting time varies
by devices.
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_ILoginHandle)
    {
        return;
    }

    // record file search
    // set record stream type
    int nStreamType = 0; // 0-main/sub stream, 1-main stream, 2-sub stream
    CLIENT_SetDeviceMode(g_ILoginHandle, DH_RECORD_STREAM_TYPE,
&nStreamType);

    int nChannelID = 0; // channel no.
    NET_TIME stuStartTime = {0};
    stuStartTime.dwYear = 2015;
    stuStartTime.dwMonth = 9;
    stuStartTime.dwDay = 17;

    NET_TIME stuStopTime = {0};
    stuStopTime.dwYear = 2015;
    stuStopTime.dwMonth = 9;
    stuStopTime.dwDay = 18;

    // start record download
    // at least one of the two parameters sSavedFileName and fDownloadDataCallBack is
valid
    g_IDownloadHandle = CLIENT_DownloadByTimeEx(g_ILoginHandle, nChannelID,
EM_RECORD_TYPE_ALL, &stuStartTime, &stuStopTime, "test.dav", TimeDownLoadPosCallBack,
```

```
NULL, DataCallBack, NULL);
    if (g_IDownloadHandle == 0)
    {
        printf("CLIENT_DownloadByTimeEx: failed! Error code: %x.\n",
CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // stop download,we can call this interface after download end or during download.
    if (0 != g_IDownloadHandle)
    {
        if (FALSE == CLIENT_StopDownload(g_IDownloadHandle))
        {
            printf("CLIENT_StopDownload Failed, g_IDownloadHandle[%x]!Last
Error[%x]\n", g_IDownloadHandle, CLIENT_GetLastError());
        }
        else
        {
            g_IDownloadHandle = 0;
        }
    }
    // logout device
    if (0 != gILoginHandle)
    {
        if(FALSE == CLIENT_Logout(gILoginHandle))
        {
            printf("CLIENT_Logout Failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            gILoginHandle = 0;
        }
    }
    // cleanup initialization resource
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}
```

```
        return;
    }

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//*****
// commonly used callback set definition
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK TimeDownLoadPosCallBack(LLONG IPlayHandle, DWORD dwTotalSize,
```

```
DWORD dwDownloadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD
dwUser)
{
    // If different playback/download uses the same process callback function, we can judge by
parameter 1PlayHandle
    if (IPlayHandle == g_IDownloadHandle)
    {
        printf("IPlayHandle[%p]\n", IPlayHandle);
        printf("dwTotalSize[%d]\n", dwTotalSize);
        printf("dwDownloadSize[%d]\n", dwDownloadSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
    }
}

int CALLBACK DataCallBack(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer,
DWORD dwBufSize, LDWORD dwUser)
{
    int nRet = 0;
    printf("call DataCallBack\n");
    // If different playback/download uses the same data callback function, we can judge by
parameter 1RealHandle
    if(IRealHandle == g_IDownloadHandle)
    {
        printf("IPlayHandle[%p]\n", IRealHandle);
        printf("dwDataType[%d]\n", dwDataType);
        printf("pBuffer[%p]\n", pBuffer);
        printf("dwBufSize[%d]\n", dwBufSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
        switch(dwDataType)
        {
            case 0:
                //Original data
                // users can save stream data here for further process such as decoding and
transferring after getting out of callback function
                nRet = 1;

                break;
            case 1:
                //Standard video data

                break;
```

```
case 2:
    //yuv data

    break;
case 3:
    //pcm audio data

    break;
case 4:
    //Original audio data

    break;
default:
    break;
}
}
return nRet;
}
```

## 2.6 PTZ Control

### 2.6.1 Intro

PTZ is a mechanical platform which carries camera device and protective cover can remote monitor and control in all directions. PTZ is made of two motors and capable for horizontal and vertical motion, therefore it can provide omnibearing and multi-angle viewing for video camera.

PTZ control is an important part of a surveillance system. Users have different demands for surveillance in different application scene. For example, users may want to track the surveillance screen in a normal application scene. Users can control PTZ device via SDK, such as move up/down/left/right, focus, zoom in/out, point-to-point tour and 3D positioning.

### 2.6.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for SDK initialization
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources
<a href="#">CLIENT_LoginEx2</a>	Extensive interface 2 for sync login



Interface	Interface Description
<a href="#">CLIENT_ParseData</a>	Interface for analyzing the obtained config info
<a href="#">CLIENT_DHPTZControlEx2</a>	Extensive interface for private PTZ control
<a href="#">CLIENT_QueryNewSystemInfo</a>	Interface for obtaining new system capacity set
<a href="#">CLIENT_Logout</a>	Interface for logout device
<a href="#">CLIENT_GetLastError</a>	Interface for getting error code after failed calling interface

### 2.6.3 Process Instruction

PTZ controller process flow sheet is shown in Figure 2-9.

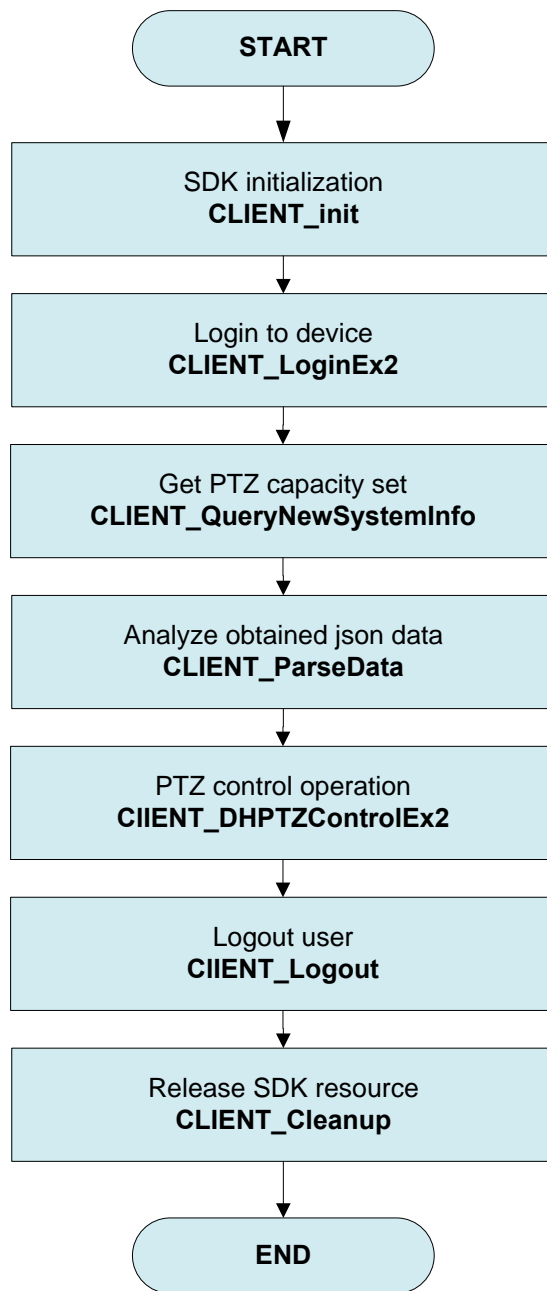


Figure 2-9 PTZ controller process flow sheet

**Process description:**

1. SDK initialization
2. After successful initialization, call CLIENT\_LoginEx2 to login device.
3. After successfully login, firstly call CLIENT\_QueryNewSystemInfo interface. The second parameter szCommand is set to CFG\_CAP\_CMD\_PTZ to obtain PTZ capacity set, and secondly call CLIENT\_ParseData interface, The parameter szCommand is set to

CFG\_CAP\_CMD\_PTZ to analyze the obtained capacity set.

4. Call CLIENT\_DHPTZControlEx2 interface to operate PTZ according to the demand.  
Different PTZ command requires may have different parameters. For example some commands as left/right movement may require corresponding stop command. Please refer to example code.
5. After operation is done, call CLIENT\_Logout to logout device.
6. After SDK function is done, call CLIENT\_Cleanup to release SDK resource.

## 2.6.4 Example Code

```
#include<windows.h>
#include <stdio.h>
#include <vector>
#include <string>
#include "dhnetsdk.h"
#include "dhconfigsdk.h"

#pragma comment(lib , "dhnetsdk.lib")
#pragma comment(lib , "dhconfigsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_ILoginHandle = 0L;
static LLONG g_IDownloadHandle = 0L;
static char g_szDevIp[32] = "171.2.7.34";
static int g_nPort = 37777; // tcp to connect port, need to match tcp port config in expected
login device page
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);
```

```
// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

//*****
void InitTest()
{
    // SDK initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // get SDK version info
    // this operation is optional
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // set CLIENT_SetAutoReconnect interface. Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally.
    // this operation is optional, but recommended
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

    // set connecting device timeout time and attempts
    // this operation is optional
    int nWaitTime = 5000;    // timeout time is set to 5s
    int nTryTimes = 3;       // if timeout, you have 3 attempts
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    //Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
    //This operation is optional.
    NET_PARAM stuNetParm = {0};
```

```
stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_ILoginHandle)
{
    // login device
    g_ILoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if (0 == g_ILoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhSDK.h. It is
to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        // example:
        // #define NET_NOT_SUPPORTED_EC(23)
        // current SDK does not support the function, corresponding error code is 0x80000017.
        Decimal number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
    }
    else
    {
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
    }
    // The first time logging to device, some data is needed to be initialized to enable
normal business function. It is recommended to wait for a while after login, the waiting time varies
by devices.
    Sleep(1000);
    printf("\n");
}

// Ptz control info struct
typedef struct tagPtzControllInfo
{
    tagPtzControllInfo():m_iCmd(-1), m_bStopFlag(false){}
    tagPtzControllInfo(int iCmd, const std::string& sDescription, bool
bStopFlag):m_iCmd(iCmd), m_sDescription(sDescription), m_bStopFlag(bStopFlag){}
    int m_iCmd;
    std::string m_sDescription;
    bool m_bStopFlag; // parial Ptz operation , after start need to call corresponding stop
```

operation

```
}PtzControllInfo;

// get int input
int GetIntInput(char *szPromt, int& nError);

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // get PTZ capacity set
    char szBuffer[2048] = "";

    int nError = 0;
    if (FALSE == CLIENT_QueryNewSystemInfo(g_lLoginHandle, CFG_CAP_CMD_PTZ, 0,
szBuffer, (DWORD)sizeof(szBuffer), &nError))
    {
        printf("CLIENT_QueryNewSystemInfo Failed, cmd[CFG_CAP_CMD_PTZ], Last
Error[%x]\n", CLIENT_GetLastError());
        return;
    }

    CFG_PTZ_PROTOCOL_CAPS_INFO stuPtzCapsInfo =
{sizeof(CFG_PTZ_PROTOCOL_CAPS_INFO)};
    if (FALSE == CLIENT_ParseData(CFG_CAP_CMD_PTZ, szBuffer, &stuPtzCapsInfo,
sizeof(stuPtzCapsInfo), NULL))
    {
        printf("CLIENT_ParseData Failed, cmd[CFG_CAP_CMD_PTZ], Last Error[%x]\n",
CLIENT_GetLastError());
        return;
    }

    // PTZ operation
    std::vector<PtzControllInfo> vecPtzControl;
    if (TRUE == stuPtzCapsInfo.bTile)
```

```
{
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_UP_CONTROL), "up", true));
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_DOWN_CONTROL), "down",
true));
}

if (TRUE == stuPtzCapsInfo.bPan)
{
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_LEFT_CONTROL), "left",
true));
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_RIGHT_CONTROL), "right",
true));
}

if (TRUE == stuPtzCapsInfo.bZoom)
{
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_ZOOM_ADD_CONTROL),
"zoom +", true));
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_ZOOM_DEC_CONTROL),
"zoom -", true));
}

if (TRUE == stuPtzCapsInfo.bFocus)
{
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_FOCUS_ADD_CONTROL),
"focus +", true));
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_FOCUS_DEC_CONTROL),
"focus -", true));
}

if (TRUE == stuPtzCapsInfo.blris)
{
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_APERTURE_ADD_CONTROL), "aperture
+", true));

    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_APERTURE_DEC_CONTROL), "aperture -",
true));
}

if (TRUE == stuPtzCapsInfo.bPreset)
{
    vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_POINT_MOVE_CONTROL),
```

```
"go to preset point", false));
    vecPtzControl.push_back(PtzControllInfo(int(DH_PTZ_POINT_SET_CONTROL), "set
preset point", false));
}

    if (TRUE == stuPtzCapsInfo.bRemovePreset)
    {
        vecPtzControl.push_back(PtzControllInfo(int(DH_PTZ_POINT_DEL_CONTROL),
"delete preset point", false));
    }

    if (TRUE == stuPtzCapsInfo.bTour)
    {
        vecPtzControl.push_back(PtzControllInfo(int(DH_PTZ_POINT_LOOP_CONTROL),
"point-to-point tour", false));
        vecPtzControl.push_back(PtzControllInfo(int(DH_EXTPTZ_ADDTOLOOP), "add
preset point to tour", false));
        vecPtzControl.push_back(PtzControllInfo(int(DH_EXTPTZ_DELFROMLOOP), "delete
preset point in tour", false));
    }

    if (TRUE == stuPtzCapsInfo.bRemoveTour)
    {
        vecPtzControl.push_back(PtzControllInfo(int(DH_EXTPTZ_CLOSELOOP), "clear
tour", false));
    }

    if (TRUE == stuPtzCapsInfo.bTile && TRUE == stuPtzCapsInfo.bPan)
    {
        vecPtzControl.push_back(PtzControllInfo(int(DH_EXTPTZ_LEFTTOP), "left up",
true));
        vecPtzControl.push_back(PtzControllInfo(int(DH_EXTPTZ_RIGHTTOP), "right up",
true));
        vecPtzControl.push_back(PtzControllInfo(int(DH_EXTPTZ_LEFTDOWN), "left down",
true));
        vecPtzControl.push_back(PtzControllInfo(int(DH_EXTPTZ_RIGHTDOWN), "right
down", true));
    }

    if (TRUE == stuPtzCapsInfo.bMoveRelatively)
    {
        vecPtzControl.push_back(PtzControllInfo(int(DH_EXTPTZ_FASTGOTO), "quick
position", false));
```



```
    }

    if (TRUE == stuPtzCapsInfo.bMoveAbsolutely)
    {
        vecPtzControl.push_back(PtzControllInfo(int(DH_EXTPTZ_EXACTGOTO), "3D
position precisely ", false));
    }

    vecPtzControl.push_back(PtzControllInfo(int(-2), "pause", false));
    vecPtzControl.push_back(PtzControllInfo(int(-1), "exit", true));
    PtzControllInfo cLastChoose;
    while(TRUE)
    {
        printf("PTZ control operation: \n");
        for (std::vector<PtzControllInfo>::const_iterator iter = vecPtzControl.begin(); iter !=
vecPtzControl.end(); ++iter)
        {
            printf("\t%d\t:%s\n", iter->m_iCmd, iter->m_sDescription.c_str());
        }
        int nError = 0;
        int nChoose = GetIntInput("\t selection: ", nError);
        if (0 != nError)
        {
            printf("invalid input!\n");
            continue;
        }

        std::vector<PtzControllInfo>::iterator iterFind = vecPtzControl.begin();
        for (; iterFind != vecPtzControl.end(); ++iterFind)
        {
            if (nChoose == iterFind->m_iCmd)
            {
                break;
            }
        }

        if (iterFind == vecPtzControl.end())
        {
            printf("pleas input valid operation\n");
            continue;
        }
        // stop the last opetation
        int nChannelId = 0;
```

```
        if (true == cLastChoose.m_bStopFlag)
        {
            if (FALSE == CLIENT_DHPTZControlEx2(g_lLoginHandle, nChannelId,
cLastChoose.m_iCmd, 0, 0, 0, TRUE))
            {
                printf("CLIENT_DHPTZControlEx2 Failed,
cLastChoose->GetCmd()[%x]! Last Error[%x]\n", cLastChoose.m_iCmd, CLIENT_GetLastError());
            }
        }

        if (iterFind->m_sDescription == "pause")
        {
            cLastChoose = *iterFind;
            continue;
        }

        if (iterFind->m_sDescription == "exit")
        {
            break;
        }

        // different PTZ commands correspond to different extra parameter setup plans .
Parameter setup guide are showing below:
        // extra parameter
        LONG lParam1 = 0;
        LONG lParam2 = 0;
        LONG lParam3 = 0;
        void* pParam4 = NULL;
        if (DH_PTZ_UP_CONTROL <= iterFind->m_iCmd && iterFind->m_iCmd <=
DH_PTZ_RIGHT_CONTROL)
        {
            // vertical/horizontal movement speed , valid range (1-8)
            lParam2 = 3;
        }
        else if (DH_PTZ_ZOOM_ADD_CONTROL <= iterFind->m_iCmd &&
iterFind->m_iCmd <= DH_PTZ_APERTURE_DEC_CONTROL)
        {
            // speed, valid range (1-8)
            lParam1 = 3;
        }
        else if (DH_PTZ_POINT_MOVE_CONTROL <= iterFind->m_iCmd &&
iterFind->m_iCmd <= DH_PTZ_POINT_DEL_CONTROL)
        {
```

```
// IParam2 means preset point NO.
printf("\t preset point NO. (%2d-%2d):",
stuPtzCapsInfo.wPresetMin, stuPtzCapsInfo.wPresetMax);
scanf("%d", &IParam2);
}
else if (DH_PTZ_POINT_LOOP_CONTROL == iterFind->m_iCmd)
{
    // IParam1 means tour path, IParam3: 76 start; 96 stop
    printf("\t tour path (%2d-%2d):",
stuPtzCapsInfo.wTourMin, stuPtzCapsInfo.wTourMax);
    scanf("%d", &IParam1);
    printf("\t1:start \n\t2: stop \n\t select:");
    int nTmp = 0;
    scanf("%d", &nTmp);
    if (1 == nTmp)
    {
        IParam3 = 76;
    }
    else if (2 == nTmp)
    {
        IParam3 = 96;
    }
}
else if (DH_PTZ_LAMP_CONTROL == iterFind->m_iCmd)
{
    // IParam1 means switch control
    printf("\t1:enable\n\t0:disable\n\t select:");
    scanf("%d", &IParam1);
}
else if (DH_EXTPTZ_LEFTTOP <= iterFind->m_iCmd && iterFind->m_iCmd <=
DH_EXTPTZ_RIGHTDOWN)
{
    // vertical speed, valid range (1-8)
    IParam1 = 1;
    // horizontal speed, valid range (1-8)
    IParam2 = 1;
}
else if (DH_EXTPTZ_ADDTOLOOP <= iterFind->m_iCmd && iterFind->m_iCmd <=
DH_EXTPTZ_DELFROMLOOP)
{
    // IParam1 means tour path
    printf("\t tour path (%2d-%2d):",
stuPtzCapsInfo.wTourMin, stuPtzCapsInfo.wTourMax);
```

```
scanf("%d", &IParam1);
// IParam2 means preset point NO.
printf("\t preset point NO. (%2d-%2d):",
stuPtzCapsInfo.wPresetMin,stuPtzCapsInfo.wPresetMax);
scanf("%d", &IParam2);
}
else if (DH_EXTPTZ_CLOSELOOP == iterFind->m_iCmd)
{
// IParam1 means tour path
printf("\t tour path(%2d-%2d):",
stuPtzCapsInfo.wTourMin,stuPtzCapsInfo.wTourMax);
scanf("%d", &IParam1);
}
else if (DH_EXTPTZ_FASTGOTO == iterFind->m_iCmd)
{
// horizontal coordinate, valid range (-8191 ~ 8191)
IParam1 = 2000;
// vertical coordinate, valid range (-8191 ~ 8191)
IParam2 = 2000;
// zoom, valid range (-16 ~ 16)
IParam3 = 2;
}
else if (DH_EXTPTZ_EXACTGOTO == iterFind->m_iCmd)
{
// horizontal coordinate, valid range and accuracy is 10x of capacity set
acquisition range
printf("\t horizontal coordinate (%2d-%2d):",
10*stuPtzCapsInfo.stuPtzMotionRange.nHorizontalAngleMin,
10*stuPtzCapsInfo.stuPtzMotionRange.nHorizontalAngleMax);
scanf("%d", &IParam1);
// vertical coordinate, valid range, accuracy is 10x of capacity set acquisition
range
printf("\t vertical coordinate (%2d-%2d):",
10*stuPtzCapsInfo.stuPtzMotionRange.nVerticalAngleMin,
10*stuPtzCapsInfo.stuPtzMotionRange.nVerticalAngleMax);
scanf("%d", &IParam2);
// zoom, valid range (1 ~ 128)
IParam3 = 2;
}

if (FALSE == CLIENT_DHPTZControlEx2(gILoginHandle, nChannelId,
iterFind->m_iCmd, IParam1, IParam2, IParam3, FALSE, pParam4))
{
```

```
        printf("CLIENT_DHPTZControlEx2 Failed, nChoose[%x]!Last Error[%x]\n" ,
nChoose, CLIENT_GetLastError());
    }
    cLastChoose = *iterFind;
}
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // logout device
    if (0 != g_ILLoginHandle)
    {
        if(FALSE == CLIENT_Logout(g_ILLoginHandle))
        {
            printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_ILLoginHandle = 0;
        }
    }
    // cleanup initialization resource
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
{
    InitTest();

    RunTest();

    EndTest();

    return 0;
}
```

```
//*****
// commonly used callback set definition
void CALLBACK DisConnectFunc(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

int GetIntInput(char *szPromt, int& nError)
{
    long int nGet = 0;
    char* pError = NULL;
    printf(szPromt);
    char szUserInput[32] = "";
    gets(szUserInput);
    nGet = strtol(szUserInput, &pError, 10);
    if ('\0' != *pError)
    {
        // input parameter error
        nError = -1;
    }
}
```

```
else
{
    nError = 0;
}

return nGet;
}
```

## 2.7 Audio intercom

### 2.7.1 Intro

Audio intercom is mainly used for audio interacting between local platform and front-end device located environment.

The chapter mainly tells users how to use SDK to audio intercoming with front-end device.

#### Audio intercom mode

Audio intercom has two modes: client mode and server mode, please refer to Ch 2.7.3 for details.

### 2.7.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for SDK Initialization.
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources
<a href="#">CLIENT_LoginEx2</a>	Extensive interface 2 for Sync login
<a href="#">CLIENT_QueryDevState</a>	Interface for searching device status
<a href="#">CLIENT_StartTalkEx</a>	Extensive interface for opening audio intercom
<a href="#">CLIENT_StopTalkEx</a>	Extensive interface for stopping audio intercom
<a href="#">CLIENT_RecordStartEx</a>	Extensive interface for starting client recording(valid in Windows platform only)
<a href="#">CLIENT_RecordStopEx</a>	Extensive interface for stopping client recording(valid in Windows platform only)

Interface	Interface Description
<a href="#">CLIENT_TalkSendData</a>	Interface for sending audio data to device
<a href="#">CLIENT_AudioDecEx</a>	Extensive interface for decoding audio data(valid in Windows platform only)
<a href="#">CLIENT_Logout</a>	Interface for logout
<a href="#">CLIENT_GetLastError</a>	Interface for getting error code after failed calling interface

## 2.7.3 Process Instruction

Audio intercom has two modes:

- Client mode

SDK allows user to provide a callback function. The callback function is called when SDK collects audio data from local sound card or receives data from the front-end. In callback function user can not only send collected local audio data to front-end device but decode and play the received front-end audio data. This mode is valid in Windows platform only.

- Server mode

SDK allows user to provide one callback function. The callback function is called when SDK receives audio data from front-end device. In callback function user can save audio data received from front-end device for future use such as audio data transfer, calling a third-party library to decode and play audio data and etc. For local audio data, user can collect it by calling a third-party library and then send it to device by calling SDK interface.

### 2.7.3.1 Client Mode

Client mode process flow sheet is shown below in Figure 2-10.



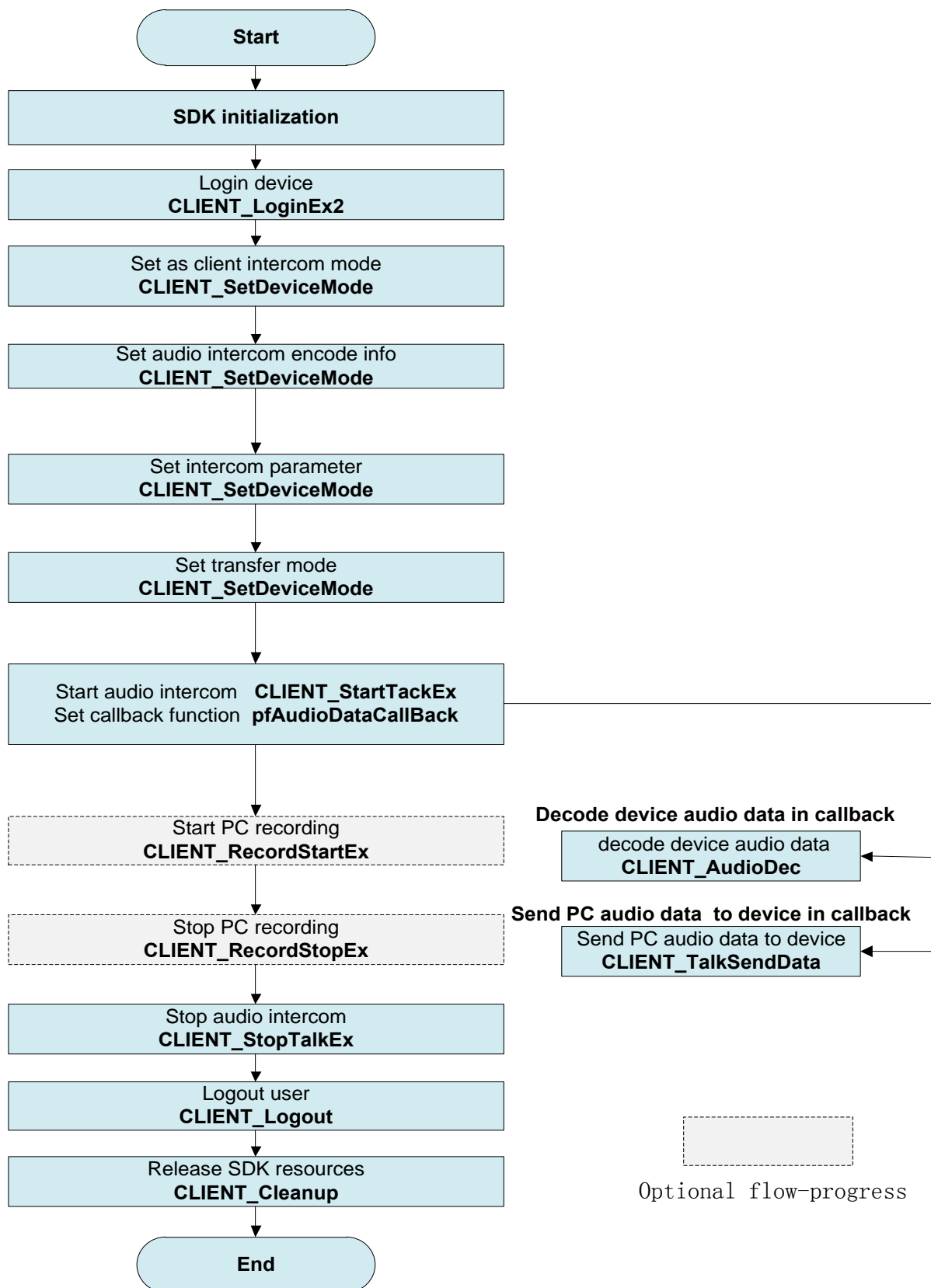


Figure 2-10 Client mode process flow sheet

**Process description:**

1. SDK initialization
2. After successful initialization, call CLIENT\_LoginEx2 to login device.
3. Call CLIENT\_SetDeviceMode to set client mode as audio intercom mode, Parameter emType is DH\_TALK\_CLIENT\_MODE.
4. Call CLIENT\_SetDeviceMode to set audio intercom encode information, Parameter emType is DH\_TALK\_ENCODE\_TYPE.
5. Call CLIENT\_SetDeviceMode to set audio intercom parameter, Parameter emType is DH\_TALK\_SPEAK\_PARAM.
6. Call CLIENT\_SetDeviceMode to set audio intercom transfer mode, no-transfer mode means audio intercom between local PC and logged device; and transfer mode means audio intercom between local PC and front-end device connected with specific channel of the logged device.
7. Call CLIENT\_StartTalkEx to set callback function and start audio intercom. In callback function, call CLIENT\_AudioDec to decode audio data sent by device and call CLIENT\_TalkSendData to send audio data from PC to device.
8. Call CLIENT\_RecordStartEx to start PC recording. Only after this interface is called, can audio intercom callback function set by CLIENT\_StartTalkEx will receive local audio data.
9. After audio intercom is finished, call CLIENT\_RecordStopEx to stop PC recording.
10. Call CLIENT\_StopTalkEx to stop audio intercom.
11. Call CLIENT\_Logout to logout device.
12. Last but not the least, call CLIENT\_Cleanup to release SDK resources.

**2.7.3.2 Server Mode**

Server mode process flow sheet is shown below in Figure 2-11.

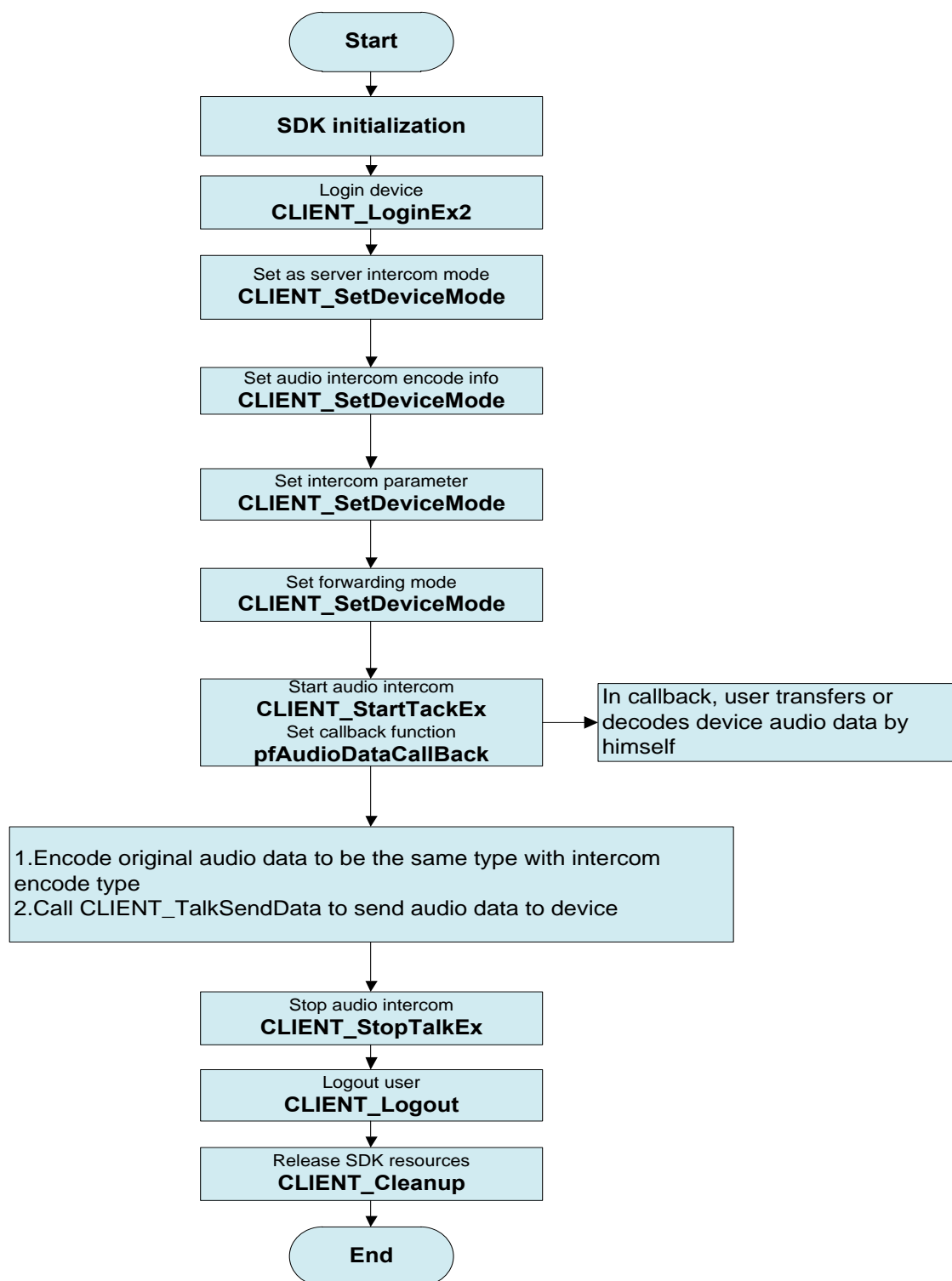


Figure 2-11 Server mode process flow sheet

**Process description:**

1. SDK initialization
2. After successful initialization, call **CLIENT\_LoginEx2** to login device.

3. Call `CLIENT_SetDeviceMode` to set server mode as audio intercom mode, Parameter `emType` is `DH_TALK_SERVER_MODE`.
4. Call `CLIENT_SetDeviceMode` to set audio intercom encode information, Parameter `emType` is `DH_TALK_ENCODE_TYPE`.
5. Call `CLIENT_SetDeviceMode` to set audio intercom parameter, Parameter `emType` is `DH_TALK_SPEAK_PARAM`.
6. Call `CLIENT_SetDeviceMode` to set audio intercom transfer mode, no-transfer mode means audio intercom between local PC and logged device; and transfer mode means audio intercom between local PC and front-end device connected with specific channel of the logged device.
7. Call `CLIENT_StartTalkEx` to set callback function and start audio intercom. In callback function, the audio data sent by device can be dealt with by users with the methods as transfer or decoding and playing.
8. At first user encodes original audio data to be the same type with intercom encode type, then adds 8 corresponding private protocol bytes in front of encoded data, and finally calls `CLIENT_TalkSendData` to send audio data to device.
9. After intercom function is finished, call `CLIENT_StopTalkEx` to stop audio intercom.
10. Call `CLIENT_Logout` to logout device.
11. Last but not the least, call `CLIENT_Cleanup` to release SDK resources.

## 2.7.4 Example Code

### 2.7.4.1 Client Mode

```
#include <windows.h>
#include <stdio.h>
#include "dhnetSDK.h"
#pragma comment(lib, "dhnetSDK.lib")
```

```
static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG gILoginHandle = 0L;
static LLONG gITalkHandle = 0L;
static BOOL g_bRecordFlag = FALSE;
static char g_szDevIp[32] = "172.23.2.66";
static WORD g_nPort = 37777;
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*****

// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

// audio intercom data callback function
// It is not recommended to call SDK interface in the SDK callback function, but in this callback
function CLIENT_TalkSendData and CLIENT_AudioDec SDK interfaces can be called.
// Set the callback function in CLIENT_StartTalkEx. SDK will call this function when receiving
sound card data detected by local PC, or audio data sent by device,.
void CALLBACK AudioDataCallBack(LLONG ITalkHandle, char *pDataBuf, DWORD
dwBufSize, BYTE byAudioFlag, DWORD dwUser);

//*****

void InitTest()
```

```
{  
    // SDK Initialization  
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);  
    if (FALSE == g_bNetSDKInitFlag)  
    {  
        printf("Initialize client SDK fail; \n");  
        return;  
    }  
    else  
    {  
        printf("Initialize client SDK done; \n");  
    }  
    //Get the SDK version info  
    // This operation is optional.  
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();  
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);  
  
    // set CLIENT_SetAutoReconnect interface.Once HaveReConnect is set, when device  
gets offline, SDK will reconnect internally  
    //This operation is optional,but recommended.  
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);  
  
    // Set device connection timeout value and trial times.  
    //This operation is optional.  
    int nWaitTime = 5000;    // timeout value is 5 seconds  
    int nTryTimes = 3;        //if timeout,it will try to log in three times.  
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);  
  
    //Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the  
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.  
    //This operation is optional.  
    NET_PARAM stuNetParm = {0};  
    stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.  
    CLIENT_SetNetworkParam(&stuNetParm);
```

```
NET_DEVICEINFO_Ex stDevInfoEx = {0};

int nError = 0;

while(0 == g_lLoginHandle)
{
    //Login device

    g_lLoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if(0 == g_lLoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhnetSDK.h.
        // It is to print hexadecimal here, not decimal shown in header file, please be careful with conversion
        // Example: #define NET_NOT_SUPPORTED_EC(23)
        // Now SDK does not support this function, error code is 0x80000017, Decimal
        // number 23 is hexadecimal 0x17.

        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
    }
    else
    {
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
    }

    // The first time logging to device, some data is needed to be initialized to enable
    // normal business function. It is recommended to wait for a while after login, the waiting time varies
    // by devices.

    Sleep(1000);
    printf("\n");
}

void RunTest()
{
```

```
if (FALSE == g_bNetSDKInitFlag)
{
    return;
}

if (0 == g_lLoginHandle)
{
    return;
}

//Set as audio intercom client mode
BOOL bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, DH_TALK_CLIENT_MODE,
NULL);
if (FALSE == bSuccess)
{
    printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n",
DH_TALK_CLIENT_MODE, CLIENT_GetLastError());
    return;
}

//Get audio intercom encode type supported by front-end device
DHDEV_TALKFORMAT_LIST stulstTalkEncode;
int retlen = 0;
bSuccess = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_TALK_ECTYPE,
(char*)&stulstTalkEncode, sizeof(stulstTalkEncode), &retlen, 3000);
if (FALSE == bSuccess || retlen != sizeof(stulstTalkEncode))
{
    printf("CLIENT_QueryDevState cmd[%d] Failed!Last Error[%x]\n",
DH_DEVSTATE_TALK_ECTYPE, CLIENT_GetLastError());
    return;
}
```



```
//Set audio intercom encode info.
DHDEV_TALKDECODE_INFO curTalkMode;

//Select the first encode method in the list, user can select other encode method as
needed.

curTalkMode = stulstTalkEncode.type[0];

bSuccess = CLIENT_SetDeviceMode(g_ILloginHandle, DH_TALK_ENCODE_TYPE,
&curTalkMode);

if (FALSE == bSuccess)
{
    printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n",
DH_TALK_ENCODE_TYPE, CLIENT_GetLastError());

    return;
}

//Set audio intercom parameter
NET_SPEAK_PARAM stuSpeak = {sizeof(stuSpeak)};

stuSpeak.nMode = 0;    // 0: intercom (default mode), 1: shout; Reset stuSpeak.nMode
when switch from shout to intercom.

stuSpeak.nSpeakerChannel = 0;    // audio intercom channel number, set 0

bSuccess = CLIENT_SetDeviceMode(g_ILloginHandle, DH_TALK_SPEAK_PARAM,
&stuSpeak);

if (FALSE == bSuccess)
{
    printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n",
DH_TALK_SPEAK_PARAM, CLIENT_GetLastError());

    return;
}

//Set as transfer mode
NET_TALK_TRANSFER_PARAM stuTransfer = {sizeof(stuTransfer)};

stuTransfer.bTransfer = FALSE; //Close transfer mode because it's intercom with login
device.

bSuccess = CLIENT_SetDeviceMode(g_ILloginHandle, DH_TALK_TRANSFER_MODE,
```

```
&stuTransfer);

    if (FALSE == bSuccess)
    {
        printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n",
DH_TALK_TRANSFER_MODE, CLIENT_GetLastError());
        return;
    }

    g_ITalkHandle = CLIENT_StartTalkEx(gILoginHandle, AudioDataCallBack,
(DWORD)NULL);
    if(0 != g_ITalkHandle)
    {
        //Start local recording.It's no need to call this interface if it is one-way audio intercom
between DVR and PC.

