SDK Programming Guide

User's Manual

V3.5.0.2

Foreword

Purpose

Welcome to use NetSDK (hereinafter referred to be "the SDK") programming guide (hereinafter referred to be "the guide").

SDK, also known as network device SDK, is a development kit for developer to develop the interfaces for network communication among surveillance products such as Network Video Recorder (NVR), Network Video Server (NVS), IP Camera (IPC), Speed Dome (SD), and intelligence devices.

The manual describes the interfaces, functions and calling relationships, and provides code examples.

The example codes provided in the guide are only for demonstrating the procedure and not assured to copy for use.

Readers

- SDK software development engineers
- Project managers
- Product managers

Safety Instructions

The following categorized signal words with defined meaning might appear in the manual.

Signal Words	Meaning
 DANGER	Indicates a high potential hazard which, if not avoided, will result in death or serious injury.
 WARNING	Indicates a medium or low potential hazard which, if not avoided, could result in slight or moderate injury.
 CAUTION	Indicates a potential risk which, if not avoided, could result in property damage, data loss, lower performance, or unpredictable result.
 TIPS	Provides methods to help you solve a problem or save you time.
 NOTE	Provides additional information as the emphasis and supplement to the text.

Revision History

Version	Revision Content	Release Time
V3.5.0	Modified the interfaces to optimize hard disk information retrieval and error code output.	February 2025
V3.4.11	Updated some descriptions	February 2022

Version	Revision Content	Release Time
V3.4.10	● Added reminders for NVR6 series device login. ● Moved structure, enumeration and interface function chapters to mainbody, and changed manual format. ● Deleted fisheye correction library.	May 2021
V3.4.9	Deleted function library avnetsdk.dll and libavnetsdk.so related content, and changed font.	March 2021
V3.4.8	Update to the latest version.	February 2020
V1.0.0	First release.	February 2018

About the Manual

- The manual is for reference only. If there is inconsistency between the manual and the actual product, the actual product shall prevail.
- We are not liable for any loss caused by the operations that do not comply with the manual.
- The manual would be updated according to the latest laws and regulations of related jurisdictions. For detailed information, refer to the paper manual, CD-ROM, QR code or our official website. If there is inconsistency between paper manual and the electronic version, the electronic version shall prevail.
- All the designs and software are subject to change without prior written notice. The product updates might cause some differences between the actual product and the manual. Please contact the customer service for the latest program and supplementary documentation.
- There still might be deviation in technical data, functions and operations description, or errors in print. If there is any doubt or dispute, we reserve the right of final explanation.
- Upgrade the reader software or try other mainstream reader software if the manual (in PDF format) cannot be opened.
- All trademarks, registered trademarks and the company names in the manual are the properties of their respective owners.
- Please visit our website, contact the supplier or customer service if there is any problem occurring when using the device.
- If there is any uncertainty or controversy, we reserve the right of final explanation.

Table of Contents

1 Overview

1.1.1 General

The manual introduces SDK interfaces reference information that includes main function modules, interface functions, and callback functions.

The following are the main functions:

SDK initialization, device login, real-time monitoring, record playback, download, PTZ control, voice talk, video snapshot, alarm upload, device search, smart event upload and snapshot, user management, device restart, decide upgrade, device timing, video parameter setting, channel name setting, and network parameter setting of device.

The development kit might be different dependent on the environment.

There are files included in development.

Table 1-1 Files included in Windows development kit

Library type	Library file name	Library file description
Function library	dhnetsdk.h	Header file
	dhnetsdk.lib	Lib file
	dhnetsdk.dll	Library file
	avnetsdk.dll	Library file
Configuration library	avglobal.h	Header file
	dhconfigsdk.h	Configuration Header file
	dhconfigsdk.lib	Lib file
	dhconfigsdk.dll	Library file
Auxiliary library of playing (coding and decoding)	dhplay.dll	Playing library
Auxiliary library of "dhnetsdk.dll"	IvsDrawer.dll	Image display library
	StreamConvertor.dll	Transcoding library

Table 1-2 files included in Linux development kit

Library type	Library file name	Library file description
Function library	dhnetsdk.h	Header file
	libdhnetsdk.so	Library file
	libavnetsdk.so	Library file
Configuration library	avglobal.h	Header file
	dhconfigsdk.h	Configuration Header file
	libdhconfigsdk.so	Configuration library
Auxiliary library of "libdhnetsdk.so"	libStreamConvertor.so	Transcoding library

- The function library and configuration library are necessary libraries.
- The function library is the main body of SDK, which is used for communication interaction between client and products, remotely controls device, queries device data, configures device data information, as well as gets and handles the streams.

- The configuration library packs and parses the structures of configuration functions.
- It is recommended to use auxiliary library of playing (coding and decoding) to parse and play the streams.
- The auxiliary library decodes the audio and video streams for the functions such as monitoring and voice talk, and collects the local audio.