        BOOL bSuccess = CLIENT_RecordStartEx(gILoginHandle);
        if(TRUE == bSuccess)
        {
            g_bRecordFlag = TRUE;
        }
        else
        {
            if (FALSE == CLIENT_StopTalkEx(g_ITalkHandle))
            {
                printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n",
CLIENT_GetLastError());
            }
            else
            {
                g_ITalkHandle = 0;
            }
        }
    }
}
```

```
else
{
    printf("CLIENT_StartTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
}
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    //Stop local recording
    if (TRUE == g_bRecordFlag)
    {
        if (!CLIENT_RecordStopEx(g_LoginHandle))
        {
            printf("CLIENT_RecordStop Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_bRecordFlag = FALSE;
        }
    }
    //Stop audio intercom
    if (0 != g_ITalkHandle)
    {
        if(FALSE == CLIENT_StopTalkEx(g_ITalkHandle))
        {
            printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {

```

```
        g_ITalkHandle = 0;
    }
}

//Exit device
if (0 != g_ILoginHandle)
{
    if(FALSE == CLIENT_Logout(g_ILoginHandle))
    {
        printf("CLIENT_Logout Failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
    else
    {
        g_ILoginHandle = 0;
    }
}

// Clean up initialization resources.
if (TRUE == g_bNetSDKInitFlag)
{
    CLIENT_Cleanup();
    g_bNetSDKInitFlag = FALSE;
}

return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}
```

```
}

//*****

//commonly used callback functions definition

void CALLBACK DisConnectFunc(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

```
void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD
dwBufSize, BYTE byAudioFlag, DWORD dwUser)
{
    //If different audio intercoms use the same data callback function, user can judge by
parameter 1TalkHandle.

    if (g_lTalkHandle != lTalkHandle)
    {
        return;
    }

    if(0 == byAudioFlag)
    {
        //Send received sound card data which is detected by local PC to device. This
interface must follow the interface CLIENT_RecordStartEx.

        LONG lSendLen = CLIENT_TalkSendData(lTalkHandle, pDataBuf, dwBufSize);
        if(lSendLen != (LONG)dwBufSize)
        {
            printf("CLIENT_TalkSendData Failed!Last Error[%x]\n" ,
CLIENT_GetLastError());
        }
    }
    else if(1 == byAudioFlag)
    {
        //Send received audio data sent by device to SDK for decoding and playing.

        CLIENT_AudioDec(pDataBuf, dwBufSize);
#ifdef _DEBUG
        FILE *stream;
        if( (stream = fopen("E:\\Talk.txt", "a+b")) != NULL )
        {
            int numwritten = fwrite( pDataBuf, sizeof( char ), dwBufSize, stream );
            fclose( stream );
        }
#endif
    }
}
```

```
    }  
#endif  
    }  
}
```

## 2.7.4.2 Server Mode

```
#include <windows.h>  
#include <stdio.h>  
#include "dhplay.h"  
#include "Alaw_encoder.h"  
#include "dhnetsdk.h"
```

#pragma comment(lib, "dhplay.lib") // the third-party encoding/decoding library. Take Dahua encoding/decoding library for example in the following example code.

```
#pragma comment(lib, "dhnetsdk.lib")
```

```
static BOOL g_bNetSDKInitFlag = FALSE;  
static LLONG g_lLoginHandle = 0L;  
static LLONG g_lTalkHandle = 0L;  
static BOOL g_bOpenAudioRecord = FALSE;  
static char g_szDevIp[32] = "172.23.1.27";  
static WORD g_nPort = 37777;  
static char g_szUserName[64] = "admin";  
static char g_szPasswd[64] = "admin";  
static DHDEV_TALKDECODE_INFO g_curTalkMode;
```

```
//*****
```

```
// commonly used callback set declaration
```

```
// callback function used when device gets offline
```

```
// It's not recommended to call SDK interfaces in this callback function
```

//Set the callback function in CLIENT\_Init, when device gets offline, SDK will call this function automatically

```
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort,  
DWORD dwUser);
```

```
// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

// audio intercom data callback function
//only audio data sent by device can be received in server mode
// It is not recommended to call SDK interface in the SDK callback function, but in this callback
function CLIENT_TalkSendData and CLIENT_AudioDec SDK interfaces can be called.
// Set the callback function in CLIENT_StartTalkEx. SDK will call this function when receiving
sound card data detected by local PC, or audio data sent by device,.
void CALLBACK AudioDataCallBack(LLONG ITalkHandle, char *pDataBuf, DWORD
dwBufSize, BYTE byAudioFlag, DWORD dwUser);

// PC audio encode send callback function
//pDataBuffer is original audio data, DataLength is the length of valid data.
//Set up PLAY_OpenAudioRecord interface By Dahua encoding/decoding library, when
detecting sound card data, Dahua encoding/decoding library will call this function.
void CALLBACK AudioCallFunction(LPBYTE pDataBuffer, DWORD DataLength, void* pUser);

//*****

// function declaration
// This interface is an example to call Dahua encoding/decoding library to collect audio
intercom data. Use Dahua encoding/decoding library to get PC original audio stream.
BOOL StartAudioRecord();
BOOL StopAudioRecord();

//*****

void InitTest()
{
    // SDK Initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
```



```
{
    printf("Initialize client SDK fail; \n");
    return;
}
else
{
    printf("Initialize client SDK done; \n");
}

//Get the SDK version info
// This operation is optional.
DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
printf("NetSDK version is [%d]\n", dwNetSdkVersion);

// set CLIENT_SetAutoReconnect interface.Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally
//This operation is optional,but recommended.
CLIENT_SetAutoReconnect(&HaveReConnect, 0);

// Set device connection timeout value and trial times.
//This operation is optional.
int nWaitTime = 5000;    // timeout value is 5 seconds
int nTryTimes = 3;      //if timeout,it will try to log in three times.
CLIENT_SetConnectTime(nWaitTime, nTryTimes);

//Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
//This operation is optional.
NET_PARAM stuNetParm = {0};
stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_ILoginHandle)
```

```
{  
    //Login device  
    g_ILLoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,  
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);  
  
    if(0 == g_ILLoginHandle)  
    {  
        // according to error code, you can find corresponding explanation in dhnetsdk.h.  
        It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion  
        //Example: #define NET_NOT_SUPPORTED_EC(23)  
        //Now SDK does not support this function, error code is 0x80000017, Decimal  
        number 23 is hexadecimal 0x17.  
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp , g_nPort ,  
CLIENT_GetLastError());  
    }  
    else  
    {  
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp , g_nPort);  
    }  
    // The first time logging to device, some data is needed to be initialized to enable  
    normal business function. It is recommended to wait for a while after login, the waiting time varies  
    by devices.  
    Sleep(1000);  
    printf("\n");  
}  
  
void RunTest()  
{  
    if (FALSE == g_bNetSDKInitFlag)  
    {  
        return;  
    }  
}
```

```
if (0 == g_ILoginHandle)
{
    return;
}

//Set as server intercom mode
BOOL bSuccess = CLIENT_SetDeviceMode(g_ILoginHandle,
DH_TALK_SERVER_MODE, NULL);
if (FALSE == bSuccess)
{
    printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n",
DH_TALK_SERVER_MODE, CLIENT_GetLastError());
    return;
}

//Get audio intercom encode type supported by front-end device
DHDEV_TALKFORMAT_LIST stulstTalkEncode;
int retlen = 0;
bSuccess = CLIENT_QueryDevState(g_ILoginHandle, DH_DEVSTATE_TALK_ECTYPE,
(char*)&stulstTalkEncode, sizeof(stulstTalkEncode), &retlen, 3000);
if (FALSE == bSuccess || retlen != sizeof(stulstTalkEncode))
{
    printf("CLIENT_QueryDevState cmd[%d] Failed!Last Error[%x]\n",
DH_DEVSTATE_TALK_ECTYPE, CLIENT_GetLastError());
    return;
}

//Set audio intercom encode info
g_curTalkMode = stulstTalkEncode.type[0];
bSuccess = CLIENT_SetDeviceMode(g_ILoginHandle, DH_TALK_ENCODE_TYPE,
&g_curTalkMode);
if (FALSE == bSuccess)
```

```
{  
    printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n",  
DH_TALK_ENCODE_TYPE, CLIENT_GetLastError());  
    return;  
}  
  
//Set intercom parameter  
NET_SPEAK_PARAM stuSpeak = {sizeof(stuSpeak)};  
stuSpeak.nMode = 0;    // 0: intercom (default mode), 1: shout; Reset stuSpeak.nMode  
when switch from shout to intercom  
stuSpeak.nSpeakerChannel = 0;    //audio intercom channel number,set 0.  
bSuccess = CLIENT_SetDeviceMode(gILoginHandle, DH_TALK_SPEAK_PARAM,  
&stuSpeak);  
if (FALSE == bSuccess)  
{  
    printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n",  
DH_TALK_SPEAK_PARAM, CLIENT_GetLastError());  
    return;  
}  
  
//Set as transfer mode  
NET_TALK_TRANSFER_PARAM stuTransfer = {sizeof(stuTransfer)};  
stuTransfer.bTransfer = FALSE; //Close transfer mode because it's intercom with login  
device.  
bSuccess = CLIENT_SetDeviceMode(gILoginHandle, DH_TALK_TRANSFER_MODE,  
&stuTransfer);  
if (FALSE == bSuccess)  
{  
    printf("CLIENT_SetDeviceMode cmd[%d] Failed!Last Error[%x]\n",  
DH_TALK_TRANSFER_MODE, CLIENT_GetLastError());  
    return;  
}
```

```
g_ITalkHandle = CLIENT_StartTalkEx(gILoginHandle, AudioDataCallBack,
(DWORD)NULL);

if(0 != g_ITalkHandle)
{
    bSuccess = StartAudioRecord();
    if(TRUE == bSuccess)
    {
        g_bOpenAudioRecord = TRUE;
    }
    else
    {
        printf("StartAudioRecord Failed!\n");
        CLIENT_StopTalkEx(g_ITalkHandle);
        g_ITalkHandle = 0;
    }
}
else
{
    printf("CLIENT_StartTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
}
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // clean up Dahua encoding/decoding library resources.
    if(TRUE == g_bOpenAudioRecord)
    {
        if (TRUE == StopAudioRecord())
        {

```

```
        g_bOpenAudioRecord = FALSE;
    }
}

//Stop audio intercom
if (0 != g_ITalkHandle)
{
    if(FALSE == CLIENT_StopTalkEx(g_ITalkHandle))
    {
        printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
    }
    else
    {
        g_ITalkHandle = 0;
    }
}

//Logout device
if (0 != gILoginHandle)
{
    if(FALSE == CLIENT_Logout(gILoginHandle))
    {
        printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
    }
    else
    {
        gILoginHandle = 0;
    }
}

// Clean up initialization resources.
if (TRUE == g_bNetSDKInitFlag)
{
```

```
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }

    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//*****

//commonly used callback functions definition

void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

```
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
```

```
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

```
void CALLBACK AudioDataCallBack(LLONG ITalkHandle, char *pDataBuf, DWORD
dwBufSize, BYTE byAudioFlag, DWORD dwUser)
```

```
{
    //If different audio intercoms use the same data callback function, user can judge by
parameter ITalkHandle.
```

```
    if (g_ITalkHandle != ITalkHandle)
    {
        return;
    }
```

```
    if(1 == byAudioFlag)
    {
```

```
        // User can handle the audio data sent by device by himself such as transfer and
decoding and playing.
```

```
        // The following is an example of dealing with the data with Dahua encoding/decoding
library
```



```
int nPort = 99;

//For PCM format without header, please add 128.
if (g_curTalkMode.encodeType == DH_TALK_DEFAULT)
{
    nPort = 100;
    for (unsigned int i = 0; i < dwBufSize; i++)
    {
        pDataBuf[i] += (char)128;
    }
}
```

//You can use PLAY SDK to decode to get PCM and then encode to other formats if you to get a uniform format.

```
PLAY_InputData(nPort, (BYTE *)pDataBuf, dwBufSize);

#ifdef _DEBUG
    FILE *stream;
    if ( (stream = fopen("E:\\Talk.txt", "a+b")) != NULL )
    {
        int numwritten = fwrite( pDataBuf, sizeof( char ), dwBufSize, stream );
        fclose( stream );
    }
#endif
}
```

```
void CALLBACK AudioCallFunction(LPBYTE pDataBuffer, DWORD DataLength, void* pUser)
{
    char* pCbData = NULL;
    pCbData = new char[102400];
    if (NULL == pCbData)
    {
```

```
        return;

    }

    int iCbLen = 0;

    //Former 8 bytes in intercom stream are private protocol data, others are audio data of
corresponding intercom encode type.

    // The following codes show that what the former 8 bytes are when PCM、g711a and g711u
encoding.

    if (g_curTalkMode.encodeType == DH_TALK_DEFAULT || g_curTalkMode.encodeType ==
DH_TALK_PCM)
    {
        if (g_curTalkMode.nAudioBit == 8)
        {
            for(unsigned int j = 0 ; j < DataLength; j++)
            {
                *(pDataBuffer + j) += 128;
            }
        }

        pCbData[0]=0x00;
        pCbData[1]=0x00;
        pCbData[2]=0x01;
        pCbData[3]=0xF0;

        pCbData[4]=g_curTalkMode.nAudioBit==8?0x07:0x0C;
        if( 8000 == g_curTalkMode.dwSampleRate )
        {
            pCbData[5]=0x02;//8k
        }
        else if(16000 == g_curTalkMode.dwSampleRate)
        {
            pCbData[5] = 0x04;
        }
    }
```

```
else if(48000 == g_curTalkMode.dwSampleRate)
{
    pCbData[5] = 0x09;
}

*(DWORD*)(pCbData+6)=DataLength;
memcpy(pCbData+8, pDataBuffer, DataLength);

iCbLen = 8+DataLength;
}
else if (g_curTalkMode.encodeType == DH_TALK_G711a)
{
    //Encode the original audio data to g711a.
    if (g711a_Encode((char*)pDataBuffer, pCbData+8, DataLength, &iCbLen) != 1)
    {
        goto end;
    }

    //Private bit stream format frame head
    pCbData[0]=0x00;
    pCbData[1]=0x00;
    pCbData[2]=0x01;
    pCbData[3]=0xF0;

    pCbData[4]=0x0E; //G711A

    if( 8000 == g_curTalkMode.dwSampleRate )
    {
        pCbData[5]=0x02;//8k
    }

    else if(16000 == g_curTalkMode.dwSampleRate)
```

```
{
    pCbData[5] = 0x04;
}
else if(48000 == g_curTalkMode.dwSampleRate)
{
    pCbData[5] = 0x09;
}

pCbData[6]=BYTE(iCbLen&0xff);
pCbData[7]=BYTE(iCbLen>>8);

iCbLen += 8;
}
else if (g_curTalkMode.encodeType == DH_TALK_G711u)
{
    // Encode the original audio data to g711u.
    if (g711u_Encode((char*)pDataBuffer, pCbData+8, DataLength, &iCbLen) != 1)
    {
        goto end;
    }

    //Private bit stream format frame head.
    pCbData[0]=0x00;
    pCbData[1]=0x00;
    pCbData[2]=0x01;
    pCbData[3]=0xF0;

    pCbData[4]=0x0A; //G711u
    if( 8000 == g_curTalkMode.dwSampleRate )
    {
        pCbData[5]=0x02; //8k
```

```
    }

    else if(16000 == g_curTalkMode.dwSampleRate)
    {
        pCbData[5] = 0x04;
    }

    else if(48000 == g_curTalkMode.dwSampleRate)
    {
        pCbData[5] = 0x09;
    }

    pCbData[6]=BYTE(iCbLen&0xff);
    pCbData[7]=BYTE(iCbLen>>8);

    iCbLen += 8;
}
else
{
    goto end;
}

// Send the data from the PC to DVR
CLIENT_TalkSendData(g_ITalkHandle, (char *)pCbData, iCbLen);

end:
if (pCbData != NULL)
{
    delete[] pCbData;
}
}

//*****
```

```
BOOL StartAudioRecord()
{
    // It is the characteristics of Dahua encoding/decoding library.
    //First confirm decode port.DH_TALK_DEFAULT is 100 port number and then rest is 99
port number.
    int nPort = 99;
    if (g_curTalkMode.encodeType == DH_TALK_DEFAULT)
    {
        nPort = 100;
    }

    //Then specify frame length
    int nFrameLength = 1024;
    switch(g_curTalkMode.encodeType)
    {
        case DH_TALK_DEFAULT:
        case DH_TALK_PCM:
            nFrameLength = 1024;
            break;
        case DH_TALK_G711a:
            nFrameLength = 1280;
            break;
        case DH_TALK_AMR:
            nFrameLength = 320;
            break;
        case DH_TALK_G711u:
            nFrameLength = 320;
            break;
        case DH_TALK_G726:
            nFrameLength = 320;
            break;
```

```
case DH_TALK_AAC:
    nFrameLength = 1024;
default:
    break;
}

if (g_curTalkMode.dwSampleRate == 48000) //If sampling rate is 48K,update audio
length.
{
    nFrameLength = 48*40*2; // sampling rate multiply by 40 and 2.
}

BOOL bRet = FALSE;

//Then call PLAYSDK library to begin recording audio
BOOL bOpenRet = PLAY_OpenStream(nPort,0,0,1024*900);
if(bOpenRet)
{
    BOOL bPlayRet = PLAY_Play(nPort,0);
    if(bPlayRet)
    {
        PLAY_PlaySoundShare(nPort);
        BOOL bSuccess =
            PLAY_OpenAudioRecord(AudioCallFunction,g_curTalkMode.nAudioBi
            t,
g_curTalkMode.dwSampleRate,nFrameLength,0,NULL);
        if(bSuccess)
        {
            bRet = TRUE;
        }
        else
        {
```

```
        PLAY_StopSoundShare(nPort);
        PLAY_Stop(nPort);
        PLAY_CloseStream(nPort);
    }
}
else
{
    PLAY_CloseStream(nPort);
}
}

return bRet;
}

BOOL StopAudioRecord()
{
    // It is the characteristics of Dahua encoding/decoding library.
    BOOL bSuccess = PLAY_CloseAudioRecord();
    if(TRUE == bSuccess)
    {
        PLAY_Stop(100);
        PLAY_Stop(99);
        PLAY_StopSoundShare(100);
        PLAY_StopSoundShare(99);
        PLAY_CloseStream(100);
        PLAY_CloseStream(99);
    }
    else
    {
        printf("PLAY_CloseAudioRecord Failed!\n");
    }
}
```



```
return bSuccess;  
}
```

## 2.8 Snapshot

### 2.8.1 Intro

Video snapshot, as to snapshot picture not only from video, but also from device, used by upper users for platform development requirements.

Snapshot picture from device: User call SDK interface to send snapshot command to device, device snapshots current image in real-time monitoring and sends to SDK, SDK will return picture data to user, user can configure interface by SDK to set some parameters, such as picture encoding type and resolution.

### 2.8.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for SDK Initialization.
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources.
<a href="#">CLIENT_LoginEx2</a>	Extensive interface 2 for Sync login.
<a href="#">CLIENT_SetSnapRevCallBack</a>	Set video snapshot data callback function
<a href="#">CLIENT_SnapPictureEx</a>	Extensive interface for snapshot request.
<a href="#">CLIENT_Logout</a>	Interface for logout device.
<a href="#">CLIENT_GetLastError</a>	Interface for getting error code after failed calling interface.

### 2.8.3 Process Instruction

Video snapshot process flow sheet is shown below in Figure 2-12.

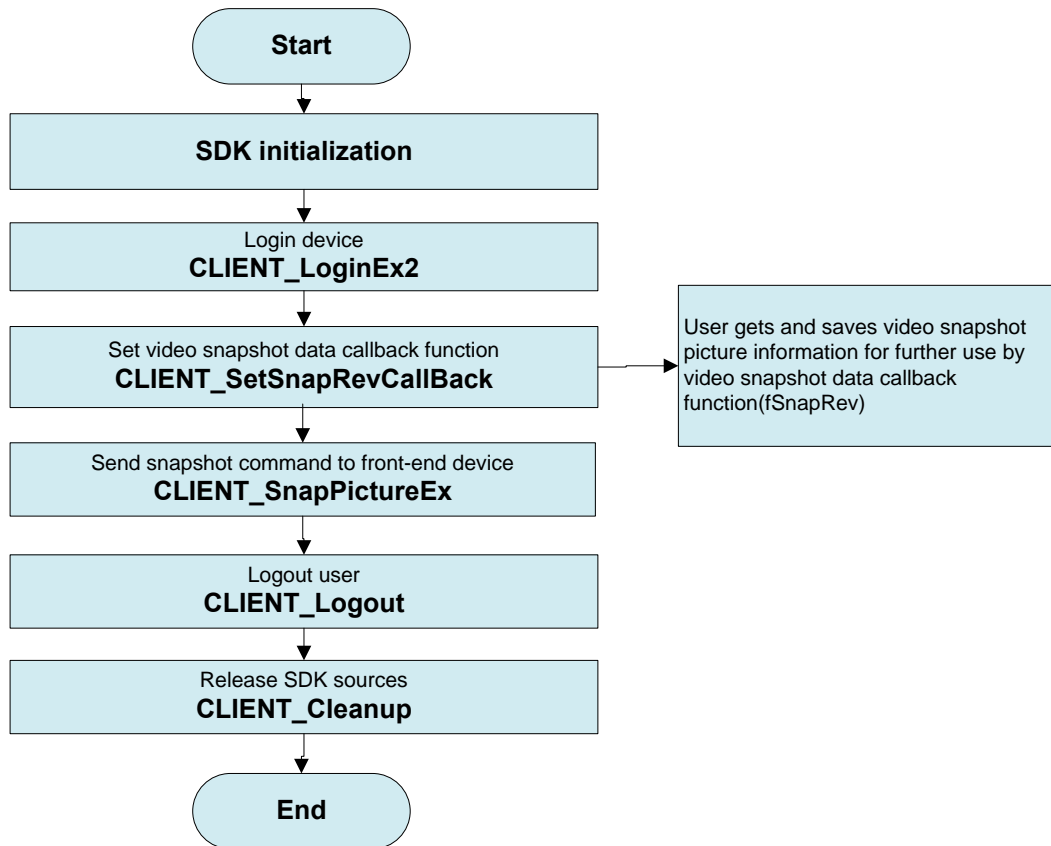


Figure 2-12 Video snapshot process flow sheet

**Process discription:**

1. SDK initialization.
2. After successful initialization, call `CLIENT_LoginEx2` to login device.
3. Call `CLIENT_SetSnapRevCallBack` to set snapshot callback function, when SDK receives snapshot data sent from device, it will call `fSnapRev` callback function to recall picture information and data to user.
4. Call `CLIENT_SnapPictureEx` to send snapshot command to front-end device, wait for device to reply picture information in callback function(`fSnapRev`).
5. Call `CLIENT_Logout` to logout device.
6. Last but not the least, call `CLIENT_Cleanup` to release SDK resources.

## 2.8.4 Example code

```
#include <windows.h>
#include <stdio.h>
#include <time.h>
#include "dhnet sdk.h"
#include "dhconfig sdk.h"

#pragma comment(lib, "dhnet sdk.lib")
#pragma comment(lib, "dhconfig sdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_ILoginHandle = 0L;
static char g_szDevIp[32] = "172.23.1.27";
static WORD g_nPort = 37777; // Before tcp connects port,the port need to keep with tcp port
config in expected login
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
static short g_nCmdSerial = 0; //snapshot SN

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init,when device gets offline,SDK will call this function
automatically
void CALLBACK DisConnectFunc(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully.
// It's not recommended to call SDK interfaces in this callback function.
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function.
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

//Snapshot callback function.
// It's not recommended to call SDK interfaces in this callback function.
//Set the callback function in CLIENT_SetSnapRevCallBack,when snapshot data is sent over
by front-end device,SDK will call this function.
void CALLBACK SnapRev(LLONG ILoginID, BYTE *pBuf, UINT RevLen, UINT EncodeType,
DWORD CmdSerial, LDWORD dwUser);

//*****
// commonly used function set declaration
```

```
//Get int input
int GetIntInput(char *szPromt, int& nError);

//Get input string
void GetStringInput(const char *szPromt , char *szBuffer);

//*****
void InitTest()
{
    //SDK initialization.
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // get SDK version info
    // this operation is optional
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // set CLIENT_SetAutoReconnect interface.Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally.
    // this operation is optional, but recommended
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

    // set connecting device timeout time and attempts
    // this operation is optional

    int nWaitTime = 5000;    //Timeout value is 5 seconds.

    int nTryTimes = 3;        //If timeout,it will try to log in three times.

    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    //Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
```

```
//This operation is optional.
NET_PARAM stuNetParm = {0};
stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);

NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_ILoginHandle)
{
    //Login device
    g_ILoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if(0 == g_ILoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhnetsdk.h.
        // It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        // Example: #define NET_NOT_SUPPORTED_EC(23)
        // Current SDK does not support this function, error code is 0x80000017, Decimal
        // number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
    }
    else
    {
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
    }
    // The first time logging to device, some data is needed to be initialized to enable
    // normal business function. It is recommended to wait for a while after login, the waiting time varies
    // by devices.
    Sleep(1000);
    printf("\n");
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
}
```

```
if (0 == g_ILLoginHandle)
{
    return;
}

//Set snapshot callback function.
CLIENT_SetSnapRevCallBack(SnapRev, NULL);

//Send snapshot command to front-end device.
// Snapshot channel defaults 0, and mode defaults" request one frame".
int    nChannelId = 0;
int    nSnapType = 0;// Snapshot mode;0:request one frame,1:send out request
regularly,2: Request consecutively

    SNAP_PARAMS stuSnapParams;
    stuSnapParams.Channel = nChannelId;
    stuSnapParams.mode = nSnapType;
    stuSnapParams.CmdSerial = ++g_nCmdSerial; //request SN, valid range
0~65535,over this range will be cut to unsigned short
    if (FALSE == CLIENT_SnapPictureEx(g_ILLoginHandle, &stuSnapParams))
    {
        printf("CLIENT_SnapPictureEx Failed!Last Error[%x]\n",
CLIENT_GetLastError());
        return;
    }
    else
    {
        printf("CLIENT_SnapPictureEx succ\n");
    }

    GetStringInput("'q': exit; 'c': continue\n", szUserChoose);
}while('q' != szUserChoose[0]);

return;
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();

    //Logout device
    if (0 != g_ILLoginHandle)
```

```
{
    if(FALSE == CLIENT_Logout(g_ILoginHandle))
    {
        printf("CLIENT_Logout Failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
    else
    {
        g_ILoginHandle = 0;
    }
}
// Clean up initialization resources.
if (TRUE == g_bNetSDKInitFlag)
{
    CLIENT_Cleanup();
    g_bNetSDKInitFlag = FALSE;
}

exit(0);
}

int main()
{
    InitTest();

    RunTest();

    EndTest();

    return 0;
}

//*****
// commonly used callback set declaration
void CALLBACK DisConnectFunc(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
}
```

```
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK SnapRev(LLONG ILoginID, BYTE *pBuf, UINT RevLen, UINT EncodeType,
DWORD CmdSerial, LDWORD dwUser)
{
    printf("[SnapRev] -- receive data!\n");
    if(ILoginID == g_ILoginHandle)
    {
        if (NULL != pBuf && RevLen > 0)
        {
            char szPicturePath[256] = "";
            time_t stuTime;
            time(&stuTime);
            char szTmpTime[128] = "";
            strftime(szTmpTime, sizeof(szTmpTime) - 1, "%y%m%d_%H%M%S",
gmtime(&stuTime));
            _snprintf(szPicturePath, sizeof(szPicturePath)-1, "%d_%s.jpg", CmdSerial,
szTmpTime);

            FILE* pFile = fopen(szPicturePath, "wb");
            if (NULL == pFile)
            {
                return;
            }

            int nWrite = 0;
            while(nWrite != RevLen)
            {
```



```
        nWrite += fwrite(pBuf + nWrite, 1, RevLen - nWrite, pFile);
    }

    fclose(pFile);
}

}

}

//*****
// commonly used function set declaration
int GetIntInput(char *szPromt, int& nError)
{
    long int nGet = 0;
    char* pError = NULL;
    printf(szPromt);
    char szUserInput[32] = "";
    gets(szUserInput);
    nGet = strtol(szUserInput, &pError, 10);
    if ('\0' != *pError)
    {
        //input parameter error
        nError = -1;
    }
    else
    {
        nError = 0;
    }

    return nGet;
}

void GetStringInput(const char *szPromt , char *szBuffer)
{
    printf(szPromt);
    gets(szBuffer);
}
```

## 2.9 Report Alarm

### 2.9.1 Intro

Report alarm, when front-end device detects special event set previously, send alarm to platform-end and notify the platform. The platform may receive external alarm、video signal lost alarm、tampering alarm and motion detection alarm uploaded by device.

The method of alarm report is that SDK actively connects device and subscribes alarm function from device. When device detects alarm event, it will immediately send the event to SDK.

### 2.9.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for initialization
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources
<a href="#">CLIENT_LoginEx2</a>	Extensive interface for sync login
<a href="#">CLIENT_SetDVRMessCallBack</a>	Interface for setting alarm callback function
<a href="#">CLIENT_StartListenEx</a>	Extensive interface for subscribing alarm event from device
<a href="#">CLIENT_StopListen</a>	Interface for stopping subscribing alarm
<a href="#">CLIENT_Logout</a>	Interface for logout device
<a href="#">CLIENT_GetLastError</a>	Interface for getting error code after failed calling interface

### 2.9.3 Process Instruction

Report alarm process flow sheet is shown in Figure 2-13.

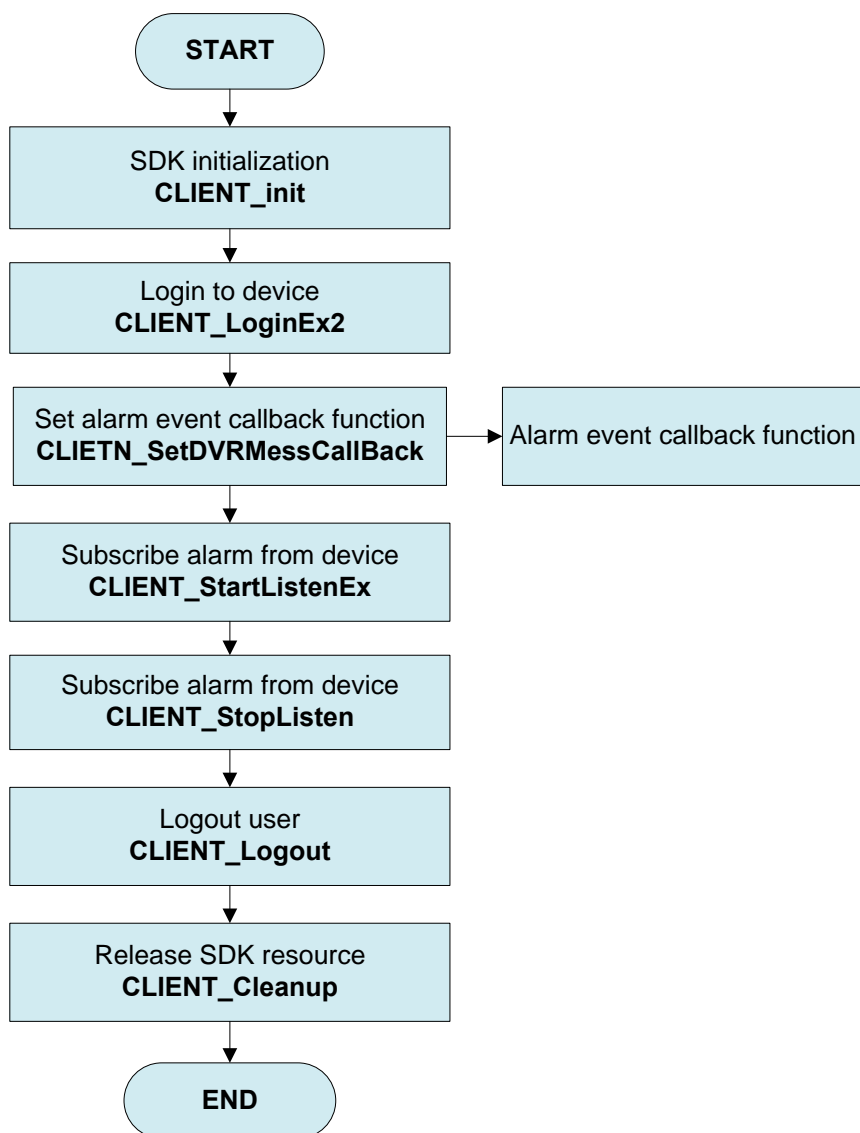


Figure 2-13 Report alarm process flow sheet

**Process discription:**

1. SDK initialization
2. After successful initialization, call CLIENT\_LoginEx2 to login device.
3. Call CLIENTN\_SetDVRMessCallBack to set alarm event callback function, this interface shall be called before alarm subscription.
4. Call CLIENT\_StartListenEx to subscribe alarm from device. After successful subscription, alarm event reported by device is sent to user via callback function set in CLIENTN\_SetDVRMessCallBack.
5. After alarm report function is finished, call CLIENT\_StopListen to stop device alarm

subscription.

6. Call CLIENT\_Logout to logout device.
7. Last but not the least, call CLIENT\_Cleanup to release SDK resources.

## 2.9.4 Example Code

```
#include<windows.h>
#include <stdio.h>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static char g_szDevIp[32] = "172.23.2.66";
static WORD g_nPort = 37777;
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
static BOOL g_bStartListenFlag = FALSE;

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

// alarm event callback function
```

// It's not recommended to call SDK interfaces in this callback function

// set this callback function in CLIENT\_SetDVRMessCallBack. When receiving alarm event reported by device, SDK will call this function

BOOL CALLBACK MessCallBack(LONG ICommand, LLONG ILoginID, char \*pBuf, DWORD dwBufLen, char \*pchDVRIP, LONG nDVRPort, LDWORD dwUser);

```
//***** void InitTest()
```

```
{
```

```
    // SDK initialization
```

```
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
```

```
    if (FALSE == g_bNetSDKInitFlag)
```

```
    {
```

```
        printf("Initialize client SDK fail; \n");
```

```
        return;
```

```
    }
```

```
    else
```

```
    {
```

```
        printf("Initialize client SDK done; \n");
```

```
    }
```

```
    // get SDK version info
```

```
    // this operation is optional
```

```
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
```

```
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);
```

// set CLIENT\_SetAutoReconnect interface. Once HaveReConnect is set, when device gets offline, SDK will reconnect internally.

```
// this operation is optional, but recommended
```

```
CLIENT_SetAutoReconnect(&HaveReConnect, 0);
```

```
// set connecting device timeout time and attempts
```

```
// this operation is optional
```

```
int nWaitTime = 5000; // timeout time is set to 5s
```

```
int nTryTimes = 3; // if timeout, you have 3 attempts
```

```
CLIENT_SetConnectTime(nWaitTime, nTryTimes);
```

//Set more network parameter, nWaittime and nConnectTryNum in NET\_PARAM has the same meaning with the timeout time and trial time set in interface CLIENT\_SetConnectTime.

```
//This operation is optional.
```

```
NET_PARAM stuNetParm = {0};
```

```
stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
```

```
CLIENT_SetNetworkParam(&stuNetParm);

NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_LoginHandle)
{
    // login device
    g_LoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);

    if (0 == g_LoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhnetsdk.h.
        // It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        // example:
        // #define NET_NOT_SUPPORTED_EC(23)
        // current SDK does not support the function, corresponding error code is
        // 0x80000017. Decimal number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
CLIENT_GetLastError());
    }
    else
    {
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
    }
    // The first time logging to device, some data is needed to be initialized to enable
    // normal business function. It is recommended to wait for a while after login, the waiting time varies
    // by devices.
    Sleep(1000);
    printf("\n");
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_LoginHandle)
```

```
{
    return;
}

// set alarm event callback function
CLIENT_SetDVRMessCallBack(MessCallBack , NULL);

// subscribe alarm from device
if( TRUE == CLIENT_StartListenEx(g_ILoginHandle))
{
    g_bStartListenFlag = TRUE;
    printf("CLIENT_StartListenEx Success!\nJust Wait Event....\n");
}
else
{
    printf("CLIENT_StartListenEx Failed!Last Error[%x]\n" , CLIENT_GetLastError());
}
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // stop subscribing alarm from device
    if (TRUE == g_bStartListenFlag)
    {
        if (FALSE == CLIENT_StopListen(g_ILoginHandle))
        {
            printf("CLIENT_StopListen Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_bStartListenFlag = FALSE;
        }
    }
}
```

```
// logout device
if (0 != g_ILLoginHandle)
{
    if(FALSE == CLIENT_Logout(g_ILLoginHandle))
    {
        printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
    }
    else
    {
        g_ILLoginHandle = 0;
    }
}

// cleanup initialization resource
if (TRUE == g_bNetSDKInitFlag)
{
    CLIENT_Cleanup();
    g_bNetSDKInitFlag = FALSE;
}

return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//*****
// commonly used callback definition set
void CALLBACK DisConnectFunc(LLONG ILLoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{

```



```
printf("Call DisconnectFunc\n");
printf("ILoginID[0x%x]", ILoginID);
if (NULL != pchDVRIP)
{
    printf("pchDVRIP[%s]\n", pchDVRIP);
}
printf("nDVRPort[%d]\n", nDVRPort);
printf("dwUser[%p]\n", dwUser);
printf("\n");
}
```

```
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
```

```
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```

```
BOOL CALLBACK MessCallBack(LONG ICommand, LLONG ILoginID, char *pBuf, DWORD
dwBufLen, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
```

```
{
    printf("[MessCallBack] -- Get Event IP[%s] , port[%d]\n", pchDVRIP , nDVRPort);
```

// only part of alarm processing methods is listed in the demo, user can deal with corresponding alarm event info accordingly, please refer to related event explanation in header file dhnetsdk.h for details.

```
switch(ICommand)
{
```

```
case DH_ALARM_ALARM_EX:
{
    printf("\n external alarm\n");
    if (NULL != pBuf)
    {
        BYTE* pInfo = (BYTE*)pBuf;
        for(unsigned int i = 0; i < dwBufLen/sizeof(BYTE); ++i)
        {
            printf("nChannelID = [%2d], state = [%d]\n", i, *(pInfo + i));
        }
    }
}
break;

case DH_MOTION_ALARM_EX:
{
    printf("\n motion detection alarm\n");
    if (NULL != pBuf)
    {
        BYTE* pInfo = (BYTE*)pBuf;
        for(unsigned int i = 0; i < dwBufLen/sizeof(BYTE); ++i)
        {
            printf("nChannelID = [%2d], state = [%d]\n", i, *(pInfo + i));
        }
    }
}
break;

case DH_ALARM_ALARM_EX_REMOTE:
{
    printf("\n remote external alarm \n");
    if (NULL != pBuf)
    {
        ALARM_REMOTE_ALARM_INFO* pInfo =
(ALARM_REMOTE_ALARM_INFO *)pBuf;
        printf("nChannelID = %d\n", pInfo->nChannelID);
        printf("nState = %d\n", pInfo->nState);
    }
}
```

```
        }
    }
    break;
case DH_ALARM_ACCESS_CTL_EVENT:
    {
        printf("\n access control event \n");
        if (NULL != pBuf)
        {
            ALARM_ACCESS_CTL_EVENT_INFO* pInfo =
(ALARM_ACCESS_CTL_EVENT_INFO *)pBuf;
            printf(" unlock method = %d\n", pInfo->emOpenMethod);
            printf(" card NO. = [%s]\n", pInfo->szCardNo);
        }
    }
    break;
default:
    printf("\n[MessCallBack] – other alarm Get ICommand = 0x%x\n", ICommand);
    break;
}
return TRUE;
}
```

## 2.10 Device Search

### 2.10.1 Intro

Device search is mainly used to help user to get device info from network. Device search can work with login function. Device search interface can find relevant devices and login interface can

login these devices.

Device search is classified into the following two types by whether crossing segment or not:

- Async same -segment device search

Search for device info within current segment.

- Sync cross-segment device search

According to user-set segment info, searching for device in corresponding segment.

## 2.10.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for initialization
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources
<a href="#">CLIENT_StartSearchDevices</a>	Interface for async searching for IPC, NVS and etc. within same segment
<a href="#">CLIENT_StopSearchDevices</a>	Interface for stopping async search for IPC, NVS and etc. within same segment
<a href="#">CLIENT_SearchDevicesByIPs</a>	Interface for sync searching cross-segment device
<a href="#">CLIENT_GetLastError</a>	Interface for getting error code after failed calling interface

## 2.10.3 Process Intruction

### 2.10.3.1 Async Search for Same-segment Device

Async search for device within the same segment is shown in Figure 2-14.

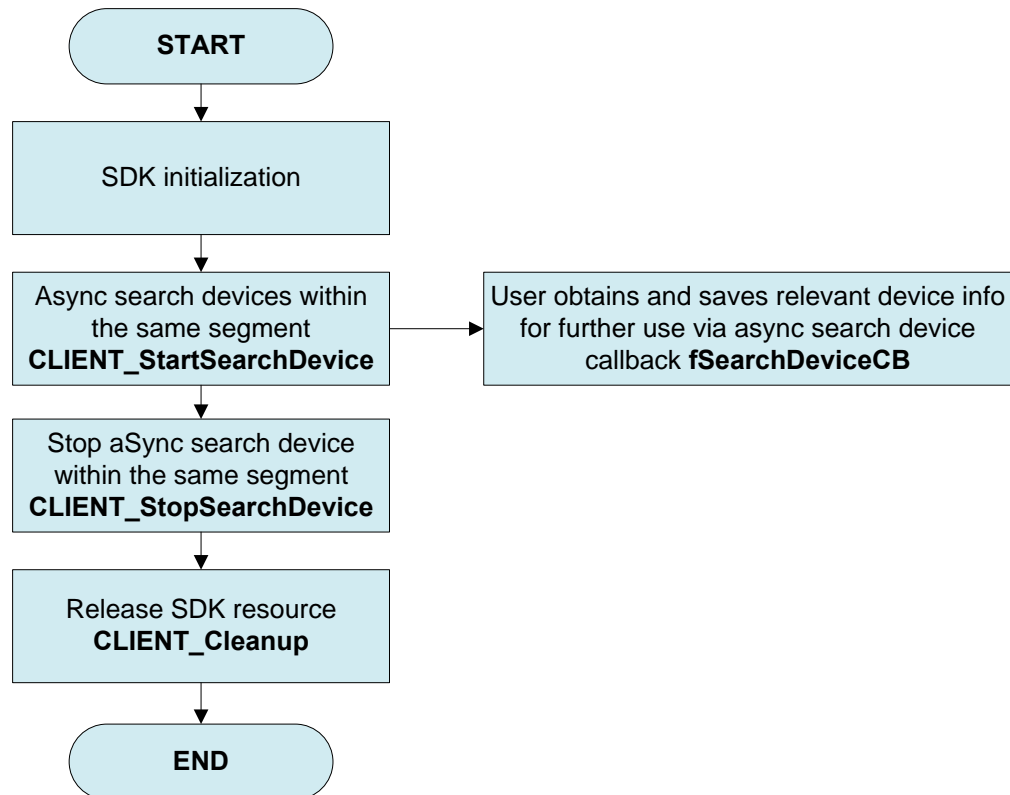


Figure 2-14 Async search for devices within the same segment

**Process discription:**

1. SDK initialization

2. Call **CLIENT\_StartSearchDevices** to async search for devices within the same segment.

User gets the obtained device info via callback function **fSearchDevicesCB** which is set in this interface. The search operation has no timeout time, user needs to stop searching by calling interface **CLIENT\_StopSearchDevices**.

3. Call **CLIENT\_StopSearchDevices** to stop async searching device within the same segment.

4. When SDK function use is done, call **CLIENT\_Cleanup** to release SDK resource.

### 2.10.3.2 Sync Search for Cross-segment Device

Sync search for cross-segment device is shown in Figure 2-15.

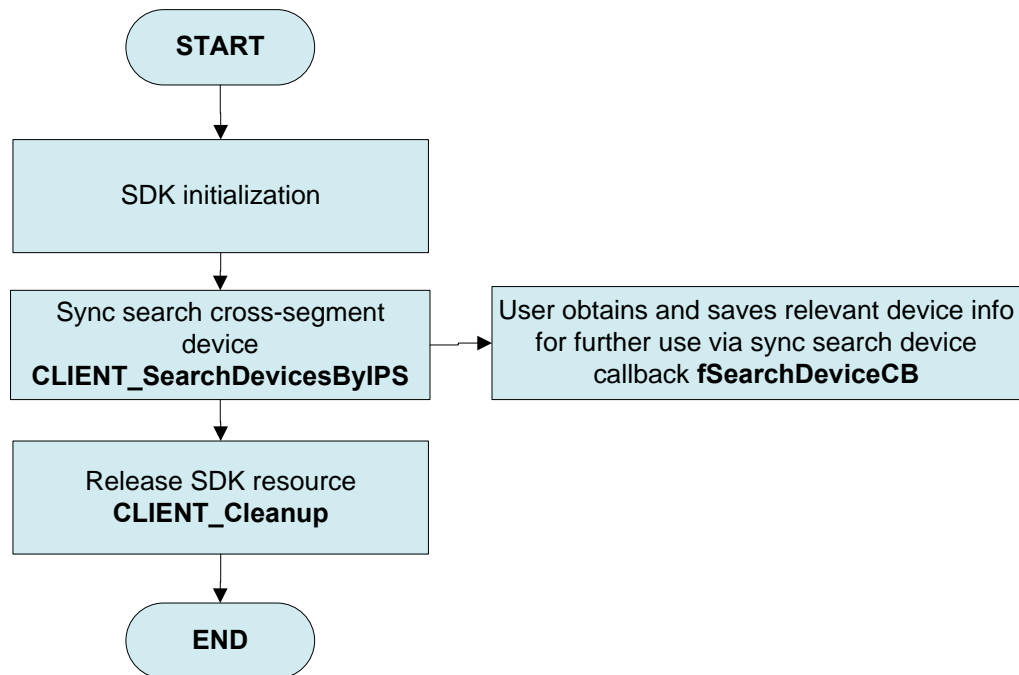


Figure 2-15 Sync search for cross-segment device

**Process description:**

1. SDK initialization.
2. Call CLIENT\_SearchDevicesByIPs to sync search for device crossing segment. User gets the obtained device info via callback function fSearchDevicesCB which is set in this interface. Only when searching time is out or searching all the devices cross the segment , the interface return. User can decide the timeout time according to net condition.
3. Call CLIENT\_Cleanup to release SDK resource.

## 2.10.4 Example Code

### 2.10.4.1 Async Search for Same-segment Device

```
#include<windows.h>
#include <stdio.h>
#include <list>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")
```

```
static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lSearchHandle = 0L;
static CRITICAL_SECTION g_mDeviceListLock;          // device list operation lock
static std::list<DEVICE_NET_INFO_EX> g_lDeviceList;  // device list

//*****

// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

// async search for device callback
// It's not recommended to call SDK interfaces in this callback function
// set this callback function in CLIENT_StartSearchDevices/CLIENT_SearchDevicesByIPs.
SDK will call this function when device is found.
void CALLBACK SearchDevicesCB(DEVICE_NET_INFO_EX *pDevNetInfo, void* pUserData);

//*****

void InitTest()
{
    // initialization thread lock
    InitializeCriticalSection(&g_mDeviceListLock);

    // initialization SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
```

```
{
    printf("Initialize client SDK fail; \n");
    return;
}
else
{
    printf("Initialize client SDK done; \n");
}

    // get SDK version info
    // this operation is optional
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // set CLIENT_SetAutoReconnect interface. Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally.
    // this operation is optional, but recommended
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

    // set connecting device timeout time and attempts
    // this operation is optional
    int nWaitTime = 5000;    // timeout time is set to 5s
    int nTryTimes = 3;      // if timeout, you have 3 attempts
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    //Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
    //This operation is optional.
    NET_PARAM stuNetParm = {0};
    stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
    CLIENT_SetNetworkParam(&stuNetParm);
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
}
```



```
// start async search for same-segment device
g_ISearchHandle = CLIENT_StartSearchDevices(SearchDevicesCB, &g_IDeviceList);
if (NULL == g_ISearchHandle)
{
    printf("CLIENT_StartSearchDevices Failed! Last Error[%x]\n",
CLIENT_GetLastError());
    return;
}
int nIndex = 0;
int nSearchTime = 0;
int nSearchLimit = 10;// search  lasts for 10s, user can modify this value according to
network condition
while (TRUE)
{
    if (nSearchLimit == nSearchTime++)
    {
        break;
    }

    EnterCriticalSection(&g_mDeviceListLock);
    for (std::list<DEVICE_NET_INFO_EX>::iterator iter = g_IDeviceList.begin(); iter !=
g_IDeviceList.end(); )
    {
        printf("\n***** find device *****\n");
        printf("nIndex[%d]\n", ++nIndex);
        printf("iIPVersion[%d]\n", iter->iIPVersion);
        printf("szIP[%s]\n", iter->szIP);
        printf("nPort[%d]\n", iter->nPort);
        g_IDeviceList.erase(iter);
        iter = g_IDeviceList.begin();
    }
    LeaveCriticalSection(&g_mDeviceListLock);

    Sleep(1000);
}
```

```
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // cleanup thread lock resource
    DeleteCriticalSection(&g_mDeviceListLock);

    // stop async search for same-segment devices
    if (NULL != g_lSearchHandle)
    {
        if (FALSE == CLIENT_StopSearchDevices(g_lSearchHandle))
        {
            printf("CLIENT_StopSearchDevices Failed!Last Error[%x]\n",
CLIENT_GetLastError());
        }
    }

    // cleanup initialization resource
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

int main()
{
    InitTest();

    RunTest();

    EndTest();
}
```

```
        return 0;
    }

    void CALLBACK DisConnectFunc(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
    DWORD dwUser)
    {
        printf("Call DisConnectFunc\n");
        printf("ILoginID[0x%x]", ILoginID);
        if (NULL != pchDVRIP)
        {
            printf("pchDVRIP[%s]\n", pchDVRIP);
        }
        printf("nDVRPort[%d]\n", nDVRPort);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
    }

    void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
    LDWORD dwUser)
    {
        printf("Call HaveReConnect\n");
        printf("ILoginID[0x%x]", ILoginID);
        if (NULL != pchDVRIP)
        {
            printf("pchDVRIP[%s]\n", pchDVRIP);
        }
        printf("nDVRPort[%d]\n", nDVRPort);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
    }

    void CALLBACK SearchDevicesCB(DEVICE_NET_INFO_EX *pDevNetInfo, void* pUserData)
    {
        if ((NULL == pDevNetInfo) || (NULL == pUserData))
        {
```

```
        printf("warming param is null\n");
        return;
    }

    std::list<DEVICE_NET_INFO_EX>* pDeviceList =
(std::list<DEVICE_NET_INFO_EX>*)pUserData;
    EnterCriticalSection(&g_mDeviceListLock);
    pDeviceList->push_back(*pDevNetInfo);
    LeaveCriticalSection(&g_mDeviceListLock);
    return;
}
```

#### 2.10.4.2 Sync Search for Cross-segment Device

```
#include<windows.h>
#include <stdio.h>
#include <list>
#include "dhnetsdk.h"

#pragma comment(lib , "dhnetsdk.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static std::list<DEVICE_NET_INFO_EX> g_IDeviceList;    // device list

// *****get local IP interface
std::string GetLocalIpAddress();

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
automatically
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);

// callback function used after device is reconnected successfully
```

```
// It's not recommended to call SDK interfaces in this callback function
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);

// sync search for device callback
// It's not recommended to call SDK interfaces in this callback function
// set this callback function in CLIENT_StartSearchDevices/CLIENT_SearchDevicesByIPs.
SDK will call this function when device is found.
void CALLBACK SearchDevicesCB(DEVICE_NET_INFO_EX *pDevNetInfo, void* pUserData);

//*****
void InitTest()
{
    // SDK initialization
    g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }

    // get SDK version info
    // this operation is optional
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // set CLIENT_SetAutoReconnect interface. Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally.
    // this operation is optional, but recommended
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

    // set connecting device timeout time and attempts
    // this operation is optional
    int nWaitTime = 5000; // timeout time is set to 5s
```

```
int nTryTimes = 3;      // if timeout, you have 3 attempts
CLIENT_SetConnectTime(nWaitTime, nTryTimes);

//Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
//This operation is optional.
NET_PARAM stuNetParm = {0};
stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
    // start sync search for cross-segment device
    char szLocalIp[64] = "";
    strncpy(szLocalIp, GetLocalIpAddress().c_str(), sizeof(szLocalIp) - 1);

    DEVICE_IP_SEARCH_INFO stuTmp = {sizeof(stuTmp)};
    stuTmp.nIpNum = 256; // valid IP address number
    for (unsigned int i = 0; i < stuTmp.nIpNum; ++i)
    {
        // user needs to ensure IP address validity and network connection. Here the
        searching segment is 172.11.1.xxx, and local IP address is 10.34.3.77
        _snprintf(stuTmp.szIP[i], sizeof(stuTmp.szIP[i]) - 1, "172.11.1.%d", i);
    }

    DWORD dwWaitTime = 5000;

    // please note that this interface will return until timeout, user shall decide timeout time
    according to net condition
    if (FALSE == CLIENT_SearchDevicesByIPs(&stuTmp, SearchDevicesCB,
    (DWORD)&g_IDeviceList, szLocalIp, dwWaitTime))
    {
        printf("CLIENT_SearchDevicesByIPs Failed! Last Error[%x]\n",
```

```
CLIENT_GetLastError());
    return;
}

int nIndex = 0;
for (std::list<DEVICE_NET_INFO_EX>::iterator iter = g_IDeviceList.begin(); iter !=
g_IDeviceList.end(); ++iter)
{
    printf("\n***** find device *****\n");
    printf("nIndex[%d]\n", ++nIndex);
    printf("ilPVersion[%d]\n", iter->ilPVersion);
    printf("szIP[%s]\n", iter->szIP);
    printf("nPort[%d]\n", iter->nPort);
}
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();

    // cleanup initialization resource
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

int main()
{
    InitTest();

    RunTest();
}
```

```
EndTest();

return 0;
}

//*****
//commonly used callback set definition
void CALLBACK DisConnectFunc(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
```



```
void CALLBACK SearchDevicesCB(DVICE_NET_INFO_EX *pDevNetInfo, void* pUserData)
{
    if ((NULL == pDevNetInfo) || (NULL == pUserData))
    {
        printf("warming param is null\n");
        return;
    }

    std::list<DVICE_NET_INFO_EX>* pDeviceList =
(std::list<DVICE_NET_INFO_EX>*)pUserData;
    pDeviceList->push_back(*pDevNetInfo);
    return;
}

// ***** get local IP interface
std::string GetLocalIpAddress()
{
    WSADATA wsaData;
    if (0 != WSStartup(MAKEWORD(2,2), &wsaData))
    {
        return "";
    }

    char local[255] = "";
    gethostname(local, sizeof(local));
    hostent* ph = gethostbyname(local);
    if (NULL == ph)
    {
        return "";
    }

    in_addr addr;
    memcpy(&addr, ph->h_addr_list[0], sizeof(in_addr));

    std::string localIP(inet_ntoa(addr));
}
```

```
WSACleanup();  
return localIP;  
}
```

## 2.11 Intelligent Event Report and Snapshot

### 2.11.1 Intro

Intelligent event report is that devices judge whether to report events and to send pictures to users or not according to real-time stream intelligent analyzation and intelligent event trigger rules configured by users. Intelligent events are: scene change, cross picket line, enter picket zone, leave picket zone, in picket zone, across enclosure, straggle detection, carry-over detection, move detection, goods protection, illegal parking, fast moving, go in the wrong direction etc.

Intelligent snapshot event: user manually sends a command to device after subscribing event successfully. Device snapshots picture of current scene and reports it to user via intelligent event.

### 2.11.2 Interface Overview

Interface	Interface Description
<a href="#">CLIENT_Init</a>	Interface for initialization
<a href="#">CLIENT_Cleanup</a>	Interface for cleaning up SDK resources
<a href="#">CLIENT_LoginEx2</a>	Extensive interface 2 for sync login
<a href="#">CLIENT_RealLoadPictureEx</a>	Interface for intelligent picture alarm subscription
<a href="#">CLIENT_ControlDeviceEx</a>	Extensive interface for device control
<a href="#">CLIENT_Logout</a>	Interface for logout device
<a href="#">CLIENT_GetLastError</a>	Interface for getting error code after failed calling interface

### 2.11.3 Process Intruction

Intelligent event report and snapshot process flow sheet is shown in Figure 2-16.

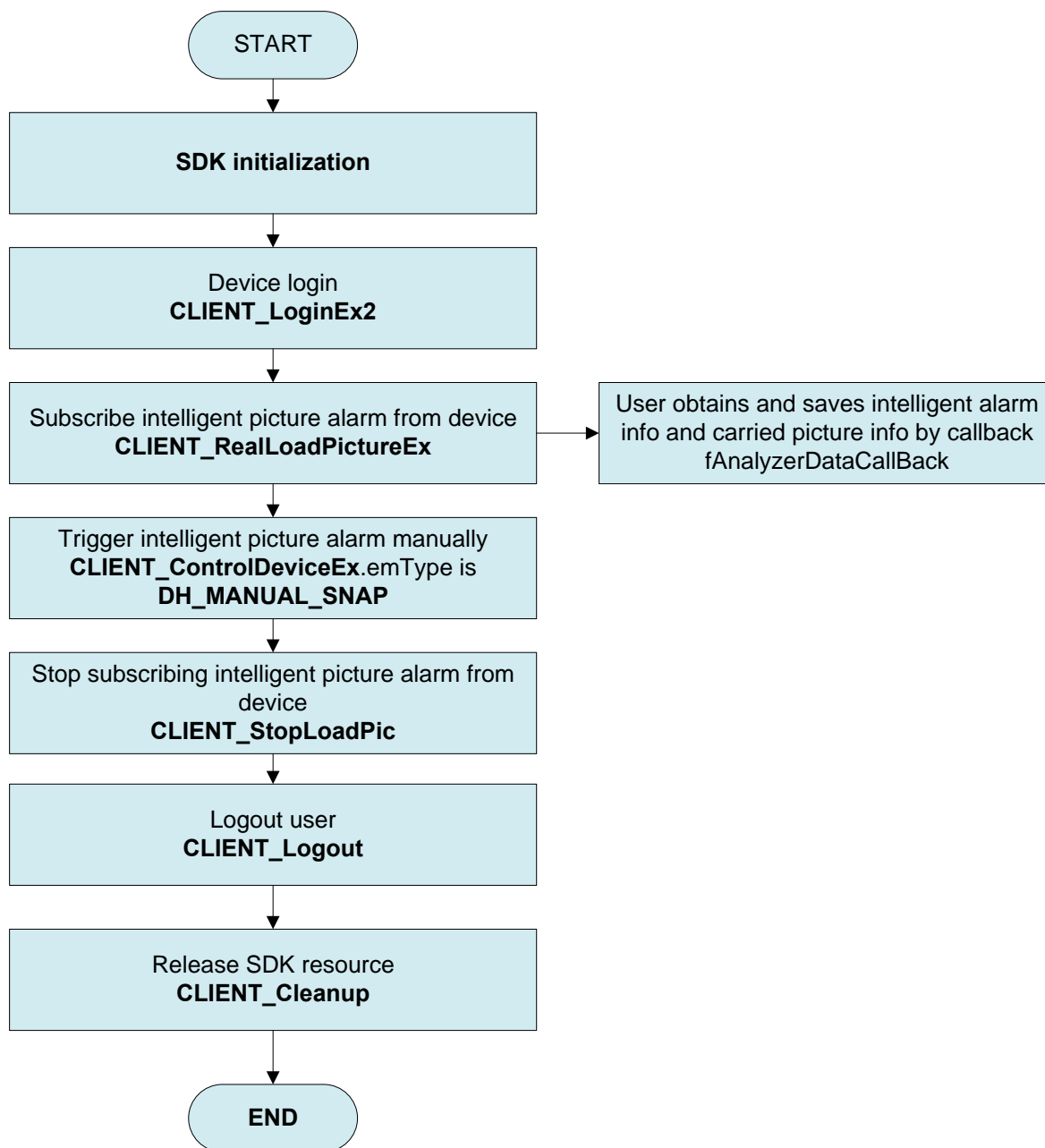


Figure 2-16 intelligent event report and snapshot

**Process discription:**

1. SDK initialization

2. After successful initialization, call `CLIENT_LoginEx2` to login device.

3. Call `CLIENT_RealLoadPictureEx` to subscribe intelligent snapshot alarm from device. After successful subscription, the intelligent snapshot alarm event reported by device will be sent to user by callback function `fAnalyzerDataCallBack`. In callback function, user should converts input character to corresponding struct according to the instruction in SDK header files, and then displays

and saves event as needed. Due to SDK receiving buffer is 2M by default, when callback picture info exceeds 2M, user needs to call CLIENT\_SetNetworkParam interface to set receiving buffer again, otherwise SDK will abandon data pack over 2M.

4. If a user wants to manually trigger intelligent picture alarm, he can call CLIENT\_ControlDeviceEx interface with parameter emType DH\_MANUAL\_SNAP. SDK will send command to device, device then snapshots current monitoring video and reports it to user.

5. After intelligent picture alarm report function is done, call CLIENT\_StopLoadPic to stop subscribing intelligent picture alarm from device.

6. After business use is done, call CLIENT\_Logout to logout device.

7. After SDK function use is done, call CLIENT\_Cleanup to release SDK resource.

## 2.11.4 Example Code

```
#include<windows.h>
#include <stdio.h>
#include <list>
#include <time.h>
#include "dhnetSDK.h"

#pragma comment(lib, "dhnetSDK.lib")

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static LLONG g_lRealLoadHandle = 0L;
static char g_szDevIp[32] = "192.168.4.12";
static int g_nPort = 37777;
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*****
// commonly used callback set declaration
// callback function used when device gets offline
// It's not recommended to call SDK interfaces in this callback function
//Set the callback function in CLIENT_Init, when device gets offline, SDK will call this function
```

automatically

```
void CALLBACK DisConnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser);
```

```
// callback function used after device is reconnected successfully
```

```
// It's not recommended to call SDK interfaces in this callback function
```

```
// Set the callback function in CLIENT_SetAutoReconnect, when offline device is reconnected
successfully, SDK will call the function
```

```
void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser);
```

```
// intelligent analyzing data callback
```

```
// It's not recommended to call SDK interfaces in this callback function
```

```
// set this callback function in CLIENT_RealLoadPictureEx/CLIENT_RealLoadPicture, SDK
will call this function when device-end has intelligent picture event to report.
```

```
// nSequence is used when uploading the same picture. 0 means it is the first time to appear; 2
means it is the last time to appear or only appear once; 1 means it will appear again later
```

```
// int nState =* (int*) reserved means current callback data status. 0 means real-time data; 1
means offline data; 2 means offline transmission done.
```

```
// return value is abolished, no special meaning
```

```
// due to SDK receiving buffer is 2M by default, when callback picture info exceed 2M, user need
to call CLIENT_SetNetworkParam interface to set receiving buffer again, otherwise SDK will
abandon data pack over 2M
```

```
int CALLBACK AnalyzerDataCallBack(LLONG IAnalyzerHandle, DWORD dwAlarmType,
void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void
*reserved);
```

```
//*****
```

```
void InitTest()
```

```
{
```

```
    // SDK initialization
```

```
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
```

```
    if (FALSE == g_bNetSDKInitFlag)
```

```
    {
```

```
        printf("Initialize client SDK fail; \n");
```

```
        return;
```

```
    }
```

```
    else
```

```
    {
```

```
        printf("Initialize client SDK done; \n");
```

```
}

// get SDK version info
// this operation is optional
DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
printf("NetSDK version is [%d]\n", dwNetSdkVersion);


// set CLIENT_SetAutoReconnect interface. Once HaveReConnect is set, when device
gets offline, SDK will reconnect internally.
// this operation is optional, but recommended
CLIENT_SetAutoReconnect(&HaveReConnect, 0);


// set connecting device timeout time and attempts
// this operation is optional
int nWaitTime = 5000;    // timeout time is set to 5s
int nTryTimes = 3;      // if timeout, you have 3 attempts
CLIENT_SetConnectTime(nWaitTime, nTryTimes);


//Set more network parameter, nWaittime and nConnectTryNum in NET_PARAM has the
same meaning with the timeout time and trial time set in interface CLIENT_SetConnectTime.
//This operation is optional.
NET_PARAM stuNetParm = {0};
stuNetParm.nWaittime = 3000; // when login, connection timeout value is 3000ms.
CLIENT_SetNetworkParam(&stuNetParm);


NET_DEVICEINFO_Ex stDevInfoEx = {0};
int nError = 0;
while(0 == g_ILoginHandle)
{
    // login device
    g_ILoginHandle = CLIENT_LoginEx2(g_szDevIp, g_nPort, g_szUserName,
g_szPasswd, EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfoEx, &nError);


    if (0 == g_ILoginHandle)
    {
        // according to error code, you can find corresponding explanation in dhSDK.h. It is
to print hexadecimal here, not decimal shows in header file, please be careful with conversion
        // example:
        // #define NET_NOT_SUPPORTED_EC(23)
        // current SDK does not support the function, corresponding error code is 0x80000017.
        Decimal number 23 is hexadecimal 0x17.
        printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevIp, g_nPort,
```

```
CLIENT_GetLastError());
    }
    else
    {
        printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevIp, g_nPort);
    }
    // The first time logging to device, some data is needed to be initialized to enable
normal business function. It is recommended to wait for a while after login, the waiting time varies
by devices.
    Sleep(1000);
    printf("\n");
}
}

void RunTest()
{
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }

    if (0 == g_ILoginHandle)
    {
        return;
    }

    // subscribe intelligent picture alarm
    LDWORD dwUser = 0;
    Int nChannel = 0;
    // each setup corresponds to one channel, and corresponds to event of a certain type
    // if a user wants to set all types of event for one channel, the parameter dwAlarmType
should be set to EVENT_IVS_ALL
    // If you want to set that one channel uploads two events, please call
CLIENT_RealLoadPictureEx twice and set different event type
    g_IRealLoadHandle = CLIENT_RealLoadPictureEx(g_ILoginHandle, nChannel,
EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, dwUser, NULL);
    if (0 == g_IRealLoadHandle)
```

```
{
    printf("CLIENT_RealLoadPictureEx Failed! Last Error[%x]\n",
CLIENT_GetLastError());
    return;
}

// manually snapshot trigger intelligent picture alarm function
while(1)
{
    char szGetBuf[64] = "";
    printf("manual snap, 'q': quit, other: yes\n");
    gets(szGetBuf);
    // enter 'q' to exit manual snapshot trigger alarm, others mean to trigger alarm
    if (0 == strncmp(szGetBuf, "q", sizeof(szGetBuf) - 1))
    {
        break;
    }

    MANUAL_SNAP_PARAMETER stuSanpParam = {0};
    stuSanpParam.nChannel = 0;
    strncpy((char*)stuSanpParam.bySequence, "just for test",
sizeof(stuSanpParam.bySequence));

    // manually snapshot trigger alarm function, this function is only valid for ITC device
    if (FALSE == CLIENT_ControlDeviceEx(gILoginHandle, DH_MANUAL_SNAP,
&stuSanpParam))
    {
        printf("CLIENT_ControlDeviceEx Failed! Last Error[%x]\n",
CLIENT_GetLastError());
        break;
    }
}

void EndTest()
{
```



```
printf("input any key to quit!\n");
getchar();

// stop subscribing picture alarm
if (0 != g_IRealLoadHandle)
{
    if (FALSE == CLIENT_StopLoadPic(g_IRealLoadHandle))
    {
        printf("CLIENT_StopLoadPic Failed!Last Error[%x]\n", CLIENT_GetLastError());
    }
    else
    {
        g_IRealLoadHandle = 0;
    }
}

// logout device
if (0 != g_ILLoginHandle)
{
    if (FALSE == CLIENT_Logout(g_ILLoginHandle))
    {
        printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
    }
    else
    {
        g_ILLoginHandle = 0;
    }
}

// cleanup initialization resource
if (TRUE == g_bNetSDKInitFlag)
{
    CLIENT_Cleanup();
    g_bNetSDKInitFlag = FALSE;
```

```
    }
    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}

//*****
// commonly used callback set definition
void CALLBACK DisConnectFunc(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort,
LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("ILoginID[0x%x]", ILoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
}
```

```
    }

    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}

int CALLBACK AnalyzerDataCallBack(LLONG IAnalyzerHandle, DWORD dwAlarmType,
void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void
*reserved)
{
    if (IAnalyzerHandle != g_IRealLoadHandle)
    {
        return 0;
    }

    int nAlarmChn = 0;
    switch(dwAlarmType)
    {
        case EVENT_IVS_TRAFFIC_OVERLINE:
        {
            printf("EVENT_IVS_TRAFFIC_OVERLINE event\n");
            DEV_EVENT_TRAFFIC_OVERLINE_INFO* pStuInfo =
(DEV_EVENT_TRAFFIC_OVERLINE_INFO*)pAlarmInfo;
            nAlarmChn = pStuInfo->nChannelID;
            printf("nChannelID[%d]\n", pStuInfo->nChannelID);
        }
        break;
        case EVENT_IVS_PARKINGDETECTION:
        {
            printf("EVENT_IVS_PARKINGDETECTION event\n");
            DEV_EVENT_PARKINGDETECTION_INFO* pStuInfo =
(DEV_EVENT_PARKINGDETECTION_INFO*)pAlarmInfo;
            nAlarmChn = pStuInfo->nChannelID;
            printf("nChannelID[%d]\n", pStuInfo->nChannelID);
        }
    }
}
```

```
        break;
    case EVENT_IVS_TRAFFIC_MANUALSNAP:
    {
        printf("EVENT_IVS_TRAFFIC_MANUALSNAP event\n");
        DEV_EVENT_TRAFFIC_MANUALSNAP_INFO* pStuInfo =
(DEV_EVENT_TRAFFIC_MANUALSNAP_INFO*)pAlarmInfo;
        nAlarmChn = pStuInfo->nChannelID;
        // pStuInfo->szManualSnapNo should be "just for test"
        printf("nChannelID[%d]\n", pStuInfo->nChannelID);

    }
    break;
default:
    printf("other event type[%d]\n", dwAlarmType);
    break;
}

if (dwBufSize > 0 && NULL != pBuffer)
{
    // in case of too many picturese being received at the same time, mark with i than
saving pictures via receiving time which may cause overlapping.
    static int i;
    char szPicturePath[256] = "";
    time_t stuTime;
    time(&stuTime);
    char szTmpTime[128] = "";
    strftime(szTmpTime, sizeof(szTmpTime) - 1, "%y%m%d_%H%M%S",
gmtime(&stuTime));
    _snprintf(szPicturePath, sizeof(szPicturePath)-1, "%d_%.jpg", ++i, szTmpTime);

    FILE* pFile = fopen(szPicturePath, "wb");
    if (NULL == pFile)
    {
        return 0;
    }
}
```

```
int nWrite = 0;
while(nWrite != dwBufSize)
{
    nWrite += fwrite(pBuffer + nWrite, 1, dwBufSize - nWrite, pFile);
}

fclose(pFile);
}

return 1;
}
```

## 3 Callback function definition

### 3.1 fDisConnect

Option	Instruction
Interface description	Disconnection callback function. Informing user via this callback when logged device gets disconnected.
Precondition	none
Function	<pre>typedef void(CALLBACK *fDisConnect)(     LLONG <a href="#">lLoginID</a>,     char *<a href="#">pchDVRIP</a>,     LONG <a href="#">nDVRPort</a>,     LDWORD <a href="#">dwUser</a> );</pre>
Parameter	<p><i>lLoginID</i> Logged device ID. Return value of login interface CLIENT_LoginEx2 .</p> <p><i>pchDVRIP</i> Device IP. Disconnected device IP which is the input IP of login interface.</p> <p><i>nDVRPort</i> Device port. Disconnected device port which is the input port of login interface.</p> <p><i>dwUser</i> User data</p>
Return value	None
Note	<p>Set this callback in interface CLIENT_Init.</p> <p>User can identify which device gets disconnected via parameters lLoginID, pchDVRIP and nDVRPort.</p>

## 3.2fHaveReConnect

Option	Instruction
Interface description	Successful reconnection callback function. When the disconnected device gets reconnected, call this interface to inform user.
Precondition	None
Function	<pre>typedef void (CALLBACK *fHaveReConnect)(     LLONG ILoginID,     char *pchDVRIP,     LONG nDVRPort,     LDWORD dwUser );</pre>
Parameter	<p><i>ILoginID</i> Logged device ID. Return value of login interface CLIENT_LoginEx2.</p> <p><i>pchDVRIP</i> Device IP. Disconnected device IP which is the input IP of login interface.</p> <p><i>nDVRPort</i> Device port. Disconnected device port which is the input port of login interface.</p> <p><i>dwUser</i> User data</p>
Return value	None
Note	Set this callback in interface CLIENT_SetAutoReconnect. User can identify which device gets reconnected via parameters ILoginID、pchDVRIP and nDVRPort.

### 3.3fRealDataCallBackEx

Option	Instruction										
Interface description	Real-time monitoring data callback function prototype extension										
Precondition	None										
Function	<pre>typedef void (CALLBACK*fRealDataCallBackEx)(     LLONG IRealHandle,     DWORD dwDataType,     BYTE *pBuffer,     DWORD dwBufSize,     LONG param,     LDWORD dwUser );</pre>										
Parameter	<p><i>IRealHandle</i></p> <p>Real-time monitoring handle.</p> <p>Return value of real-time monitoring interface like CLIENT_RealPlayExc et. <i>dwDataType</i>.</p> <p>Type of data being called back.</p> <p>The type is decided by parameter dwFlag set in CLIENT_SetRealDataCallBackEx.</p> <table> <thead> <tr> <th><b>dwDataType</b></th><th><b>description</b></th></tr> </thead> <tbody> <tr> <td><b>0</b></td><td>Original data which is the data stored by CLIENT_SaveRealData</td></tr> <tr> <td><b>1</b></td><td>Frame data</td></tr> <tr> <td><b>2</b></td><td>Yuv data</td></tr> <tr> <td><b>3</b></td><td>Pcm data</td></tr> </tbody> </table> <p><i>pBuffer</i></p> <p>Buffer for callback data.</p> <p>Data of different length will be called back according to different <i>dwDataType</i> value. The data are called back by frame for every</p>	<b>dwDataType</b>	<b>description</b>	<b>0</b>	Original data which is the data stored by CLIENT_SaveRealData	<b>1</b>	Frame data	<b>2</b>	Yuv data	<b>3</b>	Pcm data
<b>dwDataType</b>	<b>description</b>										
<b>0</b>	Original data which is the data stored by CLIENT_SaveRealData										
<b>1</b>	Frame data										
<b>2</b>	Yuv data										
<b>3</b>	Pcm data										



Option	Instruction
	<p><i>dwDataType</i> value but 0, each time one frame is called back.</p> <p><i>dwBufSize</i></p> <p>Callback data length.</p> <p>Different <i>dwDataType</i> has different length.(unit: byte).</p> <p><i>param</i></p> <p>callback data parameter struct.</p> <p>when <i>dwDataType</i> is 0 and 2, <i>param</i> is 0.</p> <p>When <i>dwDataType</i> is 1, <i>param</i> is a pointer of struct tagVideoFrameParam type. See struct tagVideoFrameParam for details.</p> <p>When <i>dwDataType</i> is 3, <i>param</i> is a pointer of struct tagCBPCMDDataParam type. See struct tagCBPCMDDataParam for details.</p> <p><i>dwUserData</i></p> <p>user data.</p>
Return value	None
Note	<p>Set the callback function in CLIENT_SetRealDataCallBackEx.</p> <p>User can identify which real-time monitoring the callback data belongs to via parameter IRealHandle.</p>

### 3.4fDownloadPosCallBack

Option	Instruction
Interface description	Playback pos callback function
Precondition	None
Function	<pre>typedef void (CALLBACK *fDownloadPosCallBack)(     LLONG IPlayHandle,     DWORD dwTotalSize,     DWORD dwDownloadSize,     LDWORD dwUser</pre>

Option	Instruction
	);
parameter	<p><i>IPlayHandle</i></p> <p>Playback handle</p> <p>Return value of playback interface like CLIENT_PlayBackByTimeEx etc.</p> <p><i>dwTotalSize</i></p> <p>Total size of the playback.(unit: KB)</p> <p><i>dwDownloadSize</i></p> <p>The size has been played. (unit: KB)</p> <p><i>dwDownloadSize</i> = -1 means the playback is finished.</p> <p><i>dwDownloadSize</i> = -2 means failed to write files.</p> <p><i>dwUser</i></p> <p>User data</p>
Return value	None
Note	<p>Set the callback function in CLIENT_PlayBackByTimeEx.</p> <p>User can identify which playback it is via parameter <i>IRealHandle</i>.</p>

### 3.5fDataCallback

Option	Instruction
Interface description	Playback data callback function
Precondition	None
Function	<pre>typedef int (CALLBACK *fDataCallBack)(     LLONG IRealHandle,     DWORD dwDataType,     BYTE *pBuffer,     DWORD dwBufSize,     LDWORD dwUser );</pre>
Parameter	<i>IRealHandle</i>

Option	Instruction
	<p>Playback handle</p> <p>Return value of playback interface like CLIENT_PlayBackByTimeEx etc.</p> <p><i>dwDataType</i></p> <p>Data type.</p> <p>The value remains 0, which means data of original type.</p> <p><i>pBuffer</i></p> <p>Data stored buffer.</p> <p>Used to store callback video data.</p> <p><i>dwBufSize</i></p> <p>Data stored buffer length.(unit: byte)</p> <p><i>dwUser</i></p> <p>User data</p>
Return value	<p>0: means callback failed, and next time the same data will be returned.</p> <p>1: means callback succeeded, and next time the follow-up data will be returned.</p>
Note	<p>Set the callback function in interfaces like CLIENT_PlayBackByTimeEx etc.</p> <p>If parameter hWnd of CLIENT_PlayBackByTimeEx is not NULL, no matter what value returns, the callback is being considered successful and next callback will return follow-up data.</p> <p>User can identify which playback it is via parameter <i>IRealHandle</i>.</p>

### 3.6fTimeDownloadPosCallBack

Option	Instruction
Interface description	Download by time callback function
Precondition	None
Function	<pre>typedef void (CALLBACK *fTimeDownloadPosCallBack) (     LLONG IPlayHandle,</pre>

Option	Instruction
	<p>DWORD dwTotalSize,          DWORD dwDownloadSize,          int index,          NET_RECORDFILE_INFO recordfileinfo,          LDWORD dwUser          );</p>
Parameter	<p><i>IPlayHandle</i>          Download handle          Return value of interfaces like CLIENT_DownloadByTimeEx etc.</p> <p><i>dwTotalSize</i>          Total size to be downloaded.(unit: KB)</p> <p><i>dwDownloadSize</i>          The size has been downloaded,(unit: KB)</p> <p><i>index</i>          Video files NO. during current download.</p> <p><i>recordfileinfo</i>          Current downloaded files info.          See struct NET_RECORDFILE_INFO for details.</p> <p><i>dwUser</i>          User data</p>
Return value	None
Note	<p>Set the callback function in interfaces like CLIENT_DownloadByTimeEx etc.</p> <p>User can identify which download it is via parameter <i>IRealHandle</i>.</p>

### 3.7fMessCallBack

Option	Instruction
Interface description	Alarm report callback function prototype
precondition	None

Option	Instruction
function	<pre>typedef BOOL (CALLBACK *fMessCallBack)(     LONG ICommand,     LLONG ILoginID,     char *pBuf,     DWORD dwBufLen,     char *pchDVRIP,     LONG nDVRPort,     LDWORD dwUser );</pre>
parameter	<p><i>ICommand</i> Alarm event type being called back. Use matched with <i>pBuf</i>, different <i>ICommand</i> has different <i>pBuf</i>. Please see “note” below for details.</p> <p><i>ILoginID</i> Logged device ID. Return value of login interface CLIENT_LoginEx2.</p> <p><i>pBuf</i> Alarm data received buffer. <i>pBuf</i> points to different data type according to different listen interface and different <i>ICommand</i> . Please see “note” for details。</p> <p><i>dwBufLen</i> The length of pBuf (unit:BYTE)</p> <p><i>pchDVRIP</i> Device IP which reports alarm</p> <p><i>nDVRPort</i> Device port which reports alarm</p> <p><i>dwUser</i> User data</p>

Option	Instruction
Return value	None
Note	<p>All the logined device use the same alarm report callback function .</p> <p>User judges which login report the alarm by parameter ILoginID.</p> <p>pBuf points to different data type according to different listen interface and different <i>ICommand</i>.</p> <p>As there are too many alarm events, here we don't introduce them all, user can search the following key section in header file Eng_dhnetSDK.h for details:</p> <pre>// Extensive alarm type,corresponding to CLIENT_StartListenEx #define DH_ALARM_ALARM_EX    0x2101 // External alarm</pre>

### 3.8fSearchDevicesCB

Option	Instruction
Interface description	Device search callback prototype
Precondition	None
Function	<pre>typedef void (CALLBACK *fSearchDevicesCB)(     DEVICE_NET_INFO_EX *pDevNetInfo,     void* pUserData );</pre>
Parameter	<p><i>pDevNetInfo</i> Device info struct. See struct DEVICE_NET_INFO_EX for details.</p> <p><i>pUserData</i> User data</p>
Return value	None
Note	<p>Device search callback function.</p> <p>It's not recommended to call SDK interfaces in this callback function.</p>

Option	Instruction
	Set the callback function in CLIENT_StartSearchDevices /CLIENT_SearchDevicesByIPs. When device is searched out, SDK will call this function.

### 3.9fAnalyzerDataCallBack

Option	Instruction
Interface description	Intelligent picture alarm callback function prototype
Precondition	None
Function	<pre>typedef int (CALLBACK *fAnalyzerDataCallBack)(     LLONG IAnalyzerHandle,     DWORD dwAlarmType,     void* pAlarmInfo,     BYTE *pBuffer,     DWORD dwBufSize,     LDWORD dwUser,     int nSequence,     void *reserved );</pre>
Parameter	<p><i>IAnalyzerHandle</i> Intelligent picture alarm subscription handle. When multiple intelligent picture alarm subscriptions use the same callback function, we can judge which subscription is by this parameter.</p> <p><i>dwAlarmType</i> Intelligent picture alarm type, this parameter should be used matched with parameter pBuffer. As there are too many alarm events, here we don't introduce them all, user can search the following key section in header file Eng_dhnetSDK.h for details:</p>

Option	Instruction
	<p><i>pAlarmInfo</i></p> <p>Struct pointer, used match with <i>dwAlarmType</i>, different <i>dwAlarmType</i> match with different pointer.see “note” below for information.</p> <p><i>pBuffer</i></p> <p>Intelligent picture info buffer</p> <p><i>dwBufSize</i></p> <p>The size of intelligent picture info</p> <p><i>dwUser</i></p> <p>User data</p> <p><i>nSequence</i></p> <p>Whether the picture is repeated</p> <p>0:it’s the first time that the picture shows up, and the follow-up alarms may use the picture.</p> <p>1: this picture is the same as the one shown in tha last alarm, and the follow-up alarms may use the picture.</p> <p>2: this picture is the same as the one shown in tha last alarm,and it will never show up again.or it is the only time that the picture shows up.</p> <p>(as most of alarms has an unique picture, the value is 2)</p> <p><i>reserved</i></p> <p>The state of the current callback. This parameter is an int pointer.</p> <p>what <i>*(int *)reserved</i> is:</p> <p>0: current data is real-time data</p> <p>1: current data is off-line data</p> <p>2: off-line transfer is finished</p> <p>(as most of the intelligent picture alarm data is real-time data ,the value of <i>*(int *)reserved</i> is 0)</p>
Return value	The return value has no meaning,0 is OK.
Note	<p>Intellengent picture alarm callback function.</p> <p>It’s not recommended to call SDK interfaces in this callback function.</p>



Option	Instruction
	<p>Set this callback function in CLIENT_RealLoadPictureEx / CLIENT_RealLoadPicture. When there are intelligent picture event reported by device-end, SDK will call this function.</p> <p>Due to SDK receiving buffer is 2M by default, when callback picture info exceed 2M, user need to call CLIENT_SetNetworkParam interface to set receiving buffer again, otherwise SDK will abandon data pack over 2M.</p> <p>Different dwAlarmType matched with different pointer.</p> <p>As there are too many alarm events, here we don't introduce them all, user can search the following key section in header file Eng_dhnetSDK.h for details:</p> <pre>#define EVENT_IVS_ALL    0x00000001    // subscription all event</pre>

### 3.10 fSnapRev

Option	Instruction
Interface description	Front-end video snapshot callback function prototype
Precondition	None
Function	<pre>typedef void (CALLBACK *fSnapRev)(     LLONG ILoginID,     BYTE *pBuf,     UINT RevLen,     UINT EncodeType,     DWORD CmdSerial,     LDWORD dwUser );</pre>
Parameter	<p><i>ILoginID</i>          Logged device ID.</p> <p>When multiple front-end video snapshots use the same callback function, we can judge which snapshot is by this parameter.</p> <p><i>pBuf</i></p>

Option	Instruction
	<p>Picture info buffer</p> <p>Used to store the picture info return by storage device.</p> <p><i>RevLen</i></p> <p>The size of picture info buffer</p> <p><i>EncodeType</i></p> <p>Encode type</p> <p>10:jpeg picture, 0:i frame of mpeg4</p> <p><i>CmdSerial</i></p> <p>The serial number.</p> <p>It is input by CLIENT_SnapPictureEx input parameter</p> <p><i>dwUser</i></p> <p>User data</p>
Return value	None
Note	<p>Snapshot callback function</p> <p>It's not recommended to call SDK interfaces in this callback function.</p> <p>Set this callback function in CLIENT_SetSnapRevCallBack. When there are snapshot picture data sent by device-end, SDK will call this function.</p> <p>Due to SDK receiving buffer is 2M by default, when callback picture info exceed 2M, user need to call CLIENT_SetNetworkParam interface to set receiving buffer again, otherwise SDK will abandon data pack over 2M.</p>

### 3.11 fRealPlayDisconnect

Option	Instruction
Interface description	Real-time monitoring disconnection callback function prototype
Precondition	None
Function	<pre>typedef void (CALLBACK *fRealPlayDisconnect)(     LLONG IOperateHandle,     EM_REALPLAY_DISCONNECT_EVENT_TYPE dwEventType,</pre>

Option	Instruction
	<pre>void* param, DWORD dwUser );</pre>
Parameter	<p><i>LOperateHandle</i> Real-time monitoring handle. When multiple real-time monitoring device use the same callback function, we can judge which operation is by this parameter.</p> <p><i>dwEventType</i> Event which causes disconnection. See enum EM_REALPLAY_DISCONNECT_EVENT_TYPE for reference.</p> <p><i>param</i> Reserved parameter, default value is NULL.</p> <p><i>dwUser</i> User data</p>
Return value	None
Note	<p>Real-time monitoring disconnection callback function</p> <p>It's not recommended to call SDK interfaces in this callback function.</p> <p>Set the callback in CLIENT_StartRealPlay when real-time monitoring is disconnected, SDK will call this function.</p>

## 3.12 pfAudioDataCallBack

Option	Instruction
Interface description	Audio data callback function prototype
Precondition	None
Function	<pre>typedef void (CALLBACK *pfAudioDataCallBack)(     LLONG ITalkHandle,     char *pDataBuf,     DWORD dwBufSize,</pre>

Option	Instruction
	<p>BYTE byAudioFlag,          LDWORD dwUser          );</p>
Parameter	<p><i>ITalkHandle</i>          Handle of audion intercom          Return value of audio intercom interfaces such as CLIENT_StartTalkEx etc.</p> <p><i>pDataBuf</i>          Audio data being called back          Where the data from is decided by parameter byAudioFlag</p> <p><i>dwBufSize</i>          The length of audio data being called back (unit:BYTE)</p> <p><i>byAudioFlag</i>          Flag indicates where the audio data from          0 indicates that the audio data received is PC-end data collected by local audio library.only CLIENT_RecordStartEx interface is called, can the data be called back.          1 indicates the data received is sent by device-end.</p> <p><i>dwUser</i>          User data</p>
Return value	None
Note	Set the callback in audio intercom interface such as CLIENT_StartTalkEx etc.

## Appendix 1 struct definition

### NET\_DEVICEINFO

Option	Instruction
Struct description	Device info
struct	<pre>typedef struct {     BYTE          sSerialNumber[DH_SERIALNO_LEN];     BYTE          byAlarmInPortNum;     BYTE          byAlarmOutPortNum;     BYTE          byDiskNum;     BYTE          byDVRType;     Union     {         BYTE      byChanNum;         BYTE      byLeftLogTimes;     }; }; } NET_DEVICEINFO, *LPNET_DEVICEINFO;</pre>
member	<pre>sSerialNumber     SN byAlarmInPortNum     DVR alarm input amount byAlarmOutPortNum     DVR alarm output amount byDiskNum     DVR HDD amount byDVRType     DVR type.Please refer to enum DHDEV_DEVICE_TYPE</pre>

Option	Instruction
	<p>byChanNum DVR channel amount</p> <p>byLeftLogTimes When login failed due to password error, notice user via this parameter, remaining login times 0 means this parameter is invalid</p>

## NET\_IN\_STARTLOGINEX

Option	Instruction
Struct description	Input parameter CLIENT_StartLoginEx
struct	<pre>typedef struct tagNET_IN_STARTLOGINEX {     DWORD          dwSize;     const char*     szIp;     DWORD          dwPort;     const char*     szName;     const char*     szPwd;     fHaveLogin      cbLogin;     LDWORD          dwUser; }NET_IN_STARTLOGINEX;</pre>
member	<p>dwSize The size of the struct.It should be set to sizeof(NET_IN_STARTLOGINEX) when used.</p> <p>szIp device ip</p> <p>dwPort login port</p> <p>szName username</p> <p>szPwd</p>

	<p>password</p> <p>cbLogin</p> <p>login result call, see callback function fHaveLogin for details.</p> <p>dwUser</p> <p>call user parameter</p>
--	-------------------------------------------------------------------------------------------------------------------------------------------------

## NET\_OUT\_STARTLOGINEX

Option	Instruction
Struct description	Out put parameter of CLIENT_StartLoginEx
Struct	<pre>typedef struct tagNET_OUT_STARTLOGINEX {     DWORD    dwSize; }NET_OUT_STARTLOGINEX;</pre>
Member	<p>dwSize</p> <p>The size of the struct.It should be set to sizeof(NET_OUT_STARTLOGINEX)when used.</p>

## NET\_PARAM

Option	Instruction
Struct description	Relevant parameters of login
Struct	<pre>typedef struct {     int        nWaittime;     int        nConnectTime;     int        nConnectTryNum;     int        nSubConnectSpaceTime;     int        nGetDevInfoTime;     int        nConnectBufSize;     int        nGetConnInfoTime;</pre>

	<pre> int        nSearchRecordTime;  int        nsubDisconnetTime;  BYTE       byNetType;  BYTE       byPlaybackBufSize;  BYTE       bDetectDisconnTime;  BYTE       bKeepLifeInterval;  int        nPicBufSize;  BYTE       bReserved[4];  } NET_PARAM; </pre>
Member	<pre> nWaittime     Waiting time(unit is ms), 0:default 5000ms.  nConnectTime     Connection timeout value(Unit is ms), 0:default 1500ms.  nConnectTryNum     Connection trial times, 0:default 1.  nSubConnectSpaceTime     Sub-connection waiting time(Unit is ms), 0:default 10ms.  nGetDevInfoTime     Access to device information timeout, 0:default 1000ms.  nConnectBufSize     Each connected to receive data buffer size(Bytes), 0:default 250*1024  nGetConnInfoTime     Access to sub-connect information timeout(Unit is ms), 0:default 1000ms.  nSearchRecordTime     Timeout value of search video (unit ms), default 3000ms  nsubDisconnetTime     dislink disconnect time,0:default 60000ms  byNetType     net type, 0-LAN, 1-WAN  byPlaybackBufSize </pre>



	<p>playback data from the receive buffer size(m),when value = 0,default 4M</p> <p>bDetectDisconnTime</p> <p>Pulse detect offline time(second) .When it is 0, the default setup is 60s, and the min time is 2s</p> <p>bKeepLifeInterval</p> <p>Pulse send out interval(second). When it is 0, the default setup is 10s, the min internal is 2s.</p> <p>nPicBufSize</p> <p>actual pictures of the receive buffer size(byte)when value = 0,default 2*1024*1024</p> <p>bReserved</p> <p>reserved</p>
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## NET\_DEVICEINFO\_Ext

Option	Instruction
Struct description	device info extension
struct	<pre>typedef struct {     BYTE    sSerialNumber[DH_SERIALNO_LEN];     int     nAlarmInPortNum;     int     nAlarmOutPortNum;     int     nDiskNum;     int     nDVRType;     int     nChanNum;     BYTE    byLimitLoginTime;     BYTE    byLeftLogTimes;     BYTE    bReserved[2];     int     nLockLeftTime;     char    Reserved[24]; }</pre>

	} NET_DEVICEINFO_Ex, *LPNET_DEVICEINFO_Ex;
Member	<p>sSerialNumber SN</p> <p>nAlarmInPortNum count of DVR alarm input</p> <p>nAlarmOutPortNum count of DVR alarm output</p> <p>nDiskNum number of DVR disk</p> <p>nDVRType DVR type, refer to enum DHDEV_DEVICE_TYPE</p> <p>nChanNum number of DVR channel</p> <p>byLimitLoginTime Online Timeout, Not Limited Access to 0, not 0 Minutes Limit Said</p> <p>byLeftLogTimes When login failed due to password error, notice user via this parameter, remaining login times is 0 means this parameter is invalid</p> <p>bReserved keep bytes, bytes aligned</p> <p>nLockLeftTime when log in failed, the left time for users to unlock (seconds), -1 indicate the device haven't set the parameter.</p> <p>Reserved reserved</p>

## tagVideoFrameParam

Option	Instruction
Struct description	Frame parameter struct of Callback video data frame

Option	Instruction
Struct	<pre>typedef struct _tagVideoFrameParam {     BYTE          encode;     BYTE          frametype;     BYTE          format;     BYTE          size;     DWORD         fourcc;     DWORD         reserved;     NET_TIME      struTime; } tagVideoFrameParam;</pre>
Member	<p>encode</p> <p>Encode type</p> <p>Different value corresponds to different encode type:</p> <p>MPEG4 encode: – 1</p> <p>Dahua H264 encode:– 2</p> <p>ADI H264 encode:– 3</p> <p>Standard H264 encode:- 4</p> <p>frametype</p> <p>frame type</p> <p>different value corresponds to different frame type:</p> <p>I frame - 0</p> <p>P frame - 1</p> <p>B frame - 2</p> <p>format</p> <p>video format</p> <p>different value corresponds to different different video format:</p> <p>PAL – 0</p> <p>NTSC - 1</p> <p>size</p>

Option	Instruction
	<p>resolution ratio</p> <p>different value corresponds to different different video resolution ratio:</p> <p>CIF – 0</p> <p>HD1 – 1</p> <p>2CIF – 2</p> <p>D1 – 3</p> <p>VGA – 4</p> <p>QCIF – 5</p> <p>QVGA – 6</p> <p>SVCD – 7</p> <p>QQVGA – 8</p> <p>SVGA – 9</p> <p>XVGA – 10</p> <p>WXGA – 11</p> <p>SXGA – 12</p> <p>WSXGA – 13</p> <p>UXGA – 14</p> <p>WUXGA – 15</p> <p>LFT – 16</p> <p>720 – 17</p> <p>1080 - 18</p> <p>fourcc</p> <p>If it is H264 encode type, the parameter value is always 0. Or is *( DWORD*)"DIVX" that is 0x58564944</p> <p>reserved</p> <p>reserved</p> <p>struTime</p> <p>Time information</p> <p>See struct NET_TIME for information</p>

## tagCBPCMDDataParam

Option	Instruction
Struct description	Frame parameter struct of audio data callback
Struct	<pre>typedef struct _tagCBPCMDDataParam {     BYTE        channels;     BYTE        samples;     BYTE        depth;     BYTE        param1;     DWORD       reserved; } tagCBPCMDDataParam;</pre>
Member	<p>channels</p> <p>Track amount</p> <p>samples</p> <p>sample rate</p> <p>different value corresponds to different different sample rate:</p> <p>0 – 8000</p> <p>1 – 11025</p> <p>2 - 16000</p> <p>3 - 22050</p> <p>4 - 32000</p> <p>5 - 44100</p> <p>6 - 48000</p> <p>depth</p> <p>Sampling depth. Value:8/16 show directly</p> <p>param1</p> <p>audio data type</p> <p>0 - indication no symbol</p> <p>1 - indication with symbol</p>

Option	Instruction
	reserved
	reserved

## NET\_TIME

Option	Instruction
Struct description	Time struct, exact to second
Struct	<pre>typedef struct {     DWORD    dwYear;     DWORD    dwMonth;     DWORD    dwDay;     DWORD    dwHour;     DWORD    dwMinute;     DWORD    dwSecond; } NET_TIME, *LPNET_TIME;</pre>
Member	<pre>dwYear     year dwMonth     month dwDay     day dwHour     hour dwMinute     minute dwSecond     second</pre>

## NET\_RECORDFILE\_INFO

Option	Instruction
Struct description	Record file info struct
Struct	<pre>typedef struct {     unsigned int    ch;     char            filename[124];     unsigned int    framenum;     unsigned int    size;     NET_TIME        starttime;     NET_TIME        endtime;     unsigned int    driveno;     unsigned int    startcluster;     BYTE            nRecordFileType;     BYTE            blmportantRecID;     BYTE            bHint;     BYTE            bRecType; } NET_RECORDFILE_INFO, *LPNET_RECORDFILE_INFO;</pre>
Member	<p>ch Channel number</p> <p>filename File name</p> <p>framenum the total number of file frames</p> <p>size File length</p> <p>starttime Start time</p> <p>endtime</p>

Option	Instruction
	<p>End time</p> <p>driveno</p> <p>HDD number</p> <p>(used to distinguish net record and local record.0~127: local record,64:HDD 1,128:net record)</p> <p>startcluster</p> <p>Initial cluster number</p> <p>nRecordFileType</p> <p>record file type</p> <p>0:general record;1:alarm record;</p> <p>2:motion detection;3:card number record ;</p> <p>4:image 5:intelligent record</p> <p>bImportantRecID</p> <p>flag of whether it's important record</p> <p>0:general record 1:Important record</p> <p>bHint</p> <p>Document Indexing</p> <p>(nRecordFileType==4&lt;picture&gt;时, bImportantRecID&lt;&lt;8 +bHint , make up picture locate index)</p> <p>bRecType</p> <p>stream type</p> <p>0-main stream record</p> <p>1-sub1 stream record</p> <p>2-sub2 stream record</p> <p>3-sub3 stream record</p>

## CFG\_PTZ\_PROTOCOL\_CAPS\_INFO

Option	Instruction
Struct	PTZ capability set information



Option	Instruction
description	
Struct	<pre> typedef struct tagCFG_PTZ_PROTOCOL_CAPS_INFO {     int    nStructSize;     BOOL   bPan;     BOOL   bTile;     BOOL   bZoom;     BOOL   bIris;     BOOL   bPreset;     BOOL   bRemovePreset;     BOOL   bTour;     BOOL   bRemoveTour;     BOOL   bPattern;     BOOL   bAutoPan;     BOOL   bAutoScan;     BOOL   bAux;     BOOL   bAlarm;     BOOL   bLight;     BOOL   bWiper;     BOOL   bFlip;     BOOL   bMenu;     BOOL   bMoveRelatively;     BOOL   bMoveAbsolutely;     BOOL   bReset;     BOOL   bGetStatus;     BOOL   bSupportLimit;     BOOL   bPtzDevice;     BOOL   bIsSupportViewRange;     WORD   wCamAddrMin; </pre>

Option	Instruction
	WORD wCamAddrMax; WORD wMonAddrMin; WORD wMonAddrMax; WORD wPresetMin; WORD wPresetMax; WORD wTourMin; WORD wTourMax; WORD wPatternMin; WORD wPatternMax; WORD wTileSpeedMin; WORD wTileSpeedMax; WORD wPanSpeedMin; WORD wPanSpeedMax; WORD wAutoScanMin; WORD wAutoScanMax; WORD wAuxMin; WORD wAuxMax; DWORD dwInterval; DWORD dwType; DWORD dwAlarmLen; DWORD dwNearLightNumber; DWORD dwFarLightNumber; DWORD dwSupportViewRangeType; DWORD dwSupportFocusMode; char szName[MAX_PROTOCOL_NAME_LEN]; char szAuxs[CFG_COMMON_STRING_32][CFG_COMMON_STRING_32]; CFG_PTZ_MOTION_RANGE stuPtzMotionRange; CFG_PTZ_LIGHTING_CONTROL stuPtzLightingControl;

Option	Instruction
	BOOL    bSupportPresetTimeSection; BOOL    bFocus; }CFG_PTZ_PROTOCOL_CAPS_INFO;
Member	nStructSize set to sizeof(CFG_PTZ_PROTOCOL_CAPS_INFO) when used bPan Whether to support PTZ horizontal swing bTile Whether to support PTZ vertical swinging bZoom Whether to support PTZ changed times blris Whether to support PTZ aperture adjustment bPreset Whether to support the preset point bRemovePreset Whether to support removal of preset point bTour Whether to support automatic cruise lines bRemoveTour Whether to support Clear cruise bPattern Whether to support the track line bAutoPan Whether to support automatic level swing bAutoScan Whether to support automatic scanning bAux Whether to support accessibility

Option	Instruction
	<p>bAlarm</p> <p>Support alarm function</p> <p>bLight</p> <p>Whether or not support the lighting, the contents see below "stuPtzLightingControl".</p> <p>bWiper</p> <p>Whether or not support the wipers</p> <p>bFlip</p> <p>Whether or not support Flip camera</p> <p>bMenu</p> <p>Whether or not support PTZ built-in menus</p> <p>bMoveRelatively</p> <p>Whether or not support the PTZ by a relative coordinate positioning</p> <p>bMoveAbsolutely</p> <p>Whether or not support PTZ in absolute coordinates</p> <p>bReset</p> <p>Whether or not support PTZ reset</p> <p>bGetStatus</p> <p>Whether or not support Get the state of motion and orientation coordinates of PTZ</p> <p>bSupportLimit</p> <p>Whether or not support the limit</p> <p>bPtzDevice</p> <p>Whether or not support PTZ equipment</p> <p>bIsSupportViewRange</p> <p>Whether or not support PTZ visible range</p> <p>wCamAddrMin</p> <p>The minimum channel address</p> <p>wCamAddrMax</p>

Option	Instruction
	<p>The maximum number of channel address</p> <p>wMonAddrMin</p> <p>Minimum monitoring addresses</p> <p>wMonAddrMax</p> <p>The maximum number of monitoring the address</p> <p>wPresetMin</p> <p>Minimum preset points</p> <p>wPresetMax</p> <p>The maximum preset points</p> <p>wTourMin</p> <p>The minimum value of automatic cruise lines</p> <p>wTourMax</p> <p>The maximum number of automatic cruise lines</p> <p>wPatternMin</p> <p>The minimum value of track circuit</p> <p>wPatternMax</p> <p>The maximum number of track circuit</p> <p>wTileSpeedMin</p> <p>The minimum value of vertical speed</p> <p>wTileSpeedMax</p> <p>The maximum vertical speed</p> <p>wPanSpeedMin</p> <p>The minimum value of horizontal velocity</p> <p>wPanSpeedMax</p> <p>The maximum horizontal velocity</p> <p>wAutoScanMin</p> <p>The minimum value of automatic scanning</p> <p>wAutoScanMax</p> <p>The maximum number of automatic scanning</p>

Option	Instruction
	<p>wAuxMin</p> <p>The minimum value of auxiliary functions</p> <p>wAuxMax</p> <p>The maximum number of auxiliary functions</p> <p>dwInterval</p> <p>Send the command time interval</p> <p>dwType</p> <p>The type of agreement, 0 - Local PTZ 1 - Remote PTZ</p> <p>dwAlarmLen</p> <p>The length of the alarm of the agreement</p> <p>dwNearLightNumber</p> <p>The number of near light group, 0-4, 0 means not supported</p> <p>dwFarLightNumber</p> <p>The number of beam group, 0-4, 0 means not supported</p> <p>dwSupportViewRangeType</p> <p>Visual field data acquisition mode supported by the mask, from low to high depending on the number , currently supported.</p> <p>The first 1: 1 expressed support the "ElectronicCompass" electronic compass mode.</p> <p>dwSupportFocusMode</p> <p>Supported Focus mode mask, from low to high depending on the number, see enum EM_SUPPORT_FOCUS_MODE</p> <p>szName</p> <p>The name of the protocol operations</p> <p>szAuxs</p> <p>PTZ auxiliary function names list</p> <p>stuPtzMotionRange</p> <p>PTZ rotation angle range, unit: degree,see struct CFG_PTZ_MOTION_RANGEfor details</p>

Option	Instruction
	<p>stuPtzLightingControl</p> <p>Lighting control content, see struct CFG_PTZ_LIGHTING_CONTROL for details</p> <p>bSupportPresetTimeSection</p> <p>Whether to support the function of the preset point time configuration</p> <p>bFocus</p> <p>Whether to support to Ptz focus</p>

## CFG\_PTZ\_MOTION\_RANGE

Option	Instruction
Struct description	PTZ rotation angle rang struct
Struct	<pre>typedef struct tagCFG_PTZ_MOTION_RANGE {     int    nHorizontalAngleMin;     int    nHorizontalAngleMax;     int    nVerticalAngleMin;     int    nVerticalAngleMax; }CFG_PTZ_MOTION_RANGE;</pre>
Member	<p>nHorizontalAngleMin</p> <p>Minimum level angle range, unit: degree</p> <p>nHorizontalAngleMax</p> <p>The maximum horizontal angle range, unit: degree</p> <p>nVerticalAngleMin</p> <p>Vertical angle range minimum, unit: degree</p> <p>nVerticalAngleMax</p> <p>Maximum vertical angle range, unit: degree</p>

## CFG\_PTZ\_LIGHTING\_CONTROL

Option	Instruction
Struct description	PTZ rotation angle range, unit: degree
Struct	<pre>typedef struct tagCFG_PTZ_LIGHTING_CONTROL {     char        szMode[CFG_COMMON_STRING_32];     DWORD       dwNearLightNumber;     DWORD       dwFarLightNumber; }CFG_PTZ_LIGHTING_CONTROL;</pre>
Member	<p>szMode</p> <p>Manual Lighting Control Mode</p> <p>"on-off": Direct Switch Mode,</p> <p>"adjustLight": Manually Adjust Brightness Mode</p> <p>dwNearLightNumber</p> <p>The number of near light group</p> <p>dwFarLightNumber</p> <p>The number of beam group</p>

## DHDEV\_TALKFORMAT\_LIST

Option	Instruction
Struct description	The audio talk type the device supported
Struct	<pre>typedef struct {     Int  nSupportNum;     DHDEV_TALKDECODE_INFO  type[64];     Char reserved[64]; } DHDEV_TALKFORMAT_LIST;</pre>
Member	<p>nSupportNum</p> <p>amount</p>



Option	Instruction
	type Encode type See struct DHDEV_TALKDECODE_INFO for details reserved reserved

## DHDEV\_TALKDECODE\_INFO

Option	Instruction
Struct description	Audio encode information
Struct	typedef struct { DH_TALK_CODING_TYPE encodeType; int nAudioBit; DWORD dwSampleRate; int nPacketPeriod; char reserved[60]; } DHDEV_TALKDECODE_INFO;
Member	encodeType Encode type See enum DH_TALK_CODING_TYPE for details nAudioBit Bit:8/16 dwSampleRate Sampling rate such as 8000 or 16000 nPacketPeriod Pack Period,Unit:ms reserved reserved

## DHDEV\_SYSTEM\_ATTR\_CFG

Option	Instruction
Struct description	System information
Struct	<pre> typedef struct {     DWORD          dwSize;      /* The following are read only for device */     DH_VERSION_INFO    stVersion;     DH_DSP_ENCODECAP    stDspEncodeCap;     BYTE              szDevSerialNo[DH_DEV_SERIALNO_LEN];     BYTE              byDevType;     BYTE              szDevType[DH_DEV_TYPE_LEN];     BYTE              byVideoCaptureNum;     BYTE              byAudioCaptureNum;     BYTE              byTalkInChanNum;     BYTE              byTalkOutChanNum;     BYTE              byDecodeChanNum;     BYTE              byAlarmInNum;     BYTE              byAlarmOutNum;     BYTE              byNetIOLNum;     BYTE              byUsbIOLNum;     BYTE              byIdeIOLNum;     BYTE              byComIOLNum;     BYTE              byLPTIOLNum;     BYTE              byVgaIOLNum;     BYTE              byIdeControlNum;     BYTE              byIdeControlType;     BYTE              byCapability;     BYTE              byMatrixOutNum;      /* The following are read-write part for device */ </pre>

Option	Instruction
	<p>             BYTE           byOverWrite;              BYTE           byRecordLen;              BYTE           byDSTEnable;              WORD           wDevNo;              BYTE           byVideoStandard;              BYTE           byDateFormat;              BYTE           byDateSprtr;              BYTE           byTimeFmt;              BYTE           byLanguage;              } DHDEV_SYSTEM_ATTR_CFG, *LPDHDEV_SYSTEM_ATTR_CFG;           </p>
Member	<p>             dwSize              size of struct, the parameter can be set to              sizeof(DHDEV_SYSTEM_ATTR_CFG) when used              /* The following are read only for device */              stVersion              device version information, see struct DH_VERSION_INFO for details              stDspEncodeCap              DSP capability description, see struct DH_DSP_ENCODECAP for details              szDevSerialNo              SN              ByDevType              device type. Please refer to enumeration DHDEV_DEVICE_TYPE              szDevType              Device detailed type. String format. It may be empty.              ByVideoCaptureNum              Video port amount              ByAudioCaptureNum              Audio port amount              byTalkInChanNum           </p>

Option	Instruction
	NSP
	ByTalkOutChanNum
	NSP
	byDecodeChanNum
	NSP
	ByAlarmInNum
	Alarm input port amount
	ByAlarmOutNum
	Alarm output amount port
	ByNetIOnum
	network port amount
	byUsbIOnum
	USB port amount
	ByIdeIOnum
	IDE amount
	ByComIOnum
	COM amount
	ByLPTIOnum
	LPT amount
	byVgaIOnum
	NSP
	byIdeControlNum
	NSP
	byIdeControlType
	NSP
	byCapability
	NSP, expansible description
	ByMatrixOutNum
	video matrix output amount

Option	Instruction
	<p>/* The following are read-write part for device */</p> <p>ByOverWrite</p> <p>Deal method when HDD is full, (1:overwrite; 0: stop)</p> <p>ByRecordLen</p> <p>Video Package length</p> <p>ByDSTEnable</p> <p>Enable DTS or not 1--enable. 0--disable</p> <p>WDevNo</p> <p>Device serial number. Remote control can use this SN to control.</p> <p>ByVideoStandard</p> <p>Video format</p> <p>ByDateFormat</p> <p>Date format</p> <p>ByDateSprtr</p> <p>Date separator(0: ".", 1: "-", 2: "/")</p> <p>ByTimeFmt</p> <p>Time separator (0-24H, 1-12H)</p> <p>byLanguage</p> <p>Please refer to DH_LANGUAGE_TYPE for enumeration value</p>

## NET\_SPEAK\_PARAM

Option	Instruction
Struct description	Audio talk parameter struct
Struct	<pre>typedef struct __NET_SPEAK_PARAM {     DWORD      dwSize;     int        nMode;     int        nSpeakerChannel;     BOOL       bEnableWait;</pre>

Option	Instruction
	} NET_SPEAK_PARAM;
Member	<p>dwSize struct size, set to sizeof(NET_SPEAK_PARAM) when used</p> <p>nMode mode type 0: talk back(default), 1: propaganda,from propaganda ro talk back,need afresh to configure</p> <p>nSpeakerChannel reproducer channel, valid when nMode ==1.</p> <p>bEnableWait Wait for device to responding or not when enable bidirectional talk. Default setup is no.TRUE:wait ;FALSE:no, the timeout time is set by CLIENT_SetNetworkParam, corresponds to nWaittime in struct NET_PARAM.</p>

## NET\_TALK\_TRANSFER\_PARAM

Option	Instruction
Struct description	Open the forwarding mode of intercom or not
Struct	<pre>typedef struct tagNET_TALK_TRANSFER_PARAM {     DWORD        dwSize;     BOOL         bTransfer; }NET_TALK_TRANSFER_PARAM;</pre>
Member	<p>dwSize sizeof sturct, set to sizeof(NET_TALK_TRANSFER_PARAM) when used.</p> <p>bTransfer Open the forwarding mode of intercom or not, TRUE: yes, FALSE: no</p>

## DEVICE\_NET\_INFO\_EX

Option	Instruction
Struct description	Callback info struct of to CLIENT_StartSearchDevices
Struct	<pre> typedef struct {     int            iIPVersion;     char           szIP[64];     int            nPort;     char           szSubmask[64];     char           szGateway[64];     char           szMac[DH_MACADDR_LEN];     char           szDeviceType[DH_DEV_TYPE_LEN];     BYTE           byManuFactory;     BYTE           byDefinition;     bool           bDhcpEn;     BYTE           byReserved1;     char           verifyData[88];     char           szSerialNo[DH_DEV_SERIALNO_LEN];     char           szDevSoftVersion[DH_MAX_URL_LEN];     char           szDetailType[DH_DEV_TYPE_LEN];     char           szVendor[DH_MAX_STRING_LEN];     char           szDevName[DH_MACHINE_NAME_NUM];     char           szUserName[DH_USER_NAME_LENGTH_EX];     char           szPassWord[DH_USER_NAME_LENGTH_EX];     unsigned short nHttpPort;     WORD           wVideoInputCh;     WORD           wRemoteVideoInputCh;     WORD           wVideoOutputCh;     WORD           wAlarmInputCh;     WORD           wAlarmOutputCh; </pre>

Option	Instruction
	char                    cReserved[244]; }DEVICE_NET_INFO_EX;
Member	iIPVersion ip version, 4 for IPV4, 6 for IPV6 szIP IP IPV4 like "192.168.0.1" IPV6 like "2008::1/64" nPort tcp port szSubmask Subnet mask(IPV6 do not have subnet mask) szGateway Gateway szMac MAC address szDeviceType Device type byManuFactory device manufactory, see class EM_IPC_TYPE byDefinition 1-Standard definition 2-High definition bDhcpEn Dhcp, true-open, false-close byReserved1 reserved, used for byte alignment verifyData verify data, obtained in CLIENT_StartSearchDevices callback function(the data can be used to verify when modifying IP) szSerialNo SN



Option	Instruction
	szDevSoftVersion soft version
	szDetailType device detail type
	szVendor OEM type
	SzDevName Device name
	szUserName user name for log in device(it need be filled when modify device ip)
	szPassWord pass word for log in device(it need be filled when modify device ip)
	nHttpPort HTTP server port
	wVideoInputCh count of video input channel
	wRemoteVideoInputCh count of remote video input
	wVideoOutputCh count of video output channel
	wAlarmInputCh count of alarm input
	wAlarmOutputCh count of alarm output
	cReserved reserved

## MANUAL\_SNAP\_PARAMETER

Option	Instruction
Struct description	Manual snap parameter
Struct	<pre>typedef struct _MANUAL_SNAP_PARAMETER{     int            nChannel;     BYTE           bySequence[64];     BYTE           byReserved[60]; }MANUAL_SNAP_PARAMETER;</pre>
Member	<p>nChannel snap channel,start with 0</p> <p>bySequence snap sequence string, the string can be returned when corresponding intelligent picture reported. If triggering multiple manual snap events at the same time, user can identify witch snap is by this parameter.</p> <p>byReserved reserved</p>

## OPR\_RIGHT\_EX

Option	Instruction
Struct description	Right information
Struct	<pre>typedef struct _OPR_RIGHT_EX {     DWORD          dwID;     char            name[DH_RIGHT_NAME_LENGTH];     char            memo[DH_MEMO_LENGTH]; } OPR_RIGHT_EX;</pre>

Option	Instruction
Member	<div>dwID</div> <div>right ID, each right has its own ID.</div> <div>name</div> <div>right name</div> <div>memo</div> <div>right note</div>

## OPR\_RIGHT\_NEW

Option	Instruction
Struct description	Right information
Struct	<pre>typedef struct _OPR_RIGHT_NEW {     DWORD      dwSize;     DWORD      dwID;     char        name[DH_RIGHT_NAME_LENGTH];     char        memo[DH_MEMO_LENGTH]; } OPR_RIGHT_NEW;</pre>
Member	<div>dwSize</div> <div>size of struct, set to sizeof(OPR_RIGHT_NEW) when used</div> <div>dwID</div> <div>right ID, each right has its own ID.</div> <div>name</div> <div>Right name</div> <div>memo</div> <div>note</div>

## NET\_DEV\_CHN\_COUNT\_INFO

Option	Instruction
--------	-------------

Option	Instruction
Struct description	Device channel amount information struct
Struct	<pre>typedef struct tagNET_DEV_CHN_COUNT_INFO {     DWORD          dwSize;     NET_CHN_COUNT_INFO  stuVideoIn;     NET_CHN_COUNT_INFO  stuVideoOut; } NET_DEV_CHN_COUNT_INFO;</pre>
Member	<p>dwSize size of struct, set to sizeof(NET_DEV_CHN_COUNT_INFO) when used</p> <p>stuVideoIn Video Input Channel, see struct NET_CHN_COUNT_INFO for details.</p> <p>stuVideoOut Video Output Channel, see struct NET_CHN_COUNT_INFO for details.</p>

## NET\_CHN\_COUNT\_INFO

Option	Instruction
Struct description	Channel number information
Struct	<pre>typedef struct tagNET_CHN_COUNT_INFO {     DWORD          dwSize;     int            nMaxTotal;     int            nCurTotal;     int            nMaxLocal;     int            nCurLocal;     int            nMaxRemote;     int            nCurRemote; } NET_CHN_COUNT_INFO;</pre>

Option	Instruction
Member	<p>dwSize size of struct, set to sizeof(NET_CHN_COUNT_INFO) when used</p> <p>nMaxTotal Equipment to the total number of channels (the sum of all valid channel number)</p> <p>nCurTotal the number of configured on channels</p> <p>nMaxLocal Maximum number of local channels, including motherboard and pluggable cartoon</p> <p>nCurLocal configured local channel number</p> <p>nMaxRemote Maximum number of remote channel</p> <p>nCurRemote Configured remote channel number</p>

## NET\_IN\_SNAP\_CFG\_CAPS

Option	Instruction
Struct description	Snap configure capacity input parameter
Struct	<pre>typedef struct tagNET_IN_SNAP_CFG_CAPS {     int          nChannelId;     BYTE         bReserved[1024]; }NET_IN_SNAP_CFG_CAPS;</pre>
Member	<p>nChannelId channel NO.</p> <p>bReserved</p>

Option	Instruction
	reserved

## NET\_OUT\_SNAP\_CFG\_CAPS

Option	Instruction
Struct description	Snap configure capacity output parameter
Struct	<pre>typedef struct tagNET_OUT_SNAP_CFG_CAPS {     int            nResolutionTypeNum;     DH_RESOLUTION_INFO  stuResolutionTypes[DH_MAX_CAPTURE_SIZE_NUM];      DWORD          dwFramesPerSecNum;     int            nFramesPerSecList[DH_MAX_FPS_NUM];     DWORD          dwQualityMun;     DWORD          nQualityList[DH_MAX_QUALITY_NUM];     DWORD          dwMode;     DWORD          dwFormat;     BYTE           bReserved[2048]; } NET_OUT_SNAP_CFG_CAPS;</pre>
Member	<p>nResolutionTypeNum NO. of supported video resolution infor, used corresponding with parameter stuResolutionTypes</p> <p>stuResolutionTypes resolution info struct, used corresponding with parameter nResolutionTypeNum</p> <p>dwFramesPerSecNum NO. of supported frame rate info, used corresponding with parameter nFramesPerSecLis</p>

Option	Instruction
	nFramesPerSecList frame rate list dwQualityMun supported video quality info, used corresponding with parameter nQualityList nQualityList video quality info list , used corresponding with parameter dwQualityMun dwMode mode presented in bit: bit 1: fixed time ,bit 2: manual dwFormat picture format presented in bit: bit 1: bmp, bit 2: jpg bReserved reserved

## DH\_RESOLUTION\_INFO

Option	Instruction
Struct description	Image resolution struct
Struct	typedef struct { unsigned short snWidth; unsigned short snHight; }DH_RESOLUTION_INFO;
Member	snWidth width snHight heigh

## DEOENC\_ VIDEOENC\_OPT

Option	Instruction
--------	-------------

Option	Instruction
Struct description	Video encode parameter
Struct	<pre> typedef struct tagCFG_VIDEOENC_OPT {     bool                abVideoEnable;     bool                abAudioEnable;     bool                abSnapEnable;     bool                abAudioAdd;     bool                abAudioFormat;     BOOL                bVideoEnable;     CFG_VIDEO_FORMAT    stuVideoFormat;     BOOL                bAudioEnable;     BOOL                bSnapEnable;     BOOL                bAudioAddEnable;     CFG_AUDIO_ENCODE_FORMAT    stuAudioFormat; } CFG_VIDEOENC_OPT; </pre>
Member	<p><b>abVideoEnable</b> used to indicate whether bVideoEnable is valid. in obtaining, it used to indentify whether supporting video enabled or not in setting, it used to indicate whether supporting video enabled or not</p> <p><b>abAudioEnable</b> used to indicate whether bAudioEnable is valid. in obtaining, it used to indentify whether supporting audio enabled or not in setting, it used to indicate whether supporting audio enabled or not</p> <p><b>abSnapEnable</b> used to indicate whether abSnapEnable is valid. in obtaining, it used to indentify whether supporting timer snapshot or not in setting, it used to indicate whether supporting timer snapshot or not</p> <p><b>abAudioAdd</b></p>



Option	Instruction
	<p>used to indicate whether bAudioAddEnable is valid.</p> <p>in obtaining, it used to identify whether supporting audio add enabled or not</p> <p>in setting, it used to indicate whether supporting audio add or not</p> <p>abAudioFormat</p> <p>used to indicate whether stuAudioFormat valid.</p> <p>in obtaining, it used to identify whether supporting audio format or not</p> <p>in setting, it used to indicate whether supporting audio format or not</p> <p>sbVideoEnable</p> <p>Video enable, used matched with abVideoEnable</p> <p>stuVideoFormat</p> <p>Video format, see struct CFG_VIDEO_FORMAT for details</p> <p>bAudioEnable</p> <p>Audio enable, used matched with abAudioEnable</p> <p>bSnapEnable</p> <p>Schedule snapshot enable, used matched with abSnapEnable</p> <p>bAudioAddEnable</p> <p>Audio add enable, used matched with abAudioAdd</p> <p>stuAudioFormat</p> <p>Audio format, used matched with abAudioFormat, see struct CFG_AUDIO_ENCODE_FORMAT for details.</p>

## CFG\_VIDEO\_FORMAT

Option	Instruction
Struct description	Video format struct
Struct	<pre>typedef struct tagCFG_VIDEO_FORMAT {     bool                abCompression;     bool                abWidth;</pre>

Option	Instruction
	<pre> bool                abHeight;  bool                abBitRateControl;  bool                abBitRate;  bool                abFrameRate;  bool                abIFrameInterval;  bool                abImageQuality;  bool                abFrameType;  bool                abProfile;  CFG_VIDEO_COMPRESSION emCompression;  int                 nWidth;  int                 nHeight;  CFG_BITRATE_CONTROL  emBitRateControl;  int                 nBitRate;  float               nFrameRate;  int                 nIFrameInterval;  CFG_IMAGE_QUALITY    emImageQuality;  int                 nFrameType;  CFG_H264_PROFILE_RANK emProfile;  } CFG_VIDEO_FORMAT; </pre>
Member	<pre> abCompression      TRUE: emCompression is valid; FALSE: emCompression is invalid      The parameter is read only, and subject to obtaining. Not recommended to     be modified.  abWidth      TRUE: nWidth is valid ; FALSE: nWidth is invalid      The parameter is read only, and subject to obtaining. Not recommended to     be modified.  abHeight      TRUE: nHeight is valid; FALSE: nHeight is invalid </pre>

Option	Instruction
	<p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abBitRateControl</p> <p>TRUE: emBitRateControl is valid; FALSE: emBitRateControl is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abBitRate</p> <p>TRUE: nBitRate is valid; FALSE: nBitRate is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abFrameRate</p> <p>TRUE: nFrameRate is valid; FALSE: nFrameRate is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abFrameInterval</p> <p>TRUE: nFrameInterval is valid; FALSE: nFrameInterval is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abImageQuality</p> <p>TRUE: emImageQuality is valid; FALSE: emImageQuality is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abFrameType</p> <p>TRUE: nFrameType is valid; FALSE: nFrameType is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abProfile</p> <p>TRUE: emProfile is valid; FALSE: emProfile is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to</p>

Option	Instruction
	<p>be modified.</p> <p>emCompression</p> <p>Video compression mode</p> <p>If this parameter is valid is decided by abCompression, see enum CFG_VIDEO_COMPRESSION for details.</p> <p>nWidth</p> <p>video width</p> <p>Whether this parameter valid or not is decided by abCompression</p> <p>nHeight</p> <p>video height</p> <p>Whether this parameter valid or not is decided by abHeight</p> <p>emBitRateControl</p> <p>stream control mode</p> <p>Whether this parameter valid or not is decided by abBitRateControl, see enum CFG_BITRATE_CONTROL for details.</p> <p>nBitRate</p> <p>video stream (kbps)</p> <p>Whether this parameter valid or not is decided by abBitRate</p> <p>nFrameRate</p> <p>video frame rate</p> <p>Whether this parameter valid or not is decided by abFrameRate</p> <p>nFrameInterval</p> <p>I frame interval(1-100). For example, 50 means there is I frame in each 49 B frame or P frame.</p> <p>Whether this parameter valid or not is decided by abIframeInterval</p> <p>emImageQuality</p> <p>Video quality</p> <p>Whether this parameter valid or not is decided by abImageQuality, see enum CFG_IMAGE_QUALITY for details.</p>

Option	Instruction
	<p>nFrameType</p> <p>Sniffer mode,0-DHAV,1-"PS"</p> <p>Whether this parameter valid or not is decided by abFrameType</p> <p>emProfile</p> <p>H.264 Encode level</p> <p>Whether this parameter valid or not is decided by abProfile, see enum CFG_H264_PROFILE_RANK for details.</p>

## CFG\_AUDIO\_ENCODE\_FORMAT

Option	Instruction
Struct description	Audion format
Struct	<pre>typedef struct tagCFG_AUDIO_FORMAT {     bool                abCompression;     bool                abDepth;     bool                abFrequency;     bool                abMode;     bool                abFrameType;     bool                abPacketPeriod;     CFG_AUDIO_FORMAT    emCompression;     AV_int32            nDepth;     AV_int32            nFrequency;     AV_int32            nMode;     AV_int32            nFrameType;     AV_int32            nPacketPeriod; } CFG_AUDIO_ENCODE_FORMAT;</pre>
Member	<p>abCompression</p> <p>TRUE: emCompression is valid;FALSE: emCompression is invalid</p>

Option	Instruction
	<p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abDepth</p> <p>TRUE: nDepth is valid;FALSE: nDepth is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abFrequency</p> <p>TRUE: nFrequency is valid;FALSE: nFrequency is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abMode</p> <p>TRUE: nMode is valid; FALSE: nMode is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abFrameType</p> <p>TRUE: nFrameType is valid;FALSE: nFrameType is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>abPacketPeriod</p> <p>TRUE: nPacketPeriod is valid;FALSE: nPacketPeriod is invalid</p> <p>The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p>emCompression</p> <p>Audio compression mode</p> <p>Whether this parameter valid or not is decided by abCompression, see enum CFG_AUDIO_FORMAT for details.</p> <p>nDepth</p> <p>Audio sampling depth</p> <p>Whether this parameter valid or not is decided by abDepth.</p>

Option	Instruction
	<p>nFrequency</p> <p>Audio sampling frequency</p> <p>Whether this parameter valid or not is decided by abFrequency</p> <p>nMode</p> <p>Audio encode mode</p> <p>Whether this parameter valid or not is decided by abMode</p> <p>nFrameType</p> <p>audio pack mode, 0-DHAV, 1-PS Audio pack mode</p> <p>Whether this parameter valid or not is decided by abFrameType</p> <p>nPacketPeriod</p> <p>Audio pack period(unit: ms)</p> <p>Whether this parameter valid or not is decided by abPacketPeriod</p>

## CFG\_VIDEO\_COVER

Option	Instruction
Struct description	Multiple-zone mask configuration
Struct	<pre>typedef struct tagCFG_VIDEO_COVER {     int          nTotalBlocks;     int          nCurBlocks;     CFG_COVER_INFO  stuCoverBlock[MAX_VIDEO_COVER_NUM]; } CFG_VIDEO_COVER;</pre>
Member	<p>nTotalBlocks</p> <p>The supported privacy mask zone amount</p> <p>nCurBlocks</p> <p>The amount of zones already set.</p> <p>stuCoverBlock</p> <p>The mask zone, see struct CFG_COVER_INFO for details</p>

Option	Instruction

## CFG\_COVER\_INFO

Option	Instruction
Struct description	Privacy mask information
Struct	<pre>typedef struct tagCFG_COVER_INFO {     bool            abBlockType;     bool            abEncodeBlend;     bool            abPreviewBlend;     CFG_RECT        stuRect;     CFG_RGBA        stuColor;     int             nBlockType;     int             nEncodeBlend;     int             nPreviewBlend; } CFG_COVER_INFO;</pre>
Member	<p><b>abBlockType</b>  TRUE: nBlockType is valid; FALSE: nBlockType is invalid  The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p><b>abEncodeBlend</b>  TRUE: nEncodeBlend is valid; FALSE: nEncodeBlend is invalid  The parameter is read only, and subject to obtaining. Not recommended to be modified.</p> <p><b>abPreviewBlend</b>  TRUE: nPreviewBlend is valid; FALSE: nPreviewBlend is invalid  The parameter is read only, and subject to obtaining. Not recommended to</p>



Option	Instruction
	<p>be modified.</p> <p>stuRect</p> <p>The position (coordinates) of the mask zone, see struct CFG_RECT for details.</p> <p>stuColor</p> <p>The mask color, see struct CFG_RGBA for details.</p> <p>nBlockType</p> <p>The mask mode ;0-black block,1-Mosaic</p> <p>Whether this parameter valid or not is decided by abBlockType.</p> <p>nEncodeBlend</p> <p>Encode-level privacy mask;1-enable,0-disable</p> <p>Whether this parameter valid or not is decided by abEncodeBlend.</p> <p>nPreviewBlend</p> <p>Preview mask;1-enable,0-disable</p> <p>Whether this parameter valid or not is decided by abPreviewBlend.</p>

## CFG\_RECT

Option	Instruction
Struct description	Zone information
Struct	<pre>typedef struct tagCFG_RECT {     int                nLeft;     int                nTop;     int                nRight;     int                nBottom; } CFG_RECT;</pre>
Member	<p>nLeft</p> <p>left side of the zone</p>

Option	Instruction
	nTop top of the zone nRight right side of the zone nBottom bottom of the zone

## CFG\_RGBA

Option	Instruction
Struct description	RGBA information
Struct	<pre>typedef struct tagCFG_RGBA {     int            nRed;     int            nGreen;     int            nBlue;     int            nAlpha; } CFG_RGBA;</pre>
member	nRed red nGreen green nBlue blue nAlpha pellucidity

## CFG\_ENCODE\_INFO

Option	Instruction
--------	-------------

Option	Instruction
Struct description	Video channel property information
Struct	<pre> typedef struct tagCFG_ENCODE_INFO {     int                nChannelID;     char                szChnName[MAX_CHANNELNAME_LEN];     CFG_VIDEOENC_OPT    stuMainStream[MAX_VIDEOSTREAM_NUM];     CFG_VIDEOENC_OPT    stuExtraStream[MAX_VIDEOSTREAM_NUM];     CFG_VIDEOENC_OPT    stuSnapFormat[MAX_VIDEOSTREAM_NUM];     DWORD               dwCoverAbilityMask;     DWORD               dwCoverEnableMask;     CFG_VIDEO_COVER      stuVideoCover;     CFG_OSD_INFO         stuChnTitle;     CFG_OSD_INFO         stuTimeTitle;     CFG_COLOR_INFO       stuVideoColor;     CFG_AUDIO_FORMAT     emAudioFormat;     int                  nProtocolVer; } CFG_ENCODE_INFO; </pre>
Member	<p>nChannelID Channel number(Begins with 0) valid when obtaining, invalid when setting</p> <p>szChnName invalid parameter</p> <p>stuMainStream main stream property information stuMainStream[0]-General record property information stuMainStream[1]-Motion detect record property information stuMainStream[2]-alarm record property information see struct CFG_VIDEOENC_OPT for details</p>

Option	Instruction
	<p>stuExtraStream</p> <p>sub-stream property information</p> <p>stuExtraStream[0]-General record property information</p> <p>stuExtraStream[1]-Motion detect record property information</p> <p>stuExtraStream[2]-alarm record property information</p> <p>see struct CFG_VIDEOENC_OPT for details.</p> <p>stuSnapFormat</p> <p>snapshot property information</p> <p>stuSnapFormat[0]—0-General snapshot property information</p> <p>stuSnapFormat[1]—Motion detect snapshot property information</p> <p>stuSnapFormat[2]—alarm snapshot property information</p> <p>see struct CFG_VIDEOENC_OPT for details</p> <p>dwCoverAbilityMask</p> <p>invalid parameter</p> <p>dwCoverEnableMask</p> <p>invalid parameter</p> <p>stuVideoCover</p> <p>invalid parameter</p> <p>stuChnTitle</p> <p>invalid parameter</p> <p>stuTimeTitle</p> <p>invalid parameter</p> <p>stuVideoColor</p> <p>invalid parameter</p> <p>emAudioFormat</p> <p>invalid parameter</p> <p>nProtocolVer</p> <p>Protocol Version No., read only</p> <p>valid when obtaining, invalid when setting</p>

## SNAP\_PARAMS

Option	Instruction
Struct description	Snapshot parameter structure
Struct	<pre>typedef struct _snap_param {     unsigned int    Channel;     unsigned int    Quality;     unsigned int    ImageSize;     unsigned int    mode;     unsigned int    InterSnap;     unsigned int    CmdSerial;     unsigned int    Reserved[4]; } SNAP_PARAMS, *LPSNAP_PARAMS;</pre>
Member	<p>Channel</p> <p>Snapshot channel</p> <p>Quality</p> <p>Image quality: evel 1 to level 6,the bigger the value is, the better quality the image is</p> <p>ImageSize</p> <p>Invalide parameter</p> <p>Mode</p> <p>Snapshot mode</p> <p>-1:stop snapshotting</p> <p>0:request one frame</p> <p>1:send out requestion regularly</p> <p>2: Request consecutively</p> <p>InterSnap</p>

Option	Instruction
	<p>Snapshot interval(unit: s)</p> <p>If mode=1, it means send out request regularly. Only certain special device(such as mobile device) sets snapshot interval through this parameter.</p> <p>Other device is suggested to realize related function through setting parameter tuSnapFormat[nSnapMode].stuVideoFormat.nFrameRate in configure CFG_CMD_ENCODE</p> <p>CmdSerial</p> <p>Snapshot Request serial number, valid range 0~65535, over this range will be cut to unsigned short.</p> <p>Reserved</p> <p>reserved</p>

## DH\_VERSION\_INFO

Option	Instruction
Struct description	Device software version information. The higher 16-bit means the main version number and low 16-bit means second version number.
Struct	<pre>typedef struct {     DWORD          dwSoftwareVersion;     DWORD          dwSoftwareBuildDate;     DWORD          dwDspSoftwareVersion;     DWORD          dwDspSoftwareBuildDate;     DWORD          dwPanelVersion;     DWORD          dwPanelSoftwareBuildDate;     DWORD          dwHardwareVersion;     DWORD          dwHardwareDate;     DWORD          dwWebVersion;     DWORD          dwWebBuildDate;</pre>

Option	Instruction
	} DH_VERSION_INFO, *LPDH_VERSION_INFO;
Member	dwSoftwareVersion software version dwSoftwareBuildDate software version build date dwDspSoftwareVersion dsp software version dwDspSoftwareBuildDate dsp software version build date dwPanelVersion not used dwPanelSoftwareBuildDate not used dwHardwareVersion hardware version dwHardwareDate not used dwWebVersion web version dwWebBuildDate web version build date

## DH\_DSP\_ENCODECAP

Option	Instruction
Struct description	DSP capacity description
Struct	typedef struct { DWORD dwVideoStandardMask;

Option	Instruction																												
	<p>DWORD dwImageSizeMask;</p> <p>DWORD dwEncodeModeMask;</p> <p>DWORD dwStreamCap;</p> <p>DWORD dwImageSizeMask_Assi[8];</p> <p>DWORD dwMaxEncodePower;</p> <p>WORD wMaxSupportChannel;</p> <p>WORD wChannelMaxSetSync;</p> <p>} DH_DSP_ENCODECAP, *LPDH_DSP_ENCODECAP;</p>																												
Member	<p>dwVideoStandardMask</p> <p>video format mask. Bit stands for the video format device supported.</p> <p>dwImageSizeMask</p> <p>Resolution mask, Bit stands for the resolution setup devices supported, see below:</p> <table> <tr> <td>0</td><td>704*576(PAL) 704*480(NTSC)</td></tr> <tr> <td>1</td><td>352*576(PAL) 352*480(NTSC)</td></tr> <tr> <td>2</td><td>704*288(PAL) 704*240(NTSC)</td></tr> <tr> <td>3</td><td>352*288(PAL) 352*240(NTSC)</td></tr> <tr> <td>4</td><td>176*144(PAL) 176*120(NTSC)</td></tr> <tr> <td>5</td><td>640*480</td></tr> <tr> <td>6</td><td>320*240</td></tr> <tr> <td>7</td><td>480*480</td></tr> <tr> <td>8</td><td>160*128</td></tr> <tr> <td>9</td><td>800*592</td></tr> <tr> <td>10</td><td>1024*768</td></tr> <tr> <td>11</td><td>1280*800</td></tr> <tr> <td>12</td><td>1600*1024</td></tr> <tr> <td>13</td><td>1600*1200</td></tr> </table>	0	704*576(PAL) 704*480(NTSC)	1	352*576(PAL) 352*480(NTSC)	2	704*288(PAL) 704*240(NTSC)	3	352*288(PAL) 352*240(NTSC)	4	176*144(PAL) 176*120(NTSC)	5	640*480	6	320*240	7	480*480	8	160*128	9	800*592	10	1024*768	11	1280*800	12	1600*1024	13	1600*1200
0	704*576(PAL) 704*480(NTSC)																												
1	352*576(PAL) 352*480(NTSC)																												
2	704*288(PAL) 704*240(NTSC)																												
3	352*288(PAL) 352*240(NTSC)																												
4	176*144(PAL) 176*120(NTSC)																												
5	640*480																												
6	320*240																												
7	480*480																												
8	160*128																												
9	800*592																												
10	1024*768																												
11	1280*800																												
12	1600*1024																												
13	1600*1200																												



Option	Instruction
	<p>14 1920*1200</p> <p>15 240*192</p> <p>16 1280*720</p> <p>17 1920*1080</p> <p>18 1280*960</p> <p>19 1872*1408</p> <p>20 3744*1408</p> <p>21 2048*1536</p> <p>22 2432*2050</p> <p>23 1216*1024</p> <p>24 1408*1024</p> <p>25 3296*2472</p> <p>26 2560*1920(5M)</p> <p>27 960*576(PAL) 960*480(NTSC)</p> <p>28 960*720</p> <p>dwEncodeModeMask</p> <p>Encode mode mask bit. Bit stands for the encode mode devices supported.</p> <p>dwStreamCap</p> <p>The multiple-media function the devices supported.</p> <p>The first bit:main stream</p> <p>The second bit:extra stream 1</p> <p>The third bit:extra stream 2</p> <p>The fifth bit: snapshot in .jpg format</p> <p>dwImageSizeMask_Assi</p> <p>When the main stream is the corresponding resolution, the supported extra stream resolution mask.</p> <p>dwMaxEncodePower</p>

Option	Instruction
	<p>The highest encode capacity DSP supported</p> <p>wMaxSupportChannel</p> <p>The max video channel amount each DSP supported.</p> <p>wChannelMaxSetSync</p> <p>Max encode bit setup in each DSP channel are synchronized or not ;0:does not synchronized,1:synchronized</p>

## Appendix 2 Enum definition

### NET\_DEVICE\_TYPE

Option	Instruction
Enum description	Enum of device type,used to different device type.
Enum definition	<pre>typedef enum tagNET_DEVICE_TYPE {     NET_PRODUCT_NONE = 0,     NET_DVR_NONREALTIME_MACE, // Non real-time MACE     NET_DVR_NONREALTIME,      // Non real-time     NET_NVS_MPEG1,             // Network video server     NET_DVR_MPEG1_2,           // MPEG1 2-ch DVR     NET_DVR_MPEG1_8,           // MPEG1 8-ch DVR     NET_DVR_MPEG4_8,           // MPEG4 8-ch DVR     NET_DVR_MPEG4_16,          // MPEG4 16-ch DVR     NET_DVR_MPEG4_SX2,         // LB series DVR     NET_DVR_MEPG4_ST2,         // GB series DVR     NET_DVR_MEPG4_SH2,         // HB series DVR     NET_DVR_MPEG4_GBE,         // GBE series DVR     NET_DVR_MPEG4_NVSII,       // II network video server     NET_DVR_STD_NEW,           //New standard configuration protocol     NET_DVR_DDNS,              // DDNS server     NET_DVR_ATM,               // ATM series     NET_NB_SERIAL,             // 2nd non real-time NB series DVR     NET_LN_SERIAL,             // LN series     NET_BAV_SERIAL,            // BAV series     NET_SDIP_SERIAL,           // SDIP series     NET_IPC_SERIAL,            //IPCseries     NET_NVS_B,                 // NVS B series }</pre>

Option	Instruction
	NET_NVS_C, // NVS H series
	NET_NVS_S, // NVS S series
	NET_NVS_E, // NVS E series
	NET_DVR_NEW_PROTOCOL, //Search device type from QueryDevState. it is in string format.
	NET_NVD_SERIAL, // NVD
	NET_DVR_N5, // N5
	NET_DVR_MIX_DVR, // HDVR
	NET_SVR_SERIAL, // SVR series
	NET_SVR_BS, //SVR-BS
	NET_NVR_SERIAL, // NVR series
	NET_DVR_N51, // N51
	NET_ITSE_SERIAL, // ITSE Intelligent Analysis Box
	NET_ITC_SERIAL, //Intelligent traffic camera equipment
	NET_HWS_SERIAL, // radar speedometer HWS
	NET_PVR_SERIAL, // portable video record
	NET_IVS_SERIAL, // IVS(intelligent video server series)
	NET_IVS_B, // universal intelligent detect video server series
	NET_IVS_F, // face recognition server
	NET_IVS_V, //video quality diagnosis server
	NET_MATRIX_SERIAL, // matrix
	NET_DVR_N52, // N52
	NET_DVR_N56, // N56
	NET_ESS_SERIAL, // ESS
	NET_IVS_PC, // number statistic server
	NET_PC_NVR, // pc-nvr

Option	Instruction
	<pre> NET_DSCON,           // screen controller NET_EVS,             //network video storage server NET_EIVS,            //an embedded intelligent video analysis                       system NET_DVR_N6,          // DVR-N6 NET_UDS,             // K-Lite Codec Pack NET_AF6016,          // Bank alarm host NET_AS5008,          //Video network alarm host NET_AH2008,          //Network alarm host NET_A_SERIAL,        // Alarm host series NET_BSC_SERIAL,      //Access control series                       of products NET_NVS_SERIAL,      //NVS series product NET_VTO_SERIAL,      //VTO series product NET_VTNC_SERIAL,     // VTNC series product NET_TPC_SERIAL,      //TPC series product,                       it is the thermal device. }NET_DEVICE_TYPE ; </pre>

## EM\_LOGIN\_SPAC\_CAP\_TYPE

Option	Instruction
Enum description	Enum of login type,used to select different ways to login.
Enum definition	<pre> typedef enum tagEM_LOGIN_SPAC_CAP_TYPE {     EM_LOGIN_SPEC_CAP_TCP = 0,           // TCP login, default     EM_LOGIN_SPEC_CAP_ANY = 1,          // No criteria login     EM_LOGIN_SPEC_CAP_SERVER_CONN=2,    //auto sign up login     EM_LOGIN_SPEC_CAP_MULTICAST=3,      //multicast login, default </pre>

Option	Instruction
	<pre> EM_LOGIN_SPEC_CAP_UDP=4,           //UDP method login EM_LOGIN_SPEC_CAP_MAIN_CONN_ONLY=6, //only main connection                                    login EM_LOGIN_SPEC_CAP_SSL=7,           //SSL encryption login EM_LOGIN_SPEC_CAP_INTELLIGENT_BOX=9, //login IVS box remote                                    device EM_LOGIN_SPEC_CAP_NO_CONFIG=10,    //login device do                                    not config EM_LOGIN_SPEC_CAP_U_LOGIN=11,      //USB key device login EM_LOGIN_SPEC_CAP_LDAP =12,        //LDAP login EM_LOGIN_SPEC_CAP_AD=13,           //AD(ActiveDirectory)                                    login EM_LOGIN_SPEC_CAP_RADIUS=14,       //Radius login EM_LOGIN_SPEC_CAP_SOCKET_5=15,     //Socks5 login EM_LOGIN_SPEC_CAP_CLOUD =16,       //cloud login EM_LOGIN_SPEC_CAP_AUTH_TWICE=17,   //dual authentication login EM_LOGIN_SPEC_CAP_TS=18,           //TS stream client login EM_LOGIN_SPEC_CAP_P2P=19,          //web private login EM_LOGIN_SPEC_CAP_MOBILE=20,       //mobile client login EM_LOGIN_SPEC_CAP_INVALID          // invalid login }EM_LOGIN_SPAC_CAP_TYPE;</pre>

## DH\_RealPlayType

Option	Instruction
Enum description	Preview type, correspond with CLIENT_RealPlayEx and CLIENT_StartRealPlay interfaces
Enum definition	<pre> typedef enum _RealPlayType {     DH_RType_Realplay = 0,    // Real-time preview</pre>

Option	Instruction
	<pre> DH_RType_Multiplay,           // Multiple-channel preview DH_RType_Realplay_0, //Real-time monitor-main stream. It is the same as                            DH_RType_Realplay DH_RType_Realplay_1, //1 Real-time monitor---extra stream 1 DH_RType_Realplay_2, //2 Real-time monitor-- extra stream 2 DH_RType_Realplay_3,    //3 Real-time monitor -- extra stream 3 DH_RType_Multiplay_1, //Multiple-channel                            preview-- 1-window DH_RType_Multiplay_4, //Multiple-channel preview--4-window DH_RType_Multiplay_8, //Multiple-channel preview--8-window DH_RType_Multiplay_9, //Multiple-channel preview--9-window DH_RType_Multiplay_16,    //Multiple-channel preview--16-window DH_RType_Multiplay_6, //Multiple-channel preview--6-window DH_RType_Multiplay_12,    //Multiple-channel preview--12-window DH_RType_Multiplay_25, //Multiple-channel preview--25-window DH_RType_Multiplay_36,    //Multiple-channel preview--36-window } DH_RealPlayType;</pre>

## EM\_QUERY\_RECORD\_TYPE

Option	Instruction
Enum description	Type of video search
Enum definition	<pre> typedef enum tagEmQueryRecordType {     EM_RECORD_TYPE_ALL=0,           //All the recorded video     EM_RECORD_TYPE_ALARM=1,        //The video of                                    external alarm     EM_RECORD_TYPE_MOTION_DETECT=2, //The video of dynamic                                    detection alarm     EM_RECORD_TYPE_ALARM_ALL=3,    //All the alarmed video }</pre>

Option	Instruction
	<pre> EM_RECORD_TYPE_CARD=4,           //query by the card number EM_RECORD_TYPE_CONDITION=5,       //query by condition EM_RECORD_TYPE_JOIN=6,           //combination query EM_RECORD_TYPE_CARD_PICTURE=8,    //query pictures by the card                                    number,used by HB-U or NVS EM_RECORD_TYPE_PICTURE=9,        //query pictures,used by HB-U                                    or NVS EM_RECORD_TYPE_FIELD=10,         //query by field EM_RECORD_TYPE_INTELLI_VIDEO =11, //Smart record search EM_RECORD_TYPE_TRANS_DATA=16,    //query the video of                                    serial data EM_RECORD_TYPE_IMPORTANT=17,     //query the important video EM_RECORD_TYPE_TALK_DATA =18,    //query the recording file EM_RECORD_TYPE_INVALID = 256,    // invalid query type }EM_QUERY_RECORD_TYPE; </pre>

## EM\_USEDEV\_MODE

Option	Instruction
Enum description	Audio talk way(not describe some extensive data type)
Enum definition	<pre> typedef enum __EM_USEDEV_MODE {     DH_TALK_CLIENT_MODE,           // Set client-end mode to begin audio                                    talk     DH_TALK_SERVER_MODE,          // Set server mode to begin audio                                    talk     DH_TALK_ENCODE_TYPE,         // Set encode format for                                    audio talk     DH_ALARM_LISTEN_MODE,        // Set alarm subscribe way     DH_CONFIG_AUTHORITY_MODE,    // Set user right to realize </pre>



Option	Instruction
	configuration management
DH_TALK_TALK_CHANNEL,	// Set talking channel(0~MaxChannel-1)
DH_RECORD_STREAM_TYPE,	// Set the stream type of the record for query(0-both main and extra stream,1-only main stream,2-only extra stream)
DH_TALK_SPEAK_PARAM,	// Set speaking parameter,corresponding to NET_SPEAK_PARAM
DH_RECORD_TYPE,	// Set by time video playback and download the video file TYPE (see.net RECORD TYPE)
DH_TALK_MODE3,	// Set voice intercom parameters of three generations of equipment and the corresponding structure NET TALK the EX
DH_PLAYBACK_REALTIME_MODE	, //Set real time playback function(0-off,1-on)
DH_TALK_TRANSFER_MODE,	//Judge the voice intercom if it was a forwarding mode, (corresponding to NET_TALK_TRANSFER_PARAM)
DH_TALK_VT_PARAM,	//Set VT talk parameter, corresponding to NET_VT_TALK_PARAM structure
DH_TARGET_DEV_ID,	//Set target device identifier for searching system capacity information, (not zero - locate device forwards the information)

Option	Instruction
	} EM_USEDEV_MODE;

## EM\_SUPPORT\_FOCUS\_MODE

Option	Instruction
Enum description	Enum of focus mode supported
Enum definition	<pre>typedef enum tagSUPPORT_FOCUS_MODE {     ENUM_SUPPORT_FOCUS_CAR= 1,    //See the car mode     ENUM_SUPPORT_FOCUS_PLATE= 2,  //See the license plate                                    number mode     ENUM_SUPPORT_FOCUS_PEOPLE= 3, //See people mode     ENUM_SUPPORT_FOCUS_FACE= 4,   //See the face mode }EM_SUPPORT_FOCUS_MODE;</pre>

## DH\_PTZ\_ControlType

Option	Instruction
Enum description	Enum of General PTZ control command
Enum definition	<pre>typedef enum _PTZ_ControlType {     DH_PTZ_UP_CONTROL = 0,          // Up     DH_PTZ_DOWN_CONTROL,           // Down     DH_PTZ_LEFT_CONTROL,           // Left     DH_PTZ_RIGHT_CONTROL,          // Right     DH_PTZ_ZOOM_ADD_CONTROL,       // +Zoom in     DH_PTZ_ZOOM_DEC_CONTROL,       // -Zoom out     DH_PTZ_FOCUS_ADD_CONTROL,      // +Zoom in     DH_PTZ_FOCUS_DEC_CONTROL,      // -Zoom out     DH_PTZ_APERTURE_ADD_CONTROL,   // + Aperture</pre>

Option	Instruction
	DH_PTZ_APERTURE_DEC_CONTROL,     // -Aperture DH_PTZ_POINT_MOVE_CONTROL,        // Go to preset DH_PTZ_POINT_SET_CONTROL,          // Set DH_PTZ_POINT_DEL_CONTROL,          // Delete DH_PTZ_POINT_LOOP_CONTROL,         // Tour DH_PTZ_LAMP_CONTROL                 // Light and wiper } DH_PTZ_ControlType;

## DH\_EXTPTZ\_ControlType

Option	Instruction
Enum description	PTZ control extensive command
Enum definition	typedef enum _EXTPTZ_ControlType { DH_EXTPTZ_LEFTTOP = 0x20,     // Upper left DH_EXTPTZ_RIGHTTOP,            // Upper right DH_EXTPTZ_LEFTDOWN,            // Down left DH_EXTPTZ_RIGHTDOWN,           // Down right DH_EXTPTZ_ADDTOLOOP,           // Add preset to tour Tour Preset value DH_EXTPTZ_DELFROMLOOP,         // Delete preset in tour Tour Preset value DH_EXTPTZ_CLOSELOOP,          // Delete tour    tour DH_EXTPTZ_STARTPANCUISE,       // Begin pan rotation DH_EXTPTZ_STOPPANCUISE,        // Stop pan rotation DH_EXTPTZ_SETLEFTBORDER,       // Set left limit DH_EXTPTZ_SETRIGHTBORDER,      // Set right limit DH_EXTPTZ_STARTLINESCAN,       // Begin scanning DH_EXTPTZ_CLOSELINESCAN,       // Stop scanning DH_EXTPTZ_SETMODESTART,        // Start mode    Mode line

Option	Instruction
	DH_EXTPTZ_SETMODESTOP, // Stop mode Mode line DH_EXTPTZ_RUNMODE, // Enable mode Mode line DH_EXTPTZ_STOPMODE, // Disable mode Mode line DH_EXTPTZ_DELETEMODE, // Delete mode Mode line DH_EXTPTZ_REVERSECOMM, // Flip DH_EXTPTZ_FASTGOTO, // 3D position X address(8192) Y address(8192)zoom(4) DH_EXTPTZ_AUXIOPEN, // auxiliary open Auxiliary point DH_EXTPTZ_AUXICLOSE, // Auxiliary close Auxiliary point DH_EXTPTZ_OPENMENU = 0x36, // Open dome menu DH_EXTPTZ_CLOSEMENU, // Close menu DH_EXTPTZ_MENUOK, // Confirm menu DH_EXTPTZ_MENUCANCEL, // Cancel menu DH_EXTPTZ_MENUUP, // menu up DH_EXTPTZ_MENUDOWN, // menu down DH_EXTPTZ_MENULEFT, // menu left DH_EXTPTZ_MENURIGHT, // Menu right DH_EXTPTZ_ALARMHANDLE = 0x40, // Alarm activate PTZ parm1:Alarm input channel;parm2:Alarm activation type 1-preset 2-scan 3-tour;parm 3:activation value,such as preset value. DH_EXTPTZ_MATRIXSWITCH = 0x41, // Matrix switch parm1:monitor number(video output number);parm2:video input number;parm3:matrix number DH_EXTPTZ_LIGHTCONTROL, // Light controller DH_EXTPTZ_EXACTGOTO, // 3D accurately positioning parm1:Pan degree(0~3600); parm2: tilt coordinates(0~900); parm3:zoom(1~128) DH_EXTPTZ_RESETZERO, // Reset 3D positioning as zero DH_EXTPTZ_MOVE_ABSOLUTELY, //Absolute motion control commands,param4 corresponding struct

Option	Instruction
	<p>PTZ_CONTROL_ABSOLUTELY</p> <p>DH_EXTPTZ_MOVE_CONTINUOUSLY, // Continuous motion control commands,param4 corresponding struct</p> <p>PTZ_CONTROL_CONTINUOUSLY</p> <p>DH_EXTPTZ_GOTOPRESET, // PTZ control command, at a certain speed to preset locus,param4 corresponding struct PTZ_CONTROL_GOTOPRESET</p> <p>DH_EXTPTZ_SET_VIEW_RANGE = 0x49, //Set to horizon(param4 corresponding struct PTZ_VIEW_RANGE_INFO)</p> <p>DH_EXTPTZ_FOCUS_ABSOLUTELY = 0x4A, // Absolute focus (param4 corresponding struct PTZ_FOCUS_ABSOLUTELY)</p> <p>DH_EXTPTZ_HORSECTORSCAN = 0x4B, //Level fan sweep (param4 corresponding PTZ_CONTROL_SECTORSCAN, param1、 param2、 param3 is invalid)</p> <p>DH_EXTPTZ_VERSECTORSCAN = 0x4C, // Vertical sweep fan (param4correspondingPTZ_CONTROL_SECTORSCAN, param1、 param2、 param3 is invalid)</p> <p>DH_EXTPTZ_SET_ABS_ZOOMFOCUS = 0x4D, // Set absolute focus, focus on value, param1 for focal length, range: [0255], param2 as the focus, scope: [0255], param3, param4 is invalid</p> <p>DH_EXTPTZ_SET_FISHEYE_EPTZ = 0x4E, //Control fish eye PTZ,param4corresponding to structure PTZ_CONTROL_SET_FISHEYE_EPTZ</p> <p>DH_EXTPTZ_UP_TELE = 0x70, // Up + TELE param1=speed(1-8) , similarly hereinafter</p> <p>DH_EXTPTZ_DOWN_TELE, // Down + TELE</p> <p>DH_EXTPTZ_LEFT_TELE, // Left + TELE</p> <p>DH_EXTPTZ_RIGHT_TELE, // Right + TELE</p> <p>DH_EXTPTZ_LEFTUP_TELE, // Upper left + TELE</p>

Option	Instruction
	DH_EXTPTZ_LEFTDOWN_TELE,     // Down left + TELE DH_EXTPTZ_TIGHTUP_TELE,       // Upper right + TELE DH_EXTPTZ_RIGHTDOWN_TELE,     // Down right + TELE DH_EXTPTZ_UP_WIDE,             // Up + WIDE param1=speed (1-8), similarly hereinafter DH_EXTPTZ_DOWN_WIDE,           // Down + WIDE DH_EXTPTZ_LEFT_WIDE,           // Left + WIDE DH_EXTPTZ_RIGHT_WIDE,          // Right + WIDE DH_EXTPTZ_LEFTUP_WIDE,         // Upper left + WIDE DH_EXTPTZ_LEFTDOWN_WIDE,       // Down left+ WIDE DH_EXTPTZ_TIGHTUP_WIDE,        // Upper right + WIDE DH_EXTPTZ_RIGHTDOWN_WIDE,      // Down right + WIDE DH_EXTPTZ_TOTAL,                // max command value } DH_EXTPTZ_ControlType;

## DH\_TALK\_CODING\_TYPE

Option	Instruction
Enum description	Audio encode type
Enum definition	typedef enum __TALK_CODING_TYPE { DH_TALK_DEFAULT = 0,            // No-head PCM DH_TALK_PCM = 1,                // With head PCM DH_TALK_G711a,                 // G711a DH_TALK_AMR,                    // AMR DH_TALK_G711u,                 // G711u DH_TALK_G726,                  // G726 DH_TALK_G723_53,               // G723_53 DH_TALK_G723_63,               // G723_63 DH_TALK_AAC,                    // AAC

Option	Instruction
	<pre> DH_TALK_OGG,           // OGG DH_TALK_G729 = 10,     // G729 DH_TALK_MPEG2,         // MPEG2 DH_TALK_MPEG2_Layer2,  // MPEG2-Layer2 DH_TALK_G722_1,        // G.722.1 DH_TALK_ADPCM = 21,    // ADPCM DH_TALK_MP3   = 22,    // MP3 } DH_TALK_CODING_TYPE;</pre>

## CtrlType

Option	Instruction
Enum description	Audio encode type
Enum definition	<pre> typedef enum _CtrlType {     DH_CTRL_REBOOT = 0,           // Reboot device     DH_CTRL_SHUTDOWN,           // Shut down device     DH_CTRL_DISK,               // HDD management     DH_KEYBOARD_POWER = 3,      // Network keyboard     DH_KEYBOARD_ENTER,     DH_KEYBOARD_ESC,     DH_KEYBOARD_UP,     DH_KEYBOARD_DOWN,     DH_KEYBOARD_LEFT,     DH_KEYBOARD_RIGHT,     DH_KEYBOARD_BTN0,     DH_KEYBOARD_BTN1,     DH_KEYBOARD_BTN2,     DH_KEYBOARD_BTN3,     DH_KEYBOARD_BTN4,</pre>

Option	Instruction
	DH_KEYBOARD_BTN5, DH_KEYBOARD_BTN6, DH_KEYBOARD_BTN7, DH_KEYBOARD_BTN8, DH_KEYBOARD_BTN9, DH_KEYBOARD_BTN10, DH_KEYBOARD_BTN11, DH_KEYBOARD_BTN12, DH_KEYBOARD_BTN13, DH_KEYBOARD_BTN14, DH_KEYBOARD_BTN15, DH_KEYBOARD_BTN16, DH_KEYBOARD_SPLIT, DH_KEYBOARD_ONE, DH_KEYBOARD_NINE, DH_KEYBOARD_ADDR, DH_KEYBOARD_INFO, DH_KEYBOARD_REC, DH_KEYBOARD_FN1, DH_KEYBOARD_FN2, DH_KEYBOARD_PLAY, DH_KEYBOARD_STOP, DH_KEYBOARD_SLOW, DH_KEYBOARD_FAST, DH_KEYBOARD_PREW, DH_KEYBOARD_NEXT, DH_KEYBOARD_JMPDOWN, DH_KEYBOARD_JMPUP, DH_KEYBOARD_10PLUS,



Option	Instruction
	DH_KEYBOARD_SHIFT, DH_KEYBOARD_BACK, DH_KEYBOARD_LOGIN ,           // new network keyboard function DH_KEYBOARD_CHNNEL ,        // switch video channel DH_TRIGGER_ALARM_IN = 100, // Activate alarm input DH_TRIGGER_ALARM_OUT,        // Activate alarm output DH_CTRL_MATRIX,                // Matrix control DH_CTRL_SDCARD,                // SD card control(for IPC series).  Please refer to HDD control DH_BURNING_START,            // Burner control:begin burning DH_BURNING_STOP,            // Burner control:stop burning DH_BURNING_ADDPWD,          // Burner control:overlay password  (The string ended with '\0'. Max length is 8 bits. ) DH_BURNING_ADDHEAD,         // Burner control:overlay head title  (The string ended with '\0'. Max length is 1024 bytes. Use '\n' to Enter.) DH_BURNING_ADDSIGN,         // Burner control:overlay dot to the burned  information(No parameter) DH_BURNING_ADDCURSTOMINFO, // Burner control:self-defined overlay  (The string ended with '\0'. Max length is 1024 bytes. Use '\n' to Enter) DH_CTRL_RESTOREDEFAULT,     //restore device default setup DH_CTRL_CAPTURE_START,      //Activate device snapshot DH_CTRL_CLEARLOG,            // Clear log DH_TRIGGER_ALARM_WIRELESS = 200, // Activate wireless  alarm (IPC series) DH_MARK_IMPORTANT_RECORD,    //Mark important record DH_CTRL_DISK_SUBAREA,        // Network hard disk partition DH_BURNING_ATTACH,            // Annex burning

Option	Instruction
	DH_BURNING_PAUSE, // Burn Pause DH_BURNING_CONTINUE, // Burn Resume DH_BURNING_POSTPONE, // Burn Postponed DH_CTRL_OEMCTRL, // OEM control DH_BACKUP_START, // Start to device backup DH_BACKUP_STOP, // Stop to device backup DH_VEHICLE_WIFI_ADD, // Add WIFI configuration manually for car device DH_VEHICLE_WIFI_DEC, // Delete WIFI configuration manually for car device DH_BUZZER_START, // Start to buzzer control DH_BUZZER_STOP, // Stop to buzzer control DH_REJECT_USER, // Reject User DH_SHIELD_USER, // Shield User DH_RAINBRUSH, // Rain Brush DH_MANUAL_SNAP, // manual snap (struct MANUAL_SNAP_PARAMETER) DH_MANUAL_NTP_TIMEADJUST, // manual ntp time adjust DH_NAVIGATION_SMS, // navigation info and note DH_CTRL_ROUTE_CROSSING, // route info DH_BACKUP_FORMAT, // backup device format DH_DEVICE_LOCALPREVIEW_SLIPT, // local preview split(struct DEVICE_LOCALPREVIEW_SLIPT_PARAMETER) DH_CTRL_INIT_RAID, // RAID init DH_CTRL_RAID, // RAID control DH_CTRL_SAPREDISK, // sapredisk control DH_WIFI_CONNECT, // wifi connect (struct WIFI_CONNECT) DH_WIFI_DISCONNECT, // wifi disconnect

Option	Instruction
	<p>(struct WIFI_CONNECT)</p> <p>DH_CTRL_ARMED, // Arm/disarm operation</p> <p>DH_CTRL_IP_MODIFY, // IP modify</p> <p>(struct DHCTRL_IPMODIFY_PARAM)</p> <p>DH_CTRL_WIFI_BY_WPS, // wps connect wifi(struct</p> <p>DHCTRL_CONNECT_WIFI_BYWPS)</p> <p>DH_CTRL_FORMAT_PARTITION, // format partition</p> <p>(struct DH_FORMAT_PARTITION)</p> <p>DH_CTRL_EJECT_STORAGE, //eject storage device</p> <p>(struct DH_EJECT_STORAGE_DEVICE)</p> <p>DH_CTRL_LOAD_STORAGE, // load storage device</p> <p>(struct DH_LOAD_STORAGE_DEVICE)</p> <p>DH_CTRL_CLOSE_BURNER, // close burner</p> <p>(struct NET_CTRL_BURNERDOOR) need wait 6s</p> <p>DH_CTRL_EJECT_BURNER, // eject burner</p> <p>(struct NET_CTRL_BURNERDOOR) need wait 4s</p> <p>DH_CTRL_CLEAR_ALARM, // alarm elimination corresponding</p> <p>structure NET (CTRL CLEAR ALARM)</p> <p>DH_CTRL_MONITORWALL_TVINFO, // TV wall information display</p> <p>corresponding structure NET (CTRL</p> <p>MONITORWALL TVINFO)</p> <p>DH_CTRL_START_VIDEO_ANALYSE, //start Intelligent VIDEO analysis</p> <p>(corresponding structure</p> <p>NET CTRL START VIDEO ANALYSE)</p> <p>DH_CTRL_STOP_VIDEO_ANALYSE, // STOP intelligent VIDEO analysis</p> <p>corresponding structure NET (CTRL STOP</p> <p>VIDEO ANALYSE)</p> <p>DH_CTRL_UPGRADE_DEVICE, //Controlled start equipment</p> <p>upgrades, independently complete the upgrade process</p>

Option	Instruction
	<p>by the equipment do not need to upgrade file</p> <p>DH_CTRL_MULTIPLAYBACK_CHANNALS, //Multi-channel preview  playback channel switching corresponding structure NET (CTRL  MULTIPLAYBACK CHANNALS)</p> <p>DH_CTRL_SEQPOWER_OPEN, // Turn on the switch power supply timing  device output corresponding.  net (CTRL SEQPOWER PARAM)</p> <p>DH_CTRL_SEQPOWER_CLOSE, // Close the switch power supply timing  device output corresponding.  net (CTRL SEQPOWER PARAM)</p> <p>DH_CTRL_SEQPOWER_OPEN_ALL, // Power timing group open the  switch quantity output corresponding.  net (CTRL SEQPOWER PARAM)</p> <p>DH_CTRL_SEQPOWER_CLOSE_ALL, // Power sequence set close the  switch quantity output corresponding.net (CTRL  SEQPOWER PARAM)</p> <p>DH_CTRL_PROJECTOR_RISE, // PROJECTOR up  corresponding.net (CTRL PROJECTOR PARAM)</p> <p>DH_CTRL_PROJECTOR_FALL, // PROJECTOR drop  (corresponding to the.net CTRL PROJECTOR PARAM)</p> <p>DH_CTRL_PROJECTOR_STOP, // PROJECTOR stop  (corresponding to the.net CTRL PROJECTOR PARAM)</p> <p>DH_CTRL_INFRARED_KEY, // INFRARED buttons  (corresponding to the.net CTRL INFRARED KEY PARAM)</p> <p>DH_CTRL_START_PLAYAUDIO, // Device START playback of audio  file corresponding structure NET  (CTRL START PLAYAUDIO)</p> <p>DH_CTRL_STOP_PLAYAUDIO, // Equipment stop playback of audio  file</p>

Option	Instruction
	<p>DH_CTRL_START_ALARMBELL, // Corresponding structure NET open alarm (CTRL ALARMBELL)</p> <p>DH_CTRL_STOP_ALARMBELL, // Close the warning signal corresponding structure NET (CTRL ALARMBELL)</p> <p>DH_CTRL_ACCESS_OPEN, // OPEN ACCESS control - corresponding structure NET (CTRL ACCESS OPEN)</p> <p>DH_CTRL_SET_BYPASS, //Corresponding structure NET BYPASS function (CTRL SET BYPASS)</p> <p>DH_CTRL_RECORDSET_INSERT, // Add records to record set number (corresponding to the.net CTRL you INSERT PARAM)</p> <p>DH_CTRL_RECORDSET_UPDATE, // Update a record of the number (corresponding to the.net CTRL you PARAM)</p> <p>DH_CTRL_RECORDSET_REMOVE, // According to the record set number to delete a record (corresponding to the.net CTRL you PARAM)</p> <p>DH_CTRL_RECORDSET_CLEAR, // Remove all RECORDSET information corresponding.net (CTRL you PARAM)</p> <p>DH_CTRL_ACCESS_CLOSE, // Entrance guard control - CLOSE corresponding structure NET (CTRL ACCESS CLOSE)</p> <p>DH_CTRL_ALARM_SUBSYSTEM_ACTIVE_SET, // Alarm sub system activation setup(corresponding structure NET_CTRL_ALARM_SUBSYSTEM_SETACTIVE)</p> <p>DH_CTRL_FORBID_OPEN_STROBE, //Disable device open gateway(corresponding to structure NET_CTRL_FORBID_OPEN_STROBE)</p> <p>DH_CTRL_OPEN_STROBE, //Enable gateway (corresponding to structure NET_CTRL_OPEN_STROBE)</p> <p>DH_CTRL_TALKING_REFUSE, // Talk no response (corresponding to structure NET_CTRL_TALKING_REFUSE)</p>

Option	Instruction
	<p> DH_CTRL_ARMED_EX, //arm-disarm operation  (corresponding to structure CTRL_ARM_DISARM_PARAM_EX),  CTRL_ARM_DISARM_PARAM upgrade,recommended  DH_CTRL_NET_KEYBOARD = 400, // Net keyboard  control(corresponding to structure DHCTRL_NET_KEYBOARD)  DH_CTRL_AIRCONDITION_OPEN, //Open air conditioner  (corresponding to structure NET_CTRL_OPEN_AIRCONDITION)  DH_CTRL_AIRCONDITION_CLOSE, //Close air-conditioner  (corresponding to structure NET_CTRL_CLOSE_AIRCONDITION)  DH_CTRL_AIRCONDITION_SET_TEMPERATURE, // Set temperature  (corresponding to structure  NET_CTRL_SET_TEMPERATURE)  DH_CTRL_AIRCONDITION_ADJUST_TEMPERATURE, //Adjust  temperature(corresponding to structure  NET_CTRL_ADJUST_TEMPERATURE)  DH_CTRL_AIRCONDITION_SETMODE, //Set air work mode  (corresponding to structure NET_CTRL_ADJUST_TEMPERATURE)  DH_CTRL_AIRCONDITION_SETWINDMODE, // Set fan mode  (corresponding to structure  NET_CTRL_AIRCONDITION_SETMODE)  DH_CTRL_RESTOREDEFAULT_EX ,  // Recover device default and set new protocol(corresponding to  structure NET_CTRL_RESTORE_DEFAULT)  // Recover config and use this enumeration first, if port failed,and  CLIENT_GetLastError return NET_UNSUPPORTED,  try again DH_CTRL_RESTOREDEFAULT restore config.  DH_CTRL_NOTIFY_EVENT, // send event to device (corresponding  to structure NET_NOTIFY_EVENT_DATA)  DH_CTRL_SILENT_ALARM_SET, // mute alarm setup </p>

Option	Instruction
	<p>DH_CTRL_START_PLAYAUDIOEX, //device start sound report(corresponding to structure NET_CTRL_START_PLAYAUDIOEX)</p> <p>DH_CTRL_STOP_PLAYAUDIOEX, //device stop sound report</p> <p>DH_CTRL_CLOSE_STROBE, //close gateway (corresponding to structure NET_CTRL_CLOSE_STROBE)</p> <p>DH_CTRL_SET_ORDER_STATE, //set parking reservation status (corresponding to structure NET_CTRL_SET_ORDER_STATE)</p> <p>DH_CTRL_RECORDSET_INSERTEX, // add record,get record collection no.(corresponding to NET_CTRL_RECORDSET_INSERT_PARAM)</p> <p>DH_CTRL_RECORDSET_UPDATEEX, // update record set no's record(corresponding to NET_CTRL_RECORDSET_PARAM)</p> <p>DH_CTRL_CAPTURE_FINGER_PRINT, // fingerprint collection (corresponding to structure NET_CTRL_CAPTURE_FINGER_PRINT)</p> <p>DH_CTRL_ECK_LED_SET, // Parking lot entrance/exit controller LED setup(corresponding structure NET_CTRL_ECK_LED_SET_PARAM)</p> <p>DH_CTRL_ECK_IC_CARD_IMPORT, // Intelligent parking system in/out device IC card info import (corresponding structure NET_CTRL_ECK_IC_CARD_IMPORT_PARAM)</p> <p>DH_CTRL_ECK_SYNC_IC_CARD, // Intelligent parking system in/out device IC card info sync command, receive this command, device will delete original IC card info(corresponding structure NET_CTRL_ECK_SYNC_IC_CARD_PARAM)</p> <p>DH_CTRL_LOWRATEWPAN_REMOVE, // Delete specific wireless device(corresponding structure NET_CTRL_LOWRATEWPAN_REMOVE)</p> <p>DH_CTRL_LOWRATEWPAN_MODIFY, // Modify wireless device</p>

Option	Instruction
	<p>info(corresponding structure</p> <p>NET_CTRL_LOWRATEWPAN_MODIFY)</p> <p>DH_CTRL_ECK_SET_PARK_INFO, // Set up the vehicle spot</p> <p>information of the machine at the passageway of the</p> <p>intelligent parking system (corresponding to</p> <p>NET_CTRL_ECK_SET_PARK_INFO_PARAM)</p> <p>DH_CTRL_VTP_DISCONNECT, // hang up the video phone</p> <p>(corresponding to NET_CTRL_VTP_DISCONNECT)</p> <p>DH_CTRL_UPDATE_FILES, // the update of the remote</p> <p>multimedia files (corresponding to NET_CTRL_UPDATE_FILES)</p> <p>DH_CTRL_MATRIX_SAVE_SWITCH, // Save up the relationship</p> <p>between the hyponymy matrixes (corresponding to</p> <p>NET_CTRL_MATRIX_SAVE_SWITCH)</p> <p>DH_CTRL_MATRIX_RESTORE_SWITCH, // recover the relationship</p> <p>between the hyponymy matrixes (corresponding to</p> <p>NET_CTRL_MATRIX_RESTORE_SWITCH)</p> <p>DH_CTRL_VTP_DIVERTACK, // video talk phone divert ack</p> <p>(corresponding to NET_CTRL_VTP_DIVERTACK)</p> <p>DH_CTRL_RAINBRUSH_MOVEONCE, // Rain-brush brush one time,</p> <p>efficient when set as manual mode(corresponding to</p> <p>NET_CTRL_RAINBRUSH_MOVEONCE)</p> <p>DH_CTRL_RAINBRUSH_MOVECONTINUOUSLY, // Rain-brush brush</p> <p>cyclic, efficient when set as manal mode(corresponding to</p> <p>NET_CTRL_RAINBRUSH_MOVECONTINUOUSLY)</p> <p>DH_CTRL_RAINBRUSH_STOPMOVE, // Rain-brush stop, efficient when</p> <p>set as manal mode(corresponding to</p> <p>NET_CTRL_RAINBRUSH_STOPMOVE)</p> <p>DH_CTRL_ALARM_ACK, // affirm the alarm event</p> <p>(corresponding to NET_CTRL_ALARM_ACK)</p>



Option	Instruction
	<pre> // DH_CTRL_ALARM_ACK DO NOT call this method in callback interface  DH_CTRL_RECORDSET_IMPORT,    // Batch import record set info     (Corresponding to NET_CTRL_RECORDSET_PARAM) DH_CTRL_ACCESS_USE_DOOR,    // Disable and enable door     (Corresponding to structure NET_CTRL_ACCESS_USE_DOOR) DH_CTRL_ACCESS_SHUT_LOCK,    // The latch and the cancellation of     the lock, can not pass through the door(Corresponding to     structure NET_CTRL_ACCESS_SHUT_LOCK) DH_CTRL_OPEN_DOOR_CONTINUE,  //Continuous unlocking     instruction (Corresponding to structure     NET_CTRL_OPEN_DOOR_CONTINUE) //The following commands are only for CLIENT_ControlDeviceEx  DH_CTRL_THERMO_GRAPHY_ENSHUTTER = 0x10000,     // Enable or disable thermal shutter,     //pInBuf= NET_IN_THERMO_EN_SHUTTER*,     //pOutBuf= NET_OUT_THERMO_EN_SHUTTER * DH_CTRL_RADIOMETRY_SETOSDMARK,     // NET_IN_RADIOMETRY_SETOSDMARK*,     //pOutBuf= NET_OUT_RADIOMETRY_SETOSDMARK * DH_CTRL_AUDIO_REC_START_NAME,     // Enable audio record and get audio name,     //pInBuf = NET_IN_AUDIO_REC_MNG_NAME *,     //pOutBuf = NET_OUT_AUDIO_REC_MNG_NAME * DH_CTRL_AUDIO_REC_STOP_NAME,     // Close audio file and return file name,     //pInBuf ,NET_IN_AUDIO_REC_MNG_NAME *,     //pOutBuf = NET_OUT_AUDIO_REC_MNG_NAME * </pre>

Option	Instruction
	<p>DH_CTRL_SNAP_MNG_SNAP_SHOT,</p> <p>//Manual snap, pInBuf = NET_IN_SNAP_MNG_SHOT * ,</p> <p>//pOutBuf = NET_OUT_SNAP_MNG_SHOT *</p> <p>DH_CTRL_LOG_STOP,</p> <p>// Forcedly sync buffer data to the database and close the database,</p> <p>//pInBuf = NET_IN_LOG_MNG_CTRL * ,</p> <p>//pOutBuf = NET_OUT_LOG_MNG_CTRL *</p> <p>DH_CTRL_LOG_RESUME,</p> <p>// Resume database, pInBuf = NET_IN_LOG_MNG_CTRL * ,</p> <p>// pOutBuf = NET_OUT_LOG_MNG_CTRL *</p> <p>DH_CTRL_POS_ADD,</p> <p>// Add a POS device, pInBuf = NET_IN_POS_ADD * ,</p> <p>//pOutBuf = NET_OUT_POS_ADD *</p> <p>DH_CTRL_POS_REMOVE,</p> <p>// Del a POS device, pInBuf = NET_IN_POS_REMOVE * ,</p> <p>//pOutBuf = NET_OUT_POS_REMOVE *</p> <p>DH_CTRL_POS_REMOVE_MULTI,</p> <p>// Del several POS device,</p> <p>//pInBuf = NET_IN_POS_REMOVE_MULTI * ,</p> <p>//pOutBuf = NET_OUT_POS_REMOVE_MULTI *</p> <p>DH_CTRL_POS_MODIFY,</p> <p>// Modify a POS device, pInBuf = NET_IN_POS_ADD * ,</p> <p>//pOutBuf = NET_OUT_POS_ADD *</p> <p>DH_CTRL_SET_SOUND_ALARM,</p> <p>//Trigger alarm with sound,</p> <p>//pInBuf = NET_IN_SOUND_ALARM * ,</p> <p>//pOutBuf = NET_OUT_SOUND_ALARM *</p> <p>DH_CTRL_AUDIO_MATRIX_SILENCE,</p> <p>//audiomatrix silence,</p>

Option	Instruction
	<pre> //pInBuf = NET_IN_AUDIO_MATRIX_SILENCE*, //pOutBuf = NET_OUT_AUDIO_MATRIX_SILENCE* DH_CTRL_MANUAL_UPLOAD_PICTURE, // manual upload picture, //pInBuf = NET_IN_MANUAL_UPLOAD_PICTURE *, //pOutBuf = NET_OUT_MANUAL_UPLOAD_PICTURE * DH_CTRL_REBOOT_NET_DECODING_DEV, // reboot network decoding device, //pInBuf = NET_IN_REBOOT_NET_DECODING_DEV *, //pOutBuf = NET_OUT_REBOOT_NET_DECODING_DEV * } CtrlType; </pre>

## CFG\_VIDEO\_COMPRESSION

Enum description	Instruction
Enum definition	Video compression format type
Enum description	<pre> typedef enum tagCFG_VIDEO_COMPRESSION {     VIDEO_FORMAT_MPEG4,          // MPEG4     VIDEO_FORMAT_MS_MPEG4,       // MS-MPEG4     VIDEO_FORMAT_MPEG2,          // MPEG2     VIDEO_FORMAT_MPEG1,          // MPEG1     VIDEO_FORMAT_H263,           // H.263     VIDEO_FORMAT_MJPEG,          // MJPG     VIDEO_FORMAT_FCC_MPEG4,      // FCC-MPEG4     VIDEO_FORMAT_H264,           // H.264     VIDEO_FORMAT_H265,           // H.265 } CFG_VIDEO_COMPRESSION; </pre>

## CFG\_BITRATE\_CONTROL

Option	Instruction
Enum description	Bit rate control
Enum definition	<pre>typedef enum tagCFG_BITRATE_CONTROL {     BITRATE_CBR,          //Fixed bit rate     BITRATE_VBR,          //variable bit rate } CFG_BITRATE_CONTROL;</pre>

## CFG\_IMAGE\_QUALITY

Option	Instruction
Enum description	Bit rate control
Enum definition	<pre>typedef enum tagCFG_IMAGE_QUALITY {     IMAGE_QUALITY_Q10 = 1,    // 10 percent image quality     IMAGE_QUALITY_Q30,        // 30 percent image quality     IMAGE_QUALITY_Q50,        // 50 percent image quality     IMAGE_QUALITY_Q60,        // 60 percent image quality     IMAGE_QUALITY_Q80,        // 80 percent image quality     IMAGE_QUALITY_Q100,       // 100 percent image quality } CFG_IMAGE_QUALITY;</pre>

## CFG\_H264\_PROFILE\_RANK

Option	Instruction
Enum description	H264 encode rank
Enum definition	<pre>typedef enum tagCFG_H264_PROFILE_RANK {     PROFILE_BASELINE=1,      //Provide I/P Frame,only support progressive                              scanning and CAVLC }</pre>

Option	Instruction
	<pre> PROFILE_MAIN,           //Provide I/P/B Frame,support progressive and                            interlaced,provide CAVLC and CABAC  PROFILE_EXTENDED,       //Provide I/P/B/SP/SI Frame, only support                            progressive scanning and CAVLC  PROFILE_HIGH,           //Based on FExt,Main_Profile, new add: 8x8                            intra prediction, custom                             //quant, lossless video coding,and many more yuv                            formats.  }CFG_H264_PROFILE_RANK;</pre>

## CFG\_AUDIO\_FORMAT

Option	Instruction
Enum description	Audio encode format
Enum definition	<pre> typedef enum tatCFG_AUDIO_FORAMT {     AUDIO_FORMAT_G711A,      // G711a     AUDIO_FORMAT_PCM,        // PCM     AUDIO_FORMAT_G711U,      // G711u     AUDIO_FORMAT_AMR,        // AMR     AUDIO_FORMAT_AAC,        // AAC } CFG_AUDIO_FORMAT;</pre>

## EM\_REALPLAY\_DISCONNECT\_EVENT\_TYPE

Option	Instruction
Enum description	Realplay disconnect event type
Enum definition	<pre> typedef enum_EM_REALPLAY_DISCONNECT_EVENT_TYPE {     DISCONNECT_EVENT_REAVE,    //resources is taken by</pre>

Option	Instruction
	<div>advanced user</div> <div>DISCONNECT_EVENT_NETFORBID, //forbidden</div> <div>DISCONNECT_EVENT_SUBCONNECT, //sublink disconnect</div> <div>}EM_REALPLAY_DISCONNECT_EVENT_TYPE;</div>

## Appendix 3 Interface definition

### CLIENT\_Init

Option	Instruction
Interface description	Interface for SDK initialization,call it when application programs need to be initialized.
Precondition	None
Function	<pre>BOOL CLIENT_Init(     fDisconnect cbDisconnect,     LDWORD dwUser );</pre>
Parameter	<p>cbDisconnect</p> <p><i>[in]</i> callback function used when device gets disconnected,when online device is offline,SDK will notify user by this callback, info includes login ID,device IP,login port etc,please refer to “3.1fDisconnect”for details.when function is set to 0,it means to prohibit the callback.</p> <p>dwUser</p> <p><i>[in]</i>user data,when callback function cbDisconnect is not 0,SDK will send data to user via this function for further operation.</p>
Return value	Succeeded return TRUE,otherwise return FALSE.
Example	<pre>// It's not recommended to call SDK interface in callback function, unless call <a href="#">CLIENT_GetLastError</a> to get error code in current process.  // callback function used when device gets disconnected.  //When online device gets offline,SDK will call this callback function set by CLINET_Init.  void CALLBACK DisconnectFunc(LONG ILoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser) {      printf("Call DisconnectFunc\n");</pre>

Option	Instruction
	<pre> printf("ILoginID[0x%x]", ILoginID); if (NULL != pchDVRIP) {     printf("pchDVRIP[%s]\n", pchDVRIP); }  printf("nDVRPort[%d]\n", nDVRPort); printf("dwUser[%p]\n", dwUser); printf("\n"); }  ***** Above are callback function definition, the underneath are interface using examples. *****  //SDK initialization g_bNetSDKInitFlag = CLIENT_Init(DisconnectFunc, 0); if (FALSE == g_bNetSDKInitFlag) {     printf("Initialize client SDK failed; \n");     return; } else {     printf("Initialize client SDK done; \n"); } </pre>
Note	<p>Before call other SDK interface, call this interface firstly.</p> <p>If call this interface repeatedly, the first time is valid.</p>

## CLIENT\_Cleanup

Option	Instruction
Interface description	Interface for SDK clearance.



Option	Instruction
Precondition	Have called CLIENT_Init interface.
Function	void CLIENT_Cleanup( );
Parameter	None
Return value	None
Example	// release initialization resources printf("CLIENT_Cleanup!\n"); CLIENT_Cleanup();
Note	When application program is closed , call this interface to release resources at last.

## CLIENT\_GetSDKVersion

Option	Instruction
Interface description	Interface for getting SDK version information.
Precondition	Have called CLIENT_Init interface.
Function	DWORD CLIENT_GetSDKVersion( );
Parameter	None
Return value	Return value is version,for example 34219000 corresponding to version 3.42 19000.
Example	//Get SDK version info DWORD dwNetSdkVersion = CLIENT_GetSDKVersion( ); printf("NetSDK version is [%d]\n", dwNetSdkVersion);
Note	None

## CLIENT\_GetLastError

Option	Instruction
Interface description	Interface for getting error code after failed calling interface,get current thread error code.
Precondition	Have called CLIENT_Init interface.

Function	DWORD CLIENT_GetLastError(void);
Parameter	None
Return value	Current thread error code
Example	<p>// according to error code, you can find corresponding explanation in dhnetSDK.h. It is to print hexadecimal here, not decimal shows in header file, please be careful with conversion.</p> <p>//Example: #define NET_NOT_SUPPORTED_EC(23)</p> <p>//Now SDK does not support this function, error code is 0x80000017, Decimal number 23 is hexadecimal 0x17.</p> <p>printf("Last Error[%x]\n" , CLIENT_GetLastError());</p>
Note	<p>Call this interface after failed calling interface.</p> <p>Because of too much error code, it is not illustrate here, user can search the following fields in dhnetSDK.h:</p> <p>//error code, corresponds with return value from CLIENT_GetLastError interface.</p> <p>#define _EC(x) (0x80000000 x)</p> <p>To find instruction of corresponding error code.</p>

## CLIENT\_SetAutoReconnect

Option	Instruction
Interface description	Interface for setting reconnection callback function after disconnection, when device gets offline, SDK will reconnect internally.
Precondition	Have called CLIENT_Init interface.
Function	<pre>void CLIENT_SetAutoReconnect(     HaveReConnect cbAutoConnect,     DWORD dwUser );</pre>
Parameter	<p>[in] cbAutoConnect</p> <p>Callback function set after the device reconnected successfully. After device reconnects successfully, SDK call this interface to note the user</p>

Option	Instruction
	<p>reconnection successful.</p> <p>[in] dwUser</p> <p>User data, set by user. Return to user for further use by callback function cbAutoConnect.</p>
Return value	None
Example	<p>//Not recommended to call SDK interface in SDK callback function,unless get current thread error code via CLIENT_GetLastError.</p> <p>//Callback function set after the device reconnected successfully.</p> <p>//When offline device is reconnected successfully,SDKwill call this function, set the callback function in CLIENT_SetAutoReconnect.</p> <pre>void CALLBACK HaveReConnect(LLONG ILoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser) {     printf("Call HaveReConnect\n");     printf("ILoginID[0x%x]", ILoginID);     if (NULL != pchDVRIP)     {         printf("pchDVRIP[%s]\n", pchDVRIP);     }     printf("nDVRPort[%d]\n", nDVRPort);     printf("dwUser[%p]\n", dwUser);     printf("\n"); }</pre> <p>***** Above are callback function definition, the underneath are interface using examples.*****</p> <p>//Set CLIENT_SetAutoReconnect interface.Once HaveReConnect is set, when device gets offline, SDK will reconnect internally.</p> <pre>CLIENT_SetAutoReconnect(&amp;HaveReConnect, 0);</pre>
Note	After setting callback function cbAutoConnect, once device gets

Option	Instruction
	<p>disconnected, SDK will try to reconnect to device constantly. If reconnection is successful, SDK will inform user via the callback function.</p> <p>If the interface is not called or callback function cbAutoConnect in the interface is NULL , when device-end gets disconnected, SDK will not try to reconnect to device.</p>

## CLIENT\_SetConnectTime

Option	Instruction
Interface description	Interface for setting device connection timeout value and trial times.
Precondition	Have called CLIENT_Init interface.
Function	<pre>void CLIENT_SetConnectTime(     int nWaitTime,     int nTryTimes );</pre>
Parameter	<p>nWaitTime</p> <p><i>[in]</i> The timeout time means waiting time for device's answer in every login.</p> <p>nTryTimes</p> <p><i>[in]</i> The trial time means the times of trying to connect device in every login.</p>
Return value	None
Example	<pre>// Set device connection timeout time and trial times. //This is optional. int nWaitTime = 5000; // timeout time is 5 seconds int nTryTimes = 3; //If timeout,it will try to log in three times. CLIENT_SetConnectTime(nWaitTime, nTryTimes);</pre>
Note	If CLIENT_SetConnectTime interface is not called, nWaitTime is 5000 as default and nTryTimes is 1 as default.

## CLIENT\_SetNetworkParam

Option	Instruction
Interface description	Interface for setting log in network environment.
Precondition	Have called CLIENT_Init interface.
Function	<pre>void CLIENT_SetNetworkParam(     NET_PARAM *pNetParam );</pre>
Parameter	pNetParam  [in]used to provide net login parameter,please refer to structure <a href="#">NET_TIME</a>
Return value	None
Example	<pre>//Set the network login parameters,include login timeout time and trial times. NET_PARAM stuNetParm = {0}; stuNetParm.nWaittime = 10000; //Change login timeout value to 10s, other parameters still use default values. CLIENT_SetNetworkParam(&amp;stuNetParm);</pre>
Note	None

## CLIENT\_LoginEx2

Option	Instruction
Interface description	Extensive interface 2 for login, used to register user to device,could define device capacity user supported.
Precondition	Have called CLIENT_Init interface.
Function	<pre>LLONG CLIENT_LoginEx2(     const char *pchDVRIP,     WORD wDVRPort,     const char *pchUserName,     const char *pchPassword,</pre>

Option	Instruction
	<pre> EM_LOGIN_SPAC_CAP_TYPE emSpecCap, void* pCapParam, LPNET_DEVICEINFO Ex lpDeviceInfo, int *error = 0 ); </pre>
Parameter	<p>pchDVRIP  <i>[in]</i>device IP.  IP address of login device,not allowed over 128 bytes.</p> <p>wDVRPort  <i>[in]</i> device port which used to login device.</p> <p>pchUserName  <i>[in]</i> User name  Not allow over 64 bytes.</p> <p>pchPassword  <i>[in]</i> User password  Not allow over 64 bytes.</p> <p>emSpecCap  <i>[in]</i> supported capacity of device,please refer to enum  EM_LOGIN_SPAC_CAP_TYPE.</p> <p>pCapParam  <i>[in]</i>Additional parameter to emSpecCap,work in with emSpecCap,please refer to enum EM_LOGIN_SPAC_CAP_TYPE If emSpecCap value not has corresponding parameter instruction about pCapParam,it will input NULL.</p> <p>lpDeviceInfo  <i>[out]</i>If device login successfully, save part information of login device;if not,save part information of relevant login ,such as the remaining login times,please refer to structure NET_DEVICEINFO.</p> <p>error  <i>[out]</i> (If function returns success,this parameter value is</p>

Option	Instruction
	<p>insignificance),return login error code.</p> <p>For example:</p> <ul style="list-style-type: none"> <li>1- erroneous password</li> <li>2- not exist user name</li> <li>3-login timeout</li> <li>4-account logged in</li> <li>5- account locked</li> <li>6- account is blacklisted</li> <li>7-system is busy,resource is not enough.</li> <li>8-failed subconnection</li> <li>9-failed main connection</li> <li>10-over max connections</li> <li>11- only support the third agreement</li> <li>12- there is no USB in device</li> <li>13- client's lp don't have power login.</li> </ul>
Return value	<p>Succeeded return device ID,otherwise return 0.</p> <p>After login successfully,device ID can cooperate with SDK interface to operate equipment.</p>
Example	<pre>//Login device NET_DEVICEINFO_Ex stDevInfoEx = {0}; int nError = 0; g_ILoginHandle=CLIENT_LoginEx2(g_szDevIp,g_nPort,g_szUserName, g_szPasswd,EM_LOGIN_SPEC_CAP_TCP,NULL,&amp;stDevInfoEx, &amp;nError); if(0 == g_ILoginHandle) {     // according to error code, you can find corresponding explanation in     dhnetsdk.h. It is to print hexadecimal here, not decimal shows in header     file, please be careful with conversion.      //Example: #define NET_NOT_SUPPORTED_EC(23)</pre>

Option	Instruction
	<pre>//Now SDK does not support this function, error code is 0x80000017, Decimal number 23 is hexadecimal 0x17.  printf("CLIENT_LoginEx2 %s[%d]Failed!Last Error[%x]\n", g_szDevlp , g_nPort , CLIENT_GetLastError()); } else {     printf("CLIENT_LoginEx2 %s[%d] Success\n", g_szDevlp , g_nPort); }</pre>
Note	<p>After successful initialization,call this interface to register to designated device.</p> <p>If succeed, device handle will be returned for relevant function to call.</p> <p>Recommended client to use emSpecCap = EM_LOGIN_SPEC_CAP_TCP which means login under TCP mode.</p>

## CLIENT\_Logout

Option	Instruction
Interface description	Interface for logout.
Function	BOOL CLIENT_Logout(LLONG ILoginID);
Parameter	<p><i>ILoginID</i></p> <p>[in]device login handle ID.</p> <p>Return value of CLIENT_LoginEx2 interface.</p>
Return value	Succeeded return TRUE,otherwise return FALSE.
Example	<pre>printf("CLIENT_Logout!\n"); if(!CLIENT_Logout(g_ILoginHandle)) {     printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError()); }</pre>



	//Please refer to example code of sync login in device login.
Note	When logout device,related businesses will stop ,such as real-time preview etc.

## CLIENT\_RealPlayEx

Option	Instruction
Interface description	Extensive interface for real-time monitoring,realize get real-time monitoring data stream from logined device.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre>LLONG CLIENT_RealPlayEx(     LLONG ILoginID,     int nChannelID,     HWND hWnd,     DH_RealPlayType rType = DH_RType_Realplay );</pre>
Parameter	<p>ILoginID [in] device login ID. Return value of corresponding device login interface CLIENT_LoginEx2.</p> <p>nChannelID [in] real-time monitoring channel number which starts from 0.</p> <p>hWnd [in] window handle,when value is 0, data is not decoded and image is not displayed.</p> <p>rType [in] real-time monitoring type. Default type is DH_RType_Realplay,please refer to definition of enum <a href="#">DH_RealPlayType</a>.</p>
Return value	Failed return 0,otherwise return real-time monitoring ID(real-time monitoring handle) and used as related function parameter.
Example	typedef HWND (WINAPI *PROCGETCONSOLEWINDOW)();

Option	Instruction
	<pre> PROCGETCONSOLEWINDOW GetConsoleWindow;  //Get the console window handle.  HMODULE hKernel32 = GetModuleHandle("kernel32");  GetConsoleWindow = (PROCGETCONSOLEWINDOW)GetProcAddress(hKernel32,"GetConsoleWi ndow");  HWND hWnd = GetConsoleWindow();  //Start real-time monitoring.  int nChannelID = 0; // preview channel NO.  DH_RealPlayType emRealPlayType = DH_RType_Realplay;  g_IRealHandle = CLIENT_RealPlayEx(gILoginHandle, nChannelID, hWnd, emRealPlayType);  if (g_IRealHandle == 0) {     printf("CLIENT_RealPlayEx: failed! Error code: %x.\n",         CLIENT_GetLastError()); } </pre>
Note	<p>NVR device fills nChannelID as video output channel number in multiplay preview.</p> <p>According to device information got when login, you can open any real-time monitoring and display it in any designated window by calling this interface. If succeeded, real-time monitoring ID is returned for more operation and control.</p>

## CLIENT\_StopRealPlayEx

Option	Instruction
Interface description	Extensive interface for stopping real-time monitoring , stop getting real-time monitoring data stream from logged device.
Precondition	Have called interface for getting real-time monitoring data stream ,such as CLIENT_RealPlayEx interface.

Option	Instruction
Function	<pre> BOOL CLIENT_StopRealPlayEx (     LLONG IRealHandle ); </pre>
Parameter	<p>IRealHandle</p> <p><i>[in]</i> real-time monitoring handle.</p> <p>Return value of getting real-time monitoring data stream interface as <a href="#">CLIENT_RealPlayEx</a> etc.</p>
Return value	Succeeded return TRUE, otherwise return FALSE.
Example	<pre> if (!CLIENT_StopRealPlayEx(g_IRealHandle)) {     printf("CLIENT_StopRealPlayEx Failed, g_IRealHandle[%x]!Last     Error[%x]\n" , g_IRealHandle, CLIENT_GetLastError()); } </pre>
Note	None

## CLIENT\_SetRealDataCallBackEx

Option	Instruction
Interface description	Extensive interface for setting real-time monitoring data callback function.
Precondition	Have called get real-time monitoring data stream interface as <a href="#">CLIENT_RealPlayEx</a> etc.
Function	<pre> BOOL CLIENT_SetRealDataCallBackEx(     LLONG IRealHandle,     <a href="#">fRealDataCallBackEx</a> cbRealData,     LDWORD dwUser,     DWORD dwFlag ); </pre>
Parameter	IRealHandle

Option	Instruction														
	<p><i>[in]</i> real-time monitoring handle.</p> <p>Return value of getting real-time monitoring data stream interface as CLIENT_RealPlayEx etc.</p> <p>cbRealData</p> <p><i>[in]</i> real-time monitoring data callback function</p> <p>If cbRealData value is 0,not call back real-time monitoring data;</p> <p>If cbRealData value is not 0,call back real-time monitoring data via callback function cbRealData,please refer to callback function (fRealDataCallBackEx) for details.</p> <p>dwUser</p> <p><i>[in]</i> User data,SDK sends the data to user for further use via callback function fRealDataCallBackEx.</p> <p>dwFlag</p> <p><i>[in]</i> flag of callback data selection</p> <p>we can recall data selectively as needed, do not recall data type which is not set,different value corresponds to different data type:</p> <table> <tr> <th>dwFlag</th><th>Data type</th></tr> <tr> <td>0x00000001</td><td>Equal with original data</td></tr> <tr> <td>0x00000002</td><td>Standard data of MPEG4/H264</td></tr> <tr> <td>0x00000004</td><td>YUV data</td></tr> <tr> <td>0x00000008</td><td>PCM data</td></tr> <tr> <td>0x00000010</td><td>Original audio data</td></tr> <tr> <td>0x0000001f</td><td>The above five types of data</td></tr> </table>	dwFlag	Data type	0x00000001	Equal with original data	0x00000002	Standard data of MPEG4/H264	0x00000004	YUV data	0x00000008	PCM data	0x00000010	Original audio data	0x0000001f	The above five types of data
dwFlag	Data type														
0x00000001	Equal with original data														
0x00000002	Standard data of MPEG4/H264														
0x00000004	YUV data														
0x00000008	PCM data														
0x00000010	Original audio data														
0x0000001f	The above five types of data														
Return value	If succeeded,return TRUE,otherwise return FALSE.														
Example	<p>// It's not recommended to call SDK interface in callback function, unless call CLIENT_GetLastError to get error code in current process.</p> <p>//prototype of real-time monitor data callback function --extensive</p> <p>//The callback function is set by CLIENT_ SetRealDataCallBackEx, When</p>														

Option	Instruction
	<p>real-time data is received, SDK will call function.</p> <p>//Recommend users only save data in this callback, In the other word:copy data to your storage, and encode or decode data after leaving callback.</p> <p>//Not recommend users encode or decode data directly in callback function.</p> <pre> void CALLBACK RealDataCallBackEx(LLONG IRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser) {     if (IRealHandle == g_IRealHandle)     {         switch(dwDataType)         {             case 0:                 //original A/V hybrid data                 printf("receive      real      data,param:IRealHandle[%p], dwDataType[%d],pBuffer[%p],dwBufSize[%d], param[%p],dwUser[%p]\n",IRealHandle,      dwDataType, pBuffer, dwBufSize, param, dwUser);                 break;             case 1:                 // standard video data                 break;             case 2:                 //yuv data                 break;             case 3:                 //pcm audio data                 break;             case 4: </pre>

Option	Instruction
	<pre> //original audio data break;  default: break;  }  }  }  }  }  ***** Above are callback function definition, the underneath are interface using examples. *****  DWORD dwFlag = 0x00000001; if(!CLIENT_SetRealDataCallBackEx(g_IRealHandle,&amp;RealDataCallBackEx, NULL, dwFlag)) { printf("CLIENT_SetRealDataCallBackEx: failed! Error code: %x.\n", CLIENT_GetLastError()); } </pre>
Note	Add a callback data type flag- dwFlag, we can recall data selectively as needed, do not recall data type which is not set.

## CLIENT\_FindFile

Option	Instruction
Interface description	Interface for opening record query handle.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> LLONG CLIENT_FindFile(     LLONG ILoginID,     int nChannelId,     int nRecordFileType, </pre>

Option	Instruction										
	char* cardid, <a href="#">LPNET_TIME</a> time_start, <a href="#">LPNET_TIME</a> time_end, BOOL bTime, int waittime );										
Parameter	ILLoginID [in]device login ID. Return value of login interface CLIENT_LoginEx2. nChannelId [in] channel ID,start from 0. nRecordFileType [in] Record file type. Different value corresponds with different record file type,please refer to enum <a href="#">EM_QUERY_RECORD_TYPE</a> cardid [in] extensive parameter,works in with nRecordFileType. <table> <tr> <th>nRecordFileType</th><th>cardid</th></tr> <tr> <td>EM_RECORD_TYPE_CARD</td><td>card number</td></tr> <tr> <td>EM_RECORD_TYPE_CONDI TION</td><td>card number &amp;&amp;transaction type&amp;&amp;transaction amount (such as wish to skip a empty word)</td></tr> <tr> <td>EM_RECORD_TYPE_CARD_ PICTURE</td><td>card number</td></tr> <tr> <td>EM_RECORD_TYPE_FIELD</td><td>FELD1&amp;&amp;FELD2&amp;&amp; FELD3&amp;&amp;( such as wish to skip a empty word,corresponding position</td></tr> </table>	nRecordFileType	cardid	EM_RECORD_TYPE_CARD	card number	EM_RECORD_TYPE_CONDI TION	card number &&transaction type&&transaction amount (such as wish to skip a empty word)	EM_RECORD_TYPE_CARD_ PICTURE	card number	EM_RECORD_TYPE_FIELD	FELD1&&FELD2&& FELD3&&( such as wish to skip a empty word,corresponding position
nRecordFileType	cardid										
EM_RECORD_TYPE_CARD	card number										
EM_RECORD_TYPE_CONDI TION	card number &&transaction type&&transaction amount (such as wish to skip a empty word)										
EM_RECORD_TYPE_CARD_ PICTURE	card number										
EM_RECORD_TYPE_FIELD	FELD1&&FELD2&& FELD3&&( such as wish to skip a empty word,corresponding position										

Option	Instruction
	<p>is null)</p> <p>In addition to the above situation, cardid value is null.</p> <p>tmStart [in] start time of query record. Please refer to structure NET_TIME.</p> <p>tmEnd [in] end time of query record. Please refer to structure NET_TIME.</p> <p>bTime [in] whether query by time This parameter is invalid now, recommend to input FALSE.</p> <p>waittime [in] wait time.</p>
Return value	Succeeded return record query handle, otherwise return 0.
Example	<pre> NET_TIME StartTime = {0}; NET_TIME StopTime = {0}; StartTime.dwYear = 2015; StartTime.dwMonth = 9; StartTime.dwDay = 20; StartTime.dwHour = 0; StartTime.dwMinute = 0; StopTime.dwYear = 2015; StopTime.dwMonth = 9; StopTime.dwDay = 21; StopTime.dwHour = 15; NET_RECORDFILE_INFO netFileInfo[30] = {0}; int nFileCount = 0; //Get record query handle. if(!CLIENT_FindFile (ILoginHandle, nChannelID,</pre>



Option	Instruction
	<pre>(int)EM_RECORD_TYPE_ALL, NULL, &amp;StartTime, &amp;StopTime, FALSE, 5000)) {     printf("CLIENT_FindFile: failed! Error code: %x.\n",         CLIENT_GetLastError()); }</pre>
Note	<p>You can call this interface for querying video record before playback, then call CLIENT_FindNextFile function to return a detailed video record for playing, after query is finished, call CLIENT_FindClose to close query handle.</p>

## CLIENT\_FindNextFile

Option	Instruction
Interface description	Interface for searching record file.
Precondition	Have called get record query handle interface as CLIENT_FindFile.
Function	<pre>int CLIENT_FindNextFile(     LLONG IFindHandle,     <a href="#">LPNET_RECORDFILE_INFO</a> lpFindData );</pre>
Parameter	<p>IFindHandle [in] record query handle. Correspond with return value of CLIENT_FindFile interface.</p> <p>lpFindData [out] buffer of record file information. Used to output queried record file information, please refer to structure NET_RECORDFILE_INFO.</p>
Return value	<p>1: take back a video record successfully.</p> <p>0: Video record has been exhausted.</p> <p>-1: parameter error.</p>
Example	NET_RECORDFILE_INFO struFileData = {0};

Option	Instruction
	<pre> int result = CLIENT_FindNextFile(IFindHandle, &amp; struFileData);  if(result == 1) //Take back a record file information. {     //Save record file. }  else if(result == 0) // Data of record file information has been exhausted. {     ; }  else //Parameter error. {     printf("CLIENT_FindNextFile: failed! Error code:0x%x.\n",         CLIENT_GetLastError()); } </pre>
Note	<p>Call CLIENT_FindFile for opening query handle before call this interface.</p> <p>One call returns one video record information.</p>

## CLIENT\_FindClose

Option	Instruction
Interface description	Interface for closing record query handle.
Precondition	Have called get record query handle interface as CLIENT_FindFile.
Function	<pre> BOOL CLIENT_FindClose(     LLONG IFindHandle ); </pre>
Parameter	<p>IFindHandle</p> <p><i>[in]</i> record query handle.</p> <p>Correspond with return value of CLIENT_FindFile interface.</p>
Return value	Succeeded return TRUE,otherwise return FALSE.

Option	Instruction
Example	<pre>if(!CLIENT_FindClose (IFindHandle)) {     printf("CLIENT_FindNextFile: failed! Error code:0x%x.\n",         CLIENT_GetLastError()); }</pre>
Note	Call CLIENT_FindFile for opening query handle,after query is finished,it is needed to call this function to close query handle,and release resources.

## CLIENT\_PlayBackByTimeEx

Option	Instruction
Interface description	Extensive interface for playing back by time
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre>LLONG CLIENT_PlayBackByTimeEx(     LLONG ILoginID,     int nChannelID,     <a href="#">LPNET_TIME</a> lpStartTime,     <a href="#">LPNET_TIME</a> lpStopTime,     HWND hWnd,     <a href="#">fDownloadPosCallBack</a> cbDownloadPos,     LDWORD dwPosUser,     <a href="#">fDataCallBack</a> fDownloadDataCallBack,     LDWORD dwDataUser );</pre>
Parameter	<p>ILoginID <i>[in]</i> device login ID. Corresponding with return value of login interface CLIENT_LoginEx2.</p> <p>nChannelID <i>[in]</i> channel ID, start from 0.</p>

Option	Instruction
	<p>lpStartTime</p> <p><i>[in]</i> start time of playback.</p> <p>Please refer to structure NET_TIME for details.</p> <p>lpStopTime</p> <p><i>[in]</i>end time of playback.</p> <p>Please refer to structure NET_TIME for details.</p> <p>hWnd</p> <p><i>[in]</i> playback window.</p> <p>cbDownloadPos</p> <p><i>[in]</i> progress callback of user parameter</p> <p>If cbDownloadPos value is 0,not call back playback data process;</p> <p>If cbDownloadPos value is not 0,call back playback data process via callback function cbDownloadPos,please refer to fDownloadPosCallBack callback function for details.</p> <p>dwPosUser</p> <p><i>[in]</i> user data.</p> <p>SDK sends the data to user for further use via callback function fDownloadPosCallBack.</p> <p>fDownloadDataCallBack</p> <p><i>[in]</i> record data callback function.</p> <p>If fDownloadDataCallBack value is 0,not call back record playback data;if fDownloadDataCallBack value is not 0,call back record playback data via callback function fDownloadDataCallBack,please refer to fDataCallBack for details.</p> <p>dwDataUser</p> <p><i>[in]</i> user data.</p> <p>SDK sends the data to user for further use via callback function fDownloadDataCallBack.</p>
Return value	If succeeded,return record playback handle ID,otherwise return 0.

Option	Instruction
Example	<pre>// this demo decoding by playsdk library typedef HWND (WINAPI *PROCGETCONSOLEWINDOW)(); PROCGETCONSOLEWINDOW GetConsoleWindow; // Get console window handle HMODULE hKernel32 = GetModuleHandle("kernel32"); GetConsoleWindow = (PROCGETCONSOLEWINDOW)GetProcAddress(hKernel32,"GetConsoleWi ndow"); HWND hWnd = GetConsoleWindow(); int nChannelID = 0; //channel NO NET_TIME stuStartTime = {0}; stuStartTime.dwYear = 2015; stuStartTime.dwMonth = 9; stuStartTime.dwDay = 3; NET_TIME stuStopTime = {0}; stuStopTime.dwYear = 2015; stuStopTime.dwMonth = 9; stuStopTime.dwDay = 12; g_IPlayHandle = CLIENT_PlayBackByTimeEx(gILoginHandle, nChannelID, &amp;stuStartTime, &amp;stuStopTime, hWnd, NULL, NULL, NULL, NULL); if (g_IPlayHandle == 0) {     printf("CLIENT_PlayBackByTimeEx: failed! Error code: 0x%x.\n",         CLIENT_GetLastError()); }</pre>
Note	<p>Parameter hWnd and fDownloadDataCallBack can not be NULL at the same time,otherwise interface will return 0.</p>

## CLIENT\_StopPlayBack

Option	Instruction
Interface description	Interface for stopping record playback.
Precondition	Have called get record playback handle interface as CLIENT_PlayBackByTimeEx etc.
Function	<pre> BOOL CLIENT_StopPlayBack(     LLONG IPlayHandle ); </pre>
Parameter	<p>IPlayHandle</p> <p><i>[in]</i> record playback handle.</p> <p>Correspond with return value of CLIENT_PlayBackByTimeEx interface etc.</p>
Return value	Succeeded return TRUE,otherwise return FALSE.
Example	<pre> if (!CLIENT_StopPlayBack(g_IPlayHandle)) {     printf("CLIENT_StopPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" , g_IPlayHandle, CLIENT_GetLastError()); } </pre>
Note	users get record playback handle via CLIENT_PlayBackByTimeEx interface etc,and close record playback handle via CLIENT_StopPlayBack.

## CLIENT\_GetPlayBackOsdTime

Option	Instruction
Interface description	<p>Interface for getting playback OSD time.</p> <p>Only when parameter hWnd in record playback interface is valid, can parameter got by this interface be valid,otherwise it is insignificant.</p>
Precondition	Have called get record playback handle interface as CLIENT_PlayBackByTimeEx etc.
Function	<pre> BOOL CLIENT_GetPlayBackOsdTime( </pre>

Option	Instruction
	<pre> LLONG IPlayHandle, LPNET_TIME lpOsdTime, LPNET_TIME lpStartTime, LPNET_TIME lpEndTime ); </pre>
Parameter	<p>IPlayHandle  <i>[in]</i>record playback handle.  Correspond with return value of CLIENT_PlayBackByTimeEx interface etc.</p> <p>lpOsdTime  <i>[out]</i> OSD time  Please refer to structure NET_TIME for details.</p> <p>lpStartTime  <i>[out]</i>start time of playback.  Please refer to structure NET_TIME for details.</p> <p>lpEndTime  <i>[out]</i>end time of playback.  Please refer to structure NET_TIME for details.</p>
Return value	Succeeded return TRUE,otherwise return FALSE.
Example	<pre> NET_TIME stuOsdTime = {0}; NET_TIME stuStartTime = {0}; NET_TIME stuEndTime = {0}; if (!CLIENT_GetPlayBackOsdTime (g_IPlayHandle, &amp;stuOsdTime, &amp;stuStartTime, &amp;stuEndTime)) {     printf("CLIENT_ GetPlayBackOsdTime Failed, g_IPlayHandle[%x]!Last Error[%x]\n" , g_IPlayHandle, CLIENT_GetLastError()); } </pre>
Note	Only when parameter hWnd in record playback interface is valid, can parameter got by this interface be valid,otherwise it is insignificant.

## CLIENT\_QueryRecordFile

Option	Instruction
Interface description	Interface for querying all files in pointed time period.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre>BOOL CLIENT_QueryRecordFile(     LLONG ILoginID,     int nChannelId,     int nRecordFileType,     <a href="#">LPNET_TIME</a> tmStart,     <a href="#">LPNET_TIME</a> tmEnd,     char* pchCardid,     <a href="#">LPNET_RECORDFILE_INFO</a> nriFileinfo,     int maxlen,     int *filecount,     int waittime=1000,     BOOL bTime = FALSE );</pre>
Parameter	<p>ILoginID <i>[in]</i> device login ID. Correspond with return value of login interface CLIENT_LoginEx2.</p> <p>nChannelId <i>[in]</i> channel ID,start from 0.</p> <p>nRecordFileType <i>[in]</i> record file type. Different value corresponds with different record file type,please refer to enum <a href="#">EM_QUERY_RECORD_TYPE</a>.</p> <p>tmStart <i>[in]</i> start time of record query.</p>



Option	Instruction										
	<p>Please refer to structure NET_TIME.</p> <p>tmEnd</p> <p><i>[in]</i> end time of record query.</p> <p>Please refer to structure NET_TIME.</p> <p>pchCardid</p> <p><i>[in]</i> extensive parameter,work in with nRecordFileType.</p> <table> <tr> <th>nRecordFileType</th><th>pchCardid</th></tr> <tr> <td>EM_RECORD_TYPE_CARD</td><td>card number</td></tr> <tr> <td>EM_RECORD_TYPE_CONDITION</td><td>card number &amp;&amp;transaction type&amp;&amp;transaction amount (such as wish to skip a empty word)</td></tr> <tr> <td>EM_RECORD_TYPE_CARD_PICTURE</td><td>card number</td></tr> <tr> <td>EM_RECORD_TYPE_FIELD</td><td>FELD1&amp;&amp;FELD2&amp;&amp;FELD3 &amp;&amp;( such as wish to skip a empty word,corresponding position is null)</td></tr> </table> <p>In addition to the above situation, pchCardid value is null.</p> <p>nriFileinfo</p> <p><i>[out]</i> record file information of returned.</p> <p>This is a pointer of structure array of NET_RECORDFILE_INFO,please refer to structure NET_RECORDFILE_INFO for details.</p> <p>maxlen</p> <p><i>[in]</i>max length of nriFileinfo buffer.</p> <p>(unit:byte,recommend length: (100~200)*sizeof(NET_RECORDFILE_INFO)</p> <p>filecount</p> <p><i>[out]</i>file count of returned.</p>	nRecordFileType	pchCardid	EM_RECORD_TYPE_CARD	card number	EM_RECORD_TYPE_CONDITION	card number &&transaction type&&transaction amount (such as wish to skip a empty word)	EM_RECORD_TYPE_CARD_PICTURE	card number	EM_RECORD_TYPE_FIELD	FELD1&&FELD2&&FELD3 &&( such as wish to skip a empty word,corresponding position is null)
nRecordFileType	pchCardid										
EM_RECORD_TYPE_CARD	card number										
EM_RECORD_TYPE_CONDITION	card number &&transaction type&&transaction amount (such as wish to skip a empty word)										
EM_RECORD_TYPE_CARD_PICTURE	card number										
EM_RECORD_TYPE_FIELD	FELD1&&FELD2&&FELD3 &&( such as wish to skip a empty word,corresponding position is null)										

Option	Instruction
	<p>Output parameter only queries to the biggest video record which buffer is full.</p> <p>waittime  <i>[in]</i> wait time.</p> <p>bTime  <i>[in]</i> whether query by time</p> <p>This parameter is invalid now, recommended to input FALSE.</p>
Return value	Succeeded return TRUE, otherwise return FALSE.
Example	<pre> NET_TIME StartTime = {0}; NET_TIME StopTime = {0}; StartTime.dwYear = 2015; StartTime.dwMonth = 9; StartTime.dwDay = 20; StartTime.dwHour = 0; StartTime.dwMinute = 0; StopTime.dwYear = 2015; StopTime.dwMonth = 9; StopTime.dwDay = 21; StopTime.dwHour = 15; NET_RECORDFILE_INFO netFileInfo[30] = {0}; int nFileCount = 0; //query record file if(!CLIENT_QueryRecordFile(ILLoginHandle, nChannelID, (int)EM_RECORD_TYPE_ALL, &amp;StartTime, &amp;StopTime, NULL, &amp;netFileInfo[0], sizeof(netFileInfo), &amp;nFileCount, 5000, FALSE)) {     printf("CLIENT_QueryRecordFile: failed! Error code: %x.\n",     CLIENT_GetLastError()); } </pre>
Note	Before playback by file, it's needed to call this interface to query video record. If

Option	Instruction
	video record info of input time slot queried is greater than defined buffer size, return video record which size is defined buffer size, and can continue querying as needed.

## CLIENT\_DownloadByTimeEx

Option	Instruction
Interface description	Extensive interface for downloading record by time
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> LLONG CLIENT_DownloadByTimeEx(     LLONG ILoginID,     int nChannelId,     int nRecordFileType,     LPNET_TIME tmStart,     LPNET_TIME tmEnd,     char *sSavedFileName,     fTimeDownLoadPosCallBack cbTimeDownLoadPos,     LDWORD dwUserData,     fDataCallBack fDownLoadDataCallBack,     LDWORD dwDataUser,     void* pReserved = NULL ); </pre>
Parameter	<p>ILoginID  <i>[in]</i> logged device ID  Return value of login interface CLIENT_LoginEx2.</p> <p>nChannelId  <i>[in]</i> channel NO., start from 0.</p> <p>nRecordFileType  <i>[in]</i> record file type</p>

Option	Instruction
	<p>See enum EM_QUERY_RECORD_TYPE for details</p> <p>tmStart</p> <p><i>[in]</i> record download start time</p> <p>See struct NET_TIME for details</p> <p>tmEnd</p> <p><i>[in]</i> record download end time</p> <p>See struct NET_TIME for details</p> <p>sSavedFileName</p> <p><i>[in]</i> name of the record file to be saved</p> <p>Recommended to use the full path name</p> <p>cbTimeDownLoadPos</p> <p><i>[in]</i> download progress callback function</p> <p>See callback function fTimeDownLoadPosCallBack for details.</p> <p>dwUserData</p> <p><i>[in]</i> user data of download progress callback function</p> <p>SDK send the data to user for further use via callback function fTimeDownLoadPosCallBack</p> <p>fDownLoadDataCallBack</p> <p><i>[in]</i> download data callback function</p> <p>See callback function fDataCallBack for details.</p> <p>dwDataUser</p> <p><i>[in]</i> user data of download data callback function</p> <p>SDK send the data to user for further use via callback function fDataCallBack</p> <p>pReserved</p> <p><i>[in]</i>reserved parameter</p> <p>Used for further extension, meaningless until now, default value is NULL.</p>
Return value	succeed return download ID, failed return 0.
Example	int nChannelID = 0; // channel NO.

Option	Instruction
	<pre> NET_TIME stuStartTime = {0}; stuStartTime.dwYear = 2015; stuStartTime.dwMonth = 9; stuStartTime.dwDay = 17; NET_TIME stuStopTime = {0}; stuStopTime.dwYear = 2015; stuStopTime.dwMonth = 9; stuStopTime.dwDay = 18;  // start record download.  // at least one of the two formal parameters sSavedFileName and fDownloadDataCallBack is valid.  g_IDownloadHandle = CLIENT_DownloadByTimeEx(gILoginHandle, nChannelID, EM_RECORD_TYPE_ALL, &amp;stuStartTime, &amp;stuStopTime, "test.dav", TimeDownloadPosCallBack, NULL, DataCallBack, NULL); if (g_IDownloadHandle == 0) {     printf("CLIENT_DownloadByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError()); } </pre>
Note	<p>sSavedFileName is not NULL, record data is written to the corresponding file.</p> <p>fDownloadDataCallBack is not NULL, record data is returned via this callback function.</p> <p>After download, call CLIENT_StopDownload to close download handle.</p>

## CLIENT\_StopDownload

Option	Instruction
Interface description	Interface for stopping record download
Precondition	Have called record download interface as CLIENT_DownloadByTimeEx etc.

Option	Instruction
Function	<pre> BOOL CLIENT_StopDownload(     LLONG IFileHandle ); </pre>
Parameter	<p>IFileHandle</p> <p><i>[in]</i>download handle</p> <p>Return value of record download interface as CLIENT_DownloadByTimeEx etc.</p>
Value	Succeed return TRUE, failed return FALSE
Example	<pre> // stop download,we can call this interface after download end or during download. if (g_IDownloadHandle) {     if (!CLIENT_StopDownload(g_IDownloadHandle))     {         printf("CLIENT_StopDownload Failed, g_IDownloadHandle[%x] !Last Error[%x]\n" , g_IDownloadHandle, CLIENT_GetLastError());     } } </pre>
Note	We can stop download after download end or during download as needed.

## CLIENT\_PlayBackByRecordFileEx

Option	Instruction
Interface description	Extensive interface for playing back by file
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> LLONG CLIENT_PlayBackByRecordFileEx(     LLONG ILoginID,     <a href="#">LPNET_RECORDFILE_INFO</a> lpRecordFile, </pre>

Option	Instruction
	<p> HWND hWnd,  <a href="#">fDownloadPosCallBack</a> cbDownloadPos,  LDWORD dwPosUser,  <a href="#">fDataCallBack</a> fDownloadDataCallBack,  LDWORD dwDataUser  ); </p>
Parameter	<p> ILoginID  [in] logged device ID  Return value of login interface CLIENT_LoginEx2.  lpRecordFile  [in] record file information  Obtained by record information search interface as CLIENT_FindNextFile.  See struct NET_RECORDFILE_INFO for details.  hWnd  [in] playback window handle  cbDownloadPos  [in] record progress callback function  If cbDownloadPos is 0, not call back playback data progress.  If cbDownloadPos is not 0, call back playback data progress via callback function cbDownloadPos, see callback function fDownloadPosCallBack for details.  dwPosUser  [in] user data  SDK send the data to user for further use via callback function fDownloadPosCallBack  fDownloadDataCallBack  [in] record data callback function  If fDownloadDataCallBack is 0, not call back playback data.  If fDownloadDataCallBack is not 0, call back playback data progress via </p>

Option	Instruction
	<p>callback function fDownloadDataCallBack, see callback function fDataCallBack for details.</p> <p>dwDataUser</p> <p>[in] user data</p> <p>SDK send the data to user for further use via callback function fDownloadDataCallBack</p>
Return value	Succeed return playback handle, failed return 0.
Example	<pre>// formal parameter hWnd must be valid. //stuNetFileInfo information is obtained by together of three interfaces CLIENT_FindFile,CLIENT_FindNextFile and CLIENT_FindClose. g_IPlayHandle = CLIENT_PlayBackByRecordFileEx(gILoginHandle, &amp;stuNetFileInfo, hWnd, NULL, NULL, NULL, NULL); if (g_IPlayHandle == 0) {     printf("CLIENT_PlayBackByRecordFileEx: failed! Error code: %x.\n",         CLIENT_GetLastError()); }</pre>
Note	Parameter hWnd and fDownloadDataCallBack can not be NULL at the same time, otherwise the interface will be failed.

## CLIENT\_PausePlayBack

Option	Instruction
Interface description	<p>Interface for pause or restoring playback</p> <p>Only when the parameter hWnd in open record playback interface is valid, can parameters this interface obtained be valid, otherwise it is no meaning.</p>
Precondition	Hava called interface as CLIENT_ <a href="#">PlayBackByTimeEx</a> etc. to obtain record playback handle.
Function	BOOL CLIENT_PausePlayBack(LLONG IPlayHandle, BOOL bPause);
Parameter	IPlayHandle



Option	Instruction
	<p><i>[in]</i> record playback handle</p> <p>Return value of interface as CLIENT_PlayBackByTimeEx etc.</p> <p>bPause</p> <p><i>[in]</i> flag of pause or restore playing</p> <p>TRUE :pause FALSE :restore playing</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> if (!CLIENT_ PausePlayBack (g_IPlayHandle)) {     printf("CLIENT_   PausePlayBack   Failed,   g_IPlayHandle[%x]! Last     Error[%x]\n" , g_IPlayHandle, CLIENT_GetLastError()); } </pre>
Note	Only when the parameter hWnd in open record playback interface is valid, can parameters this interface obtained be valid, otherwise it is no meaning.

## CLIENT\_SeekPlayBack

Option	Instruction
Interface description	Interface for positioning record playback start point
Precondition	Hava called interface as <a href="#">CLIENT_PlayBackByTimeEx</a> etc. to obtain record playback handle.
Function	<pre> BOOL CLIENT_SeekPlayBack(     LLONG IPlayHandle,     unsigned int offsettime,     unsigned int offsetbyte ); </pre>
Parameter	<p>IPlayHandle</p> <p><i>[in]</i> record playback handle</p> <p>Return value of interface as CLIENT_PlayBackByTimeEx etc.</p> <p>offsettime</p>

Option	Instruction
	<p><i>[in]</i> relative offset of start time(unit : s)</p> <p>offsetbyte</p> <p><i>[in]</i> this parameter is abolished</p> <p>Set to 0xffffffff.</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre>int nOffsetSeconds = 2 * 60 * 60; // drag to 2*60*60s after stuStartTime to start play. if (FALSE == CLIENT_SeekPlayBack (g_IPlayHandle, nOffsetSeconds, 0xffffffff)) {     printf("CLIENT_SeekPlayBack Failed, g_IPlayHandle[%x]!Last Error[%x]\n" , g_IPlayHandle, CLIENT_GetLastError()); }</pre>
Note	None

## CLIENT\_FastPlayBack

Option	Instruction
Interface description	<p>Interface for fast play. Increasing frame rate by 1x</p> <p>Only when the parameter hWnd in open record playback interface is valid, can parameters this interface obtained be valid, otherwise it is no meaning.</p>
Precondition	Hava called interface as <a href="#">CLIENT_PlayBackByTimeEx</a> etc. to obtain record playback handle.
Function	<pre>BOOL CLIENT_FastPlayBack(     LLONG IPlayHandle );</pre>
Parameter	<p>IPlayHandle</p> <p><i>[in]</i> record playback handle</p> <p>Return value of interface as <a href="#">CLIENT_PlayBackByTimeEx</a> etc.</p>

Option	Instruction
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> if (!CLIENT_ FastPlayBack (g_IPlayHandle)) {     printf("CLIENT_ FastPlayBack Failed, g_IPlayHandle[%x]!Last     Error[%x]\n" , g_IPlayHandle, CLIENT_GetLastError()); } </pre>
Note	<p>Can not fast forward without limit, currently the max frame is 200, greater than this will return FALSE. if the record is with sound, fast forward is not supported.</p> <p>Only when the parameter hWnd in open record playback interface is valid, can parameters this interface obtained be valid, otherwise it is no meaning.</p>

## CLIENT\_SlowPlayBack

Option	Instruction
Interface description	Interface for slow play. Decreasing frame rate by 1x
Precondition	Hava called interface as CLIENT_PlayBackByTimeEx etc. to obtain record playback handle.
Function	<pre> BOOL CLIENT_SlowPlayBack (     LLONG IPlayHandle ); </pre>
Parameter	<p>IPlayHandle</p> <p><i>[in]</i> record playback handle</p> <p>Return value of interface as CLIENT_PlayBackByTimeEx etc.</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> if (!CLIENT_SlowPlayBack (g_IPlayHandle)) {     printf("CLIENT_SlowPlayBack Failed, g_IPlayHandle[%x]!Last </pre>

Option	Instruction
	<pre>Error[%x]\n" , g_IPlayHandle, CLIENT_GetLastError()); }</pre>
Note	<p>The min frame is 1, less than 1 will return FALSE.</p> <p>when the parameter hWnd in open record playback interface is 0 and device supports playback speed control, SDK can send speed control command to device.</p> <p>when the parameter hWnd in open record playback interface is a valid value and device supports playback speed control, SDK can send speed control command to device and call the command to playsdk library which display records.</p>

## CLIENT\_NormalPlayBack

Option	Instruction
Interface description	Interface for restoring normal play speed
Precondition	Hava called interface as <a href="#">CLIENT_PlayBackByTimeEx</a> etc. to obtain record playback handle.
Function	<pre>BOOL CLIENT_NormalPlayBack(     LLONG IPlayHandle );</pre>
Parameter	<p>IPlayHandle</p> <p><i>[in]</i> record playback handle</p> <p>Return value of interface as <a href="#">CLIENT_PlayBackByTimeEx</a> etc.</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre>if (!CLIENT_NormalPlayBack (g_IPlayHandle)) {     printf("CLIENT_NormalPlayBack    Failed,    g_IPlayHandle[%x]!Last     Error[%x]\n" , g_IPlayHandle, CLIENT_GetLastError()); }</pre>

Option	Instruction
Note	<p>when the parameter hWnd in open record playback interface is 0 and device supports playback speed control, SDK can send speed control command to device.</p> <p>when the parameter hWnd in open record playback interface is a valid value and device supports playback speed control, SDK can send speed control command to device and call the command to playsdk library which display records.</p>

## CLIENT\_DownloadByRecordFileEx

Option	Instruction
Interface description	Extensive interface for downloading record by file
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> LLONG CLIENT_DownloadByRecordFileEx(     LLONG ILoginID,     LPNET_RECORDFILE_INFO lpRecordFile,     char *sSavedFileName,     fDownLoadPosCallBack cbDownLoadPos,     LDWORD dwUserData,     fDataCallBack fDownLoadDataCallBack,     LDWORD dwDataUser,     void* pReserved = NULL ); </pre>
Parameter	<p>ILoginID [in] logged device ID Return value of login interface CLIENT_LoginEx2.</p> <p>lpRecordFile [in] record file information pointer Obtained by record search interface <a href="#">CLIENT_QueryRecordFile</a>, see</p>

Option	Instruction
	<p>NET_RECORDFILE_INFO for details.</p> <p>sSavedFileName</p> <p><i>[in]</i> name of the record file to be saved</p> <p>Recommended to use the full path name</p> <p>cbDownLoadPos</p> <p><i>[in]</i> record progress callback function</p> <p>If cbDownLoadPos is 0, not call back playback data progress.</p> <p>If cbDownLoadPos is not 0, call back playback data progress via callback function cbDownLoadPos, see callback function fDownLoadPosCallBack for details.</p> <p>dwPosUser</p> <p><i>[in]</i> user data</p> <p>SDK send the data to user for further use via callback function fDownLoadPosCallBack</p> <p>fDownLoadDataCallBack</p> <p><i>[in]</i> record data callback function</p> <p>If fDownLoadDataCallBack is 0, not call back playback data.</p> <p>If fDownLoadDataCallBack is not 0, call back playback data progress via callback function fDownLoadDataCallBack, see callback function fDataCallBack for details.</p> <p>dwDataUser</p> <p><i>[in]</i> user data</p> <p>SDK send the data to user for further use via callback function fDownLoadDataCallBack</p> <p>pReserved</p> <p><i>[in]</i> reserved parameter</p> <p>Used for further extension, invalid until now, default value is NULL.</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	// at least one of the two formal parameters sSavedFileName and

Option	Instruction
	<p>fDownloadDataCallBack is valid.</p> <pre>g_IDownloadHandle = CLIENT_DownloadByRecordFileEx(gILoginHandle, &amp;stuNetFileInfo, "test.dav", DownloadPosCallBack, NULL, DataCallBack, NULL);</pre> <pre>if (g_IDownloadHandle == 0) {     printf("CLIENT_DownloadByRecordFileEx: failed!     Error code: %x.\n", CLIENT_GetLastError()); }</pre>
Note	<p>sSavedFileName is not NULL, record data is written to the corresponding file.</p> <p>fDownloadDataCallBack is not NULL, record data is returned via this callback function.</p> <p>After download, call CLIENT_StopDownload to close download handle.</p>

## CLIENT\_ParseData

Option	Instruction
Interface description	Interface for analyzing the obtained config info
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre>BOOL CLIENT_ParseData(     char* szCommand,     char* szInBuffer,     LPVOID lpOutBuffer,     DWORD dwOutBufferSize,     void* pReserved );</pre>
Parameter	<p>szCommand</p> <p><i>[in]</i> command parameter</p> <p>See "note" below for details.</p>

Option	Instruction
	<p>szInBuffer</p> <p><i>[in]</i> input buffer</p> <p>To be analyzed json string</p> <p>lpOutBuffer</p> <p><i>[out]</i> output buffer</p> <p>Different command parameter correspond to different struct type.</p> <p>dwOutBufferSize</p> <p><i>[in]</i> output buffer size</p> <p>pReserved</p> <p><i>[in]</i> reserved parameter</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> CFG_PTZ_PROTOCOL_CAPS_INFO          stuPtzCapsInfo          = {sizeof(stuPtzCapsInfo)}; if (FALSE == CLIENT_ParseData(CFG_CAP_CMD_PTZ,  pBuffer, &amp;stuPtzCapsInfo, sizeof(stuPtzCapsInfo), NULL)) {     printf("CLIENT_ParseData Failed, cmd[CFG_CAP_CMD_PTZ],     Last Error[%x]\n" , CLIENT_GetLastError()); } </pre>
Note	<p>Command parameter:</p> <pre> #define CFG_CAP_CMD_PTZ "ptz.getCurrentProtocolCaps" // obtain PTZ capacity set (CFG_PTZ_PROTOCOL_CAPS_INFO) #define CFG_CMD_ENCODE    "Encode"           // image channel property configure (corresponds to CFG_ENCODE_INFO) </pre> <p>See header file "dhconfigsdk.h" for more command parameters.</p>

## CLIENT\_DHPTZControlEx2

Option	Instruction
Interface	Extensive interface for private PTZ control. Support 3-D quick positioning and



Option	Instruction
description	fish-eye.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> BOOL CLIENT_DHPTZControlEx2(     LLONG ILoginID,     int nChannelID,     DWORD dwPTZCommand,     LONG IParam1,     LONG IParam2,     LONG IParam3,     BOOL dwStop ,     void* param4 = NULL ); </pre>
Parameter	<p>ILoginID  <i>[in]</i> logged device ID  Return value of login interface CLIENT_LoginEx2.</p> <p>nChannelId  <i>[in]</i> channel NO., start from 0.</p> <p>dwPTZCommand  <i>[in]</i> dome control command  See enum <a href="#">DH_PTZ_ControlType</a> and enum DH_EXTPTZ_ControlType for details.</p> <p>IParam1  <i>[in]</i> ancillary parameter 1  Used matched with other parameter, different control command has different IParam1.</p> <p>IParam2  <i>[in]</i> ancillary parameter 2  Used matched with other parameter, different control command has different IParam2</p>

Option	Instruction
	<p>IParam3</p> <p><i>[in]</i> ancillary parameter3</p> <p>Used matched with other parameter, different control command has different IParam3</p> <p>dwStop</p> <p><i>[in]</i> stop or not</p> <p>Valid when operating PTZ eight direction and lens, otherwise in when operating others the parameter is set to FALSE.</p> <p>IParam4</p> <p><i>[in]</i> ancillary parameter4, NULL as default.</p> <p>Used matched with other parameter, different control command has different IParam4</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> if (!CLIENT_DHPTZControlEx2(g_ILoginHandle, nChannelId, DH_PTZ_UP_CONTROL, 0, 0, 0, FALSE, NULL)) {     printf("CLIENT_DHPTZControlEx2 Failed, nChoose[DH_PTZ_UP_CONTROL]!Last Error[%x]\n" , CLIENT_GetLastError()); } </pre>
Note	More Information of Param ,please refer to the CLIENT_DHPTZControlEx2 interface in 'Network SDK Development Manual.chm' .

## CLIENT\_QueryNewSystemInfo

Option	Description
Interface description	Interface for obtaining new system capacity set (take the form of Json, see config SDK for details )
Precondition	Have called login interface CLIENT_LoginEx2.

Option	Description
Function	<pre> BOOL CLIENT_QueryNewSystemInfo(     LLONG ILoginID,     char* szCommand,     int nChannelID,     char* szOutBuffer,     DWORD dwOutBufferSize,     int *error,     int waittime=1000 ); </pre>
Parameter	<p><b>ILoginID</b>  <i>[in]</i> logged device ID  Return value of login interface CLIENT_LoginEx2.</p> <p><b>szCommand</b>  <i>[in]</i> search command  See Note for details</p> <p><b>nChannelID</b>  <i>[in]</i> search channel  Channel NO. is start from 0.  set to -1 means searching all the channels, partial commands not support setting to -1.</p> <p><b>szOutBuffer</b>  <i>[in]</i> buffer zone used to store data  Used to store the queried json string data</p> <p><b>dwOutBufferSize</b>  <i>[in]</i> buffer size</p> <p><b>error</b>  <i>[out]</i> returned error code  When failed obtaining, netsdk will write corresponding error code to this pointer address.</p>

Option	Description
	<p>waittime</p> <p>[in] timeout time</p> <p>Timeout time of waiting command returned, take 1000ms as default</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> char* pBuffer = new char[2048]; if (NULL == pBuffer) {     return; }  int nError = 0; if (FALSE == CLIENT_QueryNewSystemInfo(g_ILLoginHandle, CFG_CAP_CMD_PTZ, 0, pBuffer, 2048, &amp;nError)) {     printf("CLIENT_QueryNewSystemInfo Failed, cmd[CFG_CAP_CMD_PTZ], Last Error[%x]\n" , CLIENT_GetLastError());     if (pBuffer)     {         delete [] pBuffer;         pBuffer = NULL;     }     return; } </pre>
Note	<p>The json string obtained by this interface must be analyzed by CLIENT_ParseData interface,otherwise can not be used.</p> <p>The capacity set corresponds to CLIENT_QueryNewSystemInfo are as follows:</p> <pre> #define CFG_CAP_CMD_PTZ    "ptz.getCurrentProtocolCaps"    // </pre>

Option	Description
	obtaining PTZ capacity set (CFG_PTZ_PROTOCOL_CAPS_INFO) See header file “dhconfigsdk.h” for more command parameters.

## CLIENT\_SetDeviceMode

Option	Instruction
Interface description	Interface for setting work mode such as audio intercom, playback, right and etc.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> BOOL CLIENT_SetDeviceMode(     LLONG ILoginID,     <a href="#">EM_USEDEV_MODE</a> emType,     void* pValue ); </pre>
Parameter	<p>ILoginID [in] logged device ID</p> <p>Return value of login interface CLIENT_LoginEx2.</p> <p>emType [in] working mode type</p> <p>See enum <a href="#">EM_USEDEV_MODE</a> for details</p> <p>pValue [in] extensive parameter</p> <p>Different emType corresponds to different pValue,see enum EM_USEDEV_MODE for details.</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> //set playback stream type int nStreamType = 0; // 0-main/sub stream,1-main stream,2-sub stream if(!CLIENT_SetDeviceMode(gILoginHandle,     DH_RECORD_STREAM_TYPE, &amp;nStreamType)) { </pre>

Option	Instruction
	<pre>printf("CLIENT_ SetDeviceMode: failed! Error code: 0x%x.\n", CLIENT_GetLastError()); }</pre>
Note	None

## CLIENT\_StartSearchDevices

Option	Instruction
Interface description	Interface for async searching for IPC, NVS and etc. within same segment
Precondition	Have called CLIENT_Init interface
Function	<pre>LLONG CLIENT_StartSearchDevices( <a href="#">fSearchDevicesCB</a> cbSearchDevices, void* pUserData, char* szLocalIp=NULL );</pre>
Parameter	<p>cbSearchDevices  <i>[in]</i> device search callback function</p> <p>When there are response package returned from device-end, NetSDK will analyze them to valid information and inform user via callback function, see callback function fSearchDevicesCB for details.</p> <p>pUserData  <i>[in]</i> user data</p> <p>NetSDK will return the value to user for further user via callback function fSearchDevicesCB.</p> <p>szLocalIp  <i>[in]</i> local IP</p> <p>NULL as default if this parameter is not set.</p>
Return value	Succeed return handle, failed return FALSE.
Example	// start async searching for same-segment device

Option	Instruction
	<pre> g_IsearchHandle = CLIENT_StartSearchDevices(SearchDevicesCB, &amp;g_IDeviceList);  if (NULL == g_IsearchHandle) {     printf("CLIENT_StartSearchDevices Failed!Last Error[%x]\n",         CLIENT_GetLastError());     return; } </pre>
Note	This interface is for searching device in the same segment, search cross-segment device interface is CLIENT_SearchDevicesByIPs.

## CLIENT\_QueryDevState

Option	Instruction
Interface description	Interface for searching device status
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> BOOL CLIENT_QueryDevState(     LLONG ILoginID,     int nType,     char *pBuf,     int nBufLen,     int *pRetLen,     int waittime=1000 ); </pre>
Parameter	<p>ILoginID  <i>[in]</i> logged device ID</p> <p>Return value of login interface CLIENT_LoginEx2.</p> <p>nType  <i>[in]</i> search type</p>

Option	Instruction
	<p>See “note” below for details.</p> <p>pBuf  <i>[out]</i> output buffer zone</p> <p>Used to store the queried device infor, used matched with nType, see “note” below for details.</p> <p>nBufLen  <i>[in]</i> buffer zone size</p> <p>pRetLen  <i>[out]</i> lenth of queried data(unit : BYTE)</p> <p>waittime  <i>[in]</i> searching waiting time , 1000ms as default</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre>//Get audio intercom encode type supported by front-end device DHDEV_TALKFORMAT_LIST stulstTalkEncode; int retlen = 0; bSuccess = CLIENT_QueryDevState(g_ILoginHandle, DH_DEVSTATE_TALK_ECTYPE, (char*)&amp;stulstTalkEncode, sizeof(stulstTalkEncode), &amp;retlen, 3000); if (!(bSuccess &amp;&amp; retlen == sizeof(stulstTalkEncode))) {     printf("CLIENT_QueryDevState cmd[%d] Failed!Last Error[%x]\n",         DH_DEVSTATE_TALK_ECTYPE, CLIENT_GetLastError()); }</pre>
Note	<p>Supported searching type is shown in below:</p> <pre>#define DH_DEVSTATE_TALK_ECTYPE      0x0009</pre> <p>//searching audiointercom type list device supported, see struct DHDEV_TALKFORMAT_LIST for details.</p> <p>See dhnetSDK.h header file for more command parameter.</p>



## CLIENT\_StartTalkEx

Option	Instruction
Interface description	Extensive interface for opening audio intercom
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> LLONG CLIENT_StartTalkEx(     LLONG ILoginID,     <a href="#">pfAudioDataCallBack</a> pfcbb,     LDWORD dwUser ); </pre>
Parameter	<p>ILoginID  <i>[in]</i> logged device ID  Return value of login interface CLIENT_LoginEx2.</p> <p>pfcbb  <i>[in]</i> audio intercom audio data callback function  See callback function <a href="#">pfAudioDataCallBack</a> for details.</p> <p>dwUser  <i>[in]</i> user data  SDK will call back the data to user for further use via callback function <a href="#">pfAudioDataCallBack</a></p>
Return value	Succeed return audio intercom handle, failed return FALSE.
Example	<pre> g_ITalkHandle = CLIENT_StartTalkEx(gILoginHandle, AudioDataCallBack, (DWORD)NULL); if(0 == g_ITalkHandle) {     printf("CLIENT_StartTalkEx          Failed!Last          Error[%x]\n",         CLIENT_GetLastError()); } </pre>
Note	None

## CLIENT\_StopTalkEx

Option	Instruction
Interface description	Extensive interface for stopping audio intercom
Precondition	Have called open audio intercom interface as CLIENT_StartTalkEx etc.
Function	<pre> BOOL CLIENT_StopTalkEx(     LLONG ITalkHandle ); </pre>
Parameter	<p>ITalkHandle</p> <p><i>[in]</i> audio intercom handle</p> <p>Corresponding to return value of interface as CLIENT_StartTalkEx etc.</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> if(!CLIENT_StopTalkEx(g_ITalkHandle)) {     printf("CLIENT_StopTalkEx          Failed!Last          Error[%x]\n",         CLIENT_GetLastError()); } else {     g_ITalkHandle = 0; } </pre>
Note	None

## CLIENT\_RecordStartEx

Option	Instruction
Interface description	Extensive interface for starting PC-end recording(extension of CLIENT_RecordStart())
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> BOOL CLIENT_RecordStartEx( </pre>

Option	Instruction
	<pre>         LLONG ILoginID     ); </pre>
Parameter	<pre> ILoginID [in] logged device ID </pre> <p>Return value of login interface CLIENT_LoginEx2.</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> BOOL bSuccess = CLIENT_RecordStartEx(g_ILoginHandle); if(!bSuccess) {     printf("CLIENT_RecordStartEx Failed!     Last Error[%x]\n", CLIENT_GetLastError()); } </pre>
Note	None

## CLIENT\_RecordStopEx

Option	Instruction
Interface description	Extensive interface for stopping PC-end recording(extension of CLIENT_RecordStop())
Precondition	Have called open local recording interface as CLIENT_RecordStartEx etc.
Function	<pre> BOOL CLIENT_RecordStopEx(     LLONG ILoginID ); </pre>
Parameter	<pre> ILoginID [in] logged device ID </pre> <p>Return value of login interface CLIENT_LoginEx2.</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre> // stop local recording if (g_RecordFlag) </pre>

Option	Instruction
	<pre> {     if (!CLIENT_RecordStopEx(g_ILLoginHandle))     {         printf("CLIENT_RecordStop Failed!         Last Error[%x]\n", CLIENT_GetLastError());     }     else     {         g_RecordFlag = FALSE;     } } </pre>
Note	CLIENT_RecordStopEx interface needs to be used corresponding to CLIENT_RecordStartEx

## CLIENT\_TalkSendData

Option	Instruction
Interface description	Interface for sending audio data to device
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> LONG CLIENT_TalkSendData(     LLONG ITalkHandle,     char *pSendBuf,     DWORD dwBufSize ); </pre>
Parameter	<p>ITalkHandle  <i>[in]</i> audio intercom handle</p> <p>Corresponding to return value of interface as CLIENT_StartTalkEx etc.</p> <p>pSendBuf  <i>[in]</i> send buffer zone</p>

Option	Instruction
	<p>Store data to be sent</p> <p>dwBufSize</p> <p><i>[in]</i>buffer size</p> <p>Length of data to be sent(unit: BYTE)</p>
Return value	Succeed return data length has been transferred practically, failed return -1.
Example	<pre> LONG   ISendLen   =   CLIENT_TalkSendData(ITalkHandle,   pDataBuf, dwBufSize); if(ISendLen != (long)dwBufSize) {     printf("CLIENT_TalkSendData Failed!Last Error[%x]\n" ,         CLIENT_GetLastError()); } </pre>
Note	<p>Send the data which is called back by callback function set in</p> <p><a href="#">CLIENT_StartTalkEx</a> to device</p>

## CLIENT\_AudioDecEx

Option	Instruction
Interface description	Extensive interface for decoding audio data
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> BOOL CLIENT_AudioDecEx(     LLONG ITalkHandle,     char *pAudioDataBuf,     DWORD dwBufSize ); </pre>
Parameter	<p>ITalkHandle</p> <p><i>[in]</i> audio intercom handle</p> <p>Corresponding to return value of interface as CLIENT_StartTalkEx etc.</p> <p>pAudioDataBuf</p>

Option	Instruction
	<p><i>[in]</i> audio buffer zone</p> <p>The audio data to be decoded.</p> <p>dwBufSize</p> <p><i>[in]</i> buffer size</p> <p>Length of audio data to be decoded (unit : BYTE)</p>
Return value	Succeed return TRUE, failed return FALSE.
Example	<pre>// send the audio data which received from device-end to SDK to decode and play. if (!CLIENT_AudioDecEx(ITalkHandle, pDataBuf, dwBufSize)) {     printf("CLIENT_AudioDecEx    Failed!Last    Error[%x]\n"    ,         CLIENT_GetLastError()); }</pre>
Note	Decode the data sent by device in audio intercom

## CLIENT\_SetDVRMessCallBack

Option	Instruction
Interface description	Interface for setting alarm callback function.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre>void CLIENT_SetDVRMessCallBack(     <a href="#">fMessCallBack</a> cbMessage,     LDWORD dwUser );</pre>
Parameter	<p>cbMessage</p> <p><i>[in]</i> alarm callback function.</p> <p>Please refer to fMessCallBack for details.</p> <p>dwUser</p> <p><i>[in]</i> user data, SDK sends the data to user for further use via real-time</p>

Option	Instruction
	monitoring data callback function fMessCallBack.
Return value	None
Example	//Set alarm event callback function. CLIENT_SetDVRMessCallBack(MessCallBack , NULL);
Note	Call CLIENT_SetDVRMessCallBack before alarm subscription. The events received by the callback function set don't contain picture event.

## CLIENT\_StartListenEx

Option	instruction
Interface description	Extensive interface for subscribing alarm event from device
Precondition	Have called login interface CLIENT_LoginEx2.
Function	BOOL CLIENT_StartListenEx( LLONG ILoginID );
Parameter	ILoginID [in] logged device ID Return value of login interface CLIENT_LoginEx2.
Return value	Succeeded return TRUE, failed return FALSE.
Example	// subscribe alarm event from device if( CLIENT_StartListenEx(g_ILoginHandle)) { g_bStartListenFlag = TRUE; printf("Listen Success!\nJust Wait Event....\n"); } else { printf("CLIENT_StartListenEx Failed! Last

Option	instruction
	<pre>Error[%x]\n" , CLIENT_GetLastError()); }</pre>
Note	All the device alarm events are sent to user via callback function set in interface <a href="#">CLIENT_SetDVRMessCallBack</a> .

## CLIENT\_StopListen

Option	Instruction
Interface description	Interface for stopping subscribing alarm
Precondition	Have called alarm report subscription interface as CLIENT_StartListenEx etc.
Function	<pre>BOOL CLIENT_StopListen(     LLONG ILoginID );</pre>
Parameter	<p>ILoginID</p> <p><i>[in]</i> logged device ID</p> <p>Return value of login interface CLIENT_LoginEx2.</p>
Return value	Succeeded return TRUE, failed return FALSE.
Example	<pre>// stop subscribing alarm from device if (g_bStartListenFlag) {     if (!CLIENT_StopListen(gILoginHandle))     {         printf("CLIENT_StopListen Failed!Last             Error[%x]\n", CLIENT_GetLastError());     }     else     {         g_bStartListenFlag = FALSE;     } }</pre>



Option	Instruction
	}
Note	None

## CLIENT\_StopSearchDevices

Option	Instruction
Interface description	Interface for stopping async search for IPC, NVS and etc. within same segment.
Precondition	Have called async search interface as CLIENT_StartSearchDevices etc.
Function	<pre> BOOL CLIENT_StopSearchDevices(     LLONG ISearchHandle ); </pre>
Parameter	<p>ISearchHandle</p> <p><i>[in]</i> async search for device ID</p> <p>Corresponding to return value of async search for device interface as CLIENT_StartSearchDevices etc.</p>
Return value	Succeeded return TRUE, failed return FALSE.
Example	<pre> // stop async search for device within same segment if (NULL != g_ISearchHandle) {     if (FALSE == CLIENT_StopSearchDevices(g_ISearchHandle))     {         printf("CLIENT_StopSearchDevices Failed!Last Error[%x]\n",             CLIENT_GetLastError());     } } </pre>
Note	This interface is used with <a href="#">CLIENT_StartSearchDevices</a> as a pair.

## CLIENT\_SearchDevicesByIPs

Option	Instruction
Interface description	Interface for sync searching for cross-segment device
Precondition	Have called CLIENT_Init interface.
Function	<pre> BOOL CLIENT_SearchDevicesByIPs(     DEVICE_IP_SEARCH_INFO* plpSearchInfo,     <a href="#">fSearchDevicesCB</a> cbSearchDevices,     LDWORD dwUserData,     char* szLocalIp,     DWORD dwWaitTime ); </pre>
Parameter	<p>plpSearchInfo  <i>[in]</i> information of device to be searched for.  IP of device to be searched for. See header file “dhnetsdk.h” for more information about “DEVICE_IP_SEARCH_INFO”.</p> <p>cbSearchDevices  <i>[in]</i> searching for device callback function  When there are response package returned from device-end, SDK will analyze them to valid information and inform user via callback function, see callback function fSearchDevicesCB for details.</p> <p>dwUserData  <i>[in]</i> user data  SDK send the data to user for further use via callback function fSearchDevicesCB</p> <p>szLocalIp  <i>[in]</i> local IP  NULL as default if this parameter is not set.</p> <p>dwWaitTime</p>

Option	Instruction
	<p><i>[in]</i> time for searching</p> <p>User can set the parameter as needed, as this is a sync interface, only when the time set here is out, can the interface have a return .</p>
Return value	Succeeded return TRUE, failed return FALSE.
Example	<pre> DWORD dwWaitTime = 5000;  // please note that only when the time set here is out, can the interface have a return.user can decide the timeout time according to net condition. if (FALSE == CLIENT_SearchDevicesByIPs(&amp;stuTmp, SearchDevicesCB, (LDWORD)&amp;g_IDeviceList, szLocalIp, dwWaitTime)) {     printf("CLIENT_SearchDevicesByIPs      Failed!Last      Error[%x]\n", CLIENT_GetLastError());     return; } </pre>
Note	As this is a sync interface, only when the time set here is out, can the interface have a return . user can decide the timeout time according to net condition.

## CLIENT\_RealLoadPictureEx

Option	Instruction
Interface description	Interface for intelligent picture alarm subscription.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre> LLONG CLIENT_RealLoadPictureEx(     LLONG ILoginID,     int nChannelID,     DWORD dwAlarmType,     BOOL bNeedPicFile, <a href="#">fAnalyzerDataCallBack</a> cbAnalyzerData,     LDWORD dwUser, </pre>

Option	Instruction
	<p>void* Reserved</p> <p>);</p>
Parameter	<p>lLoginID</p> <p><i>[in]</i> device login ID.</p> <p>Correspond with return value of login interface CLIENT_LoginEx2.</p> <p>nChannelID</p> <p><i>[in]</i>channel number of intelligent picture alarm subscription, start from 0.</p> <p>dwAlarmType</p> <p><i>[in]</i> Alarm type of expected subscription .</p> <p>Example: EVENT_IVS_ALL // subscription all event</p> <p>See header file “dhnetsdk.h” for more command parameters.</p> <p>bNeedPicFile</p> <p><i>[in]</i> whether subscribe picture file.</p> <p>TRUE:means subscribe picture file,return intelligent picture info in the callback function.</p> <p>FALSE:means not subscribe picture file,not return intelligent picture info in the callback function(reduce network flux when do not need to picture info).</p> <p>cbAnalyzerData</p> <p><i>[in]</i> intelligent picture alarm callback function.</p> <p>When device has intelligent picture alarm to report,SDK will call this function to recall data for users.</p> <p>dwUser</p> <p><i>[in]</i> user data, SDK sends the data to user for further use via intelligent picture alarm callback function fAnalyzerDataCallBack.</p> <p>Reserved</p> <p><i>[in]</i> reserved parameter.</p> <p>Recommended to be set to NULL.</p>
Return value	Failed return 0,otherwise return ID of intelligent picture alarm subscription to

Option	Instruction
	be used as parameters of relatited function
Example	<pre>//subscribe intelligent picture alarm. LDWORD dwUser = 0; g_IRealLoadHandle = CLIENT_RealLoadPictureEx(gILoginHandle, 0, EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, dwUser, NULL); if (0 == g_IRealLoadHandle) {     printf("CLIENT_RealLoadPictureEx Failed! Last Error[%x]\n",     CLIENT_GetLastError());     return; }</pre>
Note	<p>Each subscription corresponds to one channel, and corresponds to a type of event.</p> <p>If a user wants to set all types of event for one channel, the parameter dwAlarmType should be set to EVENT_IVS_ALL.</p> <p>If you want to set two types of events for one channel, please call CLIENT_RealLoadPictureEx twice and set different event type.</p> <p>You can cancel subscription by CLIENT_StopLoadPic.</p>

## CLIENT\_ControlDeviceEx

Option	Instruction
Interface description	Extensive interface for device control.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre>BOOL CLIENT_ControlDeviceEx(     LLONG ILoginID,     CtrlType emType,     void* pInBuf,     void* pOutBuf = NULL,</pre>

Option	Instruction
	<pre>int nWaitTime = 1000</pre> <pre>);</pre>
Parameter	<p><b>ILoginID</b></p> <p><i>[in]</i> device login ID.</p> <p>Correspond with return value of login interface CLIENT_LoginEx2.</p> <p><b>emType</b></p> <p><i>[in]</i>control type.</p> <p>Work in with pInBuf and pOutBuf, different emType,pInBuf and pOutBuf point to different structures,please refer to enum CtrlType for details.</p> <p><b>pInBuf</b></p> <p><i>[in]</i> input parameter of device control.</p> <p>Work in with emType, different emType and pInBuf point to different structure,please refer to enum CtrlType for details. If the value of emType did not indicate what struct pInBuf is in enum instruction, set the parameter to NULL.</p> <p><b>pOutBuf</b></p> <p><i>[in]</i> output parameter of device control,the default is NULL.</p> <p>Work in with emType, different emType and pOutBuf point to different structures,please refer to enum CtrlType for details. If the value of emType did not indicate what struct pOutBuf is in enum instruction, set the parameter to NULL.</p> <p><b>nWaitTime</b></p> <p><i>[in]</i> timeout time of waiting for device returns,unit is ms.</p> <p>The default value is 1000ms.</p>
Return value	Succeeded return TRUE,otherwise return FALSE.
Example	<pre>MANUAL_SNAP_PARAMETER stuSanpParam = {0};</pre> <pre>stuSanpParam.nChannel = 0;</pre> <pre>memcpy(stuSanpParam.bySequence, "just for test",</pre> <pre>sizeof(stuSanpParam.bySequence) - 1);</pre>

Option	Instruction
	<pre>// manually snapshot trigger alarm function, this function is only valid for ITC device  if (FALSE == CLIENT_ControlDeviceEx(g_LoginHandle, DH_MANUAL_SNAP, &amp;stuSanpParam)) {     printf("CLIENT_ControlDeviceEx Failed!Last Error[%x]\n", CLIENT_GetLastError());     break; }</pre>
Note	None

## CLIENT\_StopLoadPic

Option	Instruction
Interface description	Interface for stopping subscribing intelligent picture alarm.
Precondition	Have called intelligent picture alarm subscription interface as CLIENT_RealLoadPictureEx etc.
Function	<pre>BOOL CLIENT_StopLoadPic(     LLONG IAnalyzerHandle );</pre>
Parameter	<p>IAnalyzerHandle</p> <p><i>[in]</i> intelligent picture alarm subscription ID.</p> <p>Correspond with return value of intelligent picture alarm subscription interface as CLIENT_RealLoadPictureEx.</p>
Return value	Succeeded return TRUE,otherwise return FALSE.
Example	//stop subscribing picture alarm

Option	Instruction
	<pre> if (0 != g_IRealLoadHandle) {     if (FALSE == CLIENT_StopLoadPic(g_IRealLoadHandle))     {         printf("CLIENT_StopLoadPic Failed! Last Error[%x]\n",             CLIENT_GetLastError());     }     else     {         g_IRealLoadHandle = 0;     } } </pre>
Note	None.

## CLIENT\_GetDownloadPos

Option	Instruction
Interface description	Interface for searching record download process, unit is KB.
Precondition	Have called record download interface as CLIENT_DownloadByTimeEx etc.
Function	<pre> BOOL CLIENT_GetDownloadPos(     LLONG IFileHandle,     int *nTotalSize,     int *nDownloadSize ); </pre>
Parameter	<p>IFileHandle</p> <p><i>[in]</i> download handle.</p> <p>Correspond with return value of download record interface as CLIENT_DownloadByTimeEx.</p> <p>nTotalSize</p>



Option	Instruction
	<p><i>[out]</i> total size of download,unit is KB.</p> <p>nDownloadSize</p> <p><i>[out]</i> Downloaded Size,unit is KB.</p>
Return value	Succeeded return TRUE,otherwise return FALSE.
Example	<pre>int nTotal = 0; int nDownload = 0; if (FALSE == CLIENT_GetDownloadPos(g_IDownloadHandle, &amp;nTotal, &amp;nDownload)) {     printf("CLIENT_GetDownloadPos Failed!Last Error[%x]\n",         CLIENT_GetLastError()); }</pre>
Note	None

## CLIENT\_SetSnapRevCallback

Option	Instruction
Interface description	Interface for setting front-end snapshot callback function.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	<pre>void CLIENT_SetSnapRevCallBack(     fSnapRev OnSnapRevMessage,     LDWORD dwUser );</pre>
Parameter	<p>OnSnapRevMessage</p> <p><i>[in]</i> front-end snapshot callback function.</p> <p>Please refer to fSnapRev callback function.</p> <p>dwUser</p> <p><i>[in]</i> user data, SDK returns data to user by front-end snapshot callback function (fSnapRev) for further use.</p>

Option	Instruction
Return value	None
Example	// set front-end snapshot callback function. CLIENT_SetSnapRevCallBack(SnapRev, NULL);
Note	CLIENT_SetSnapRevCallBack interface needs to be called before front-end snapshot interface.

## CLIENT\_SnapPictureEx

Option	Instruction
Interface description	Extensive interface for snapshot request.
Precondition	Have called login interface CLIENT_LoginEx2.
Function	BOOL CLIENT_SnapPictureEx( LLONG ILoginID, <a href="#">SNAP_PARAMS</a> *par, int *reserved = 0 );
Parameter	ILoginID [in] device login ID. Correspond with return value(dwUser) of login interface CLIENT_LoginEx2. par [in]snapshot parameter. Please refer to structure <a href="#">SNAP_PARAMS</a> . reserved [in]reserved field.
Return value	Succeeded return TRUE,otherwise return FALSE.
Example	// send snapshot command to front-end device.  SNAP_PARAMS  stuSnapParams;

Option	Instruction
	<pre>stuSnapParams.Channel = nChannelId; stuSnapParams.mode = nSnapType; stuSnapParams.CmdSerial = ++g_nCmdSerial; //request SN,valid range: 0~65535,over this range will be cut as unsigned short. if (FALSE == CLIENT_SnapPictureEx(gILoginHandle, &amp;stuSnapParams)) {     printf("CLIENT_SnapPictureEx Failed!Last Error[%x]\n",         CLIENT_GetLastError());     return; } else {     printf("CLIENT_SnapPictureEx succ\n"); }</pre>
Note	None