1.2 Applicability

- Recommended memory: No less than 512 M
- System supported by SDK:
  - ◇ Windows
    Windows 10, Windows 8, Windows 7, and Windows Server 2008/2003
  - ◇ Linux
    The common Linux systems such as Red Hat and SUSE

Table 1-3 The device suitable for functions

Function	**Supported device**
Device login	DVR, NVR, IPC and SD
Real-time monitoring	DVR, NVR, IPC and SD
Record playback	Storage devices, such as DVR and NVR
Download	Storage devices, such as DVR and NVR
PTZ control	SD
Voice talk	DVR, NVR, IPC and SD
Video snapshot	DVR, NVR, IPC and SD
Alarm upload	DVR, NVR, IPC and SD
Device search	DVR, NVR, IPC and SD
Smart event upload and snapshot	IVS, mobile and smart SD
ser management	DVR, NVR, IPC and SD

2 Overview

All the example codes are tested by VS2005sp1 under Windows OS.

2.1 SDK Initialization

2.1.1 Introduction

Initialization is the first step of SDK to conduct all the function modules. It does not have the surveillance function but can set some parameters that affect the SDK overall functions.

- Initialization occupies some memory.
- Only the first initialization is valid within one process.
- After using this function, call cleanup interface to release SDK resource.

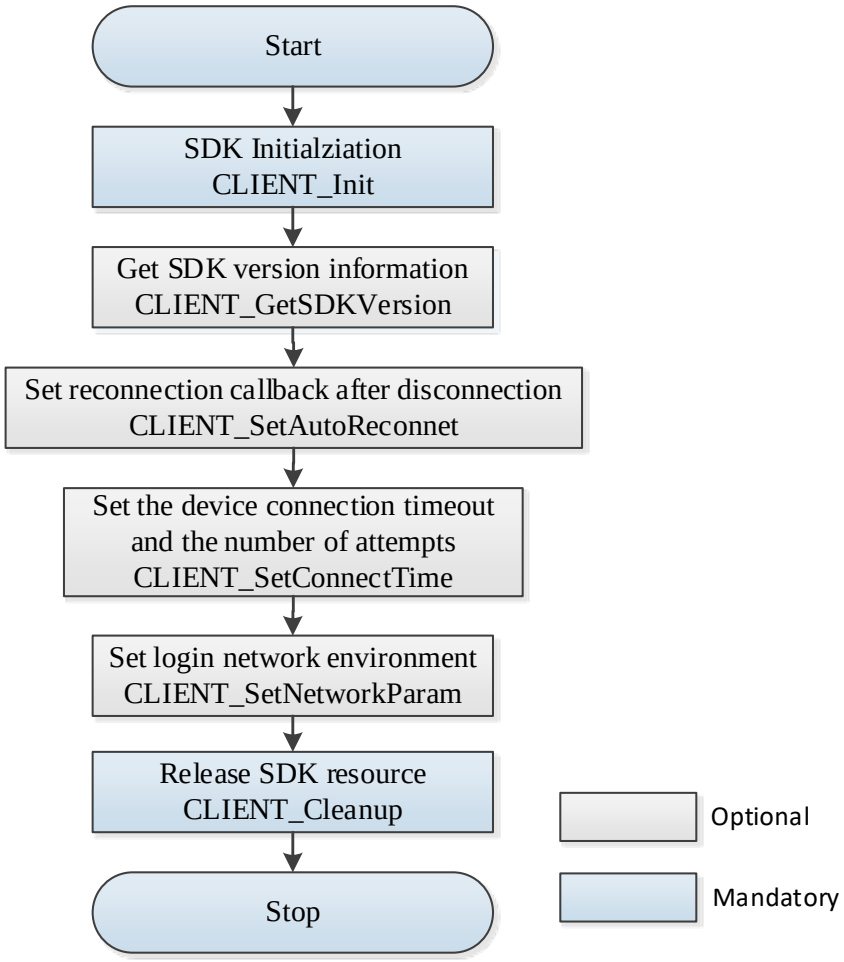
2.1.2 Interface Overview

Table 2-1 Interfaces of SDK initialization

Interface	Implication
CLIENT_Init	SDK initialization
CLIENT_Cleanup	SDK cleaning up
CLIENT_GetSDKVersion	Get SDK version information
CLIENT_GetLastError	Get error codes of other interfaces whichare fail to call
CLIENT_SetAutoReconnect	Set reconnection callback after disconnection
CLIENT_SetConnectTime	Set the device connection timeout and the number of attempts
CLIENT_SetNetworkParam	Set login network environment

2.1.3 Process

Figure 2-1 Process of SDK initialization



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 (Optional) Call **CLIENT_GetSDKVersion** to get SDK version information.

Step 3 (Optional but suggested) Call CLIENT_SetAutoReconnect **to set reconnection callback. Internal SDK auto connects when the device disconnected.**

Step 4 (Optional) Call **CLIENT_SetConnectTime** to set device connection timeout and trial times.

Step 5 (Optional) Call **CLIENT_SetNetworkParam** to set network login parameters, including device login timeout and trial times.

Step 6 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

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 is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LONG lLoginID, 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 by 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");
    }

    // Optional operation
    // Get the SDK version information
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]", dwNetSdkVersion);

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
```

```
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

    // When first time logging in, some data is needed to be initialized to enable normal business function. It
is recommended to wait for a while after login, and 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 operation
```

```
    // Clean up initialization resources
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}
int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}
//**********************************************************************************
// Commonly used callback function 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("lLoginID[0x%x]", lLoginID);
    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 Introduction

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 completing, users 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.

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

- The provided login operation is for Dahua devices only, not for other manufactures' devices. Do the login operation carefully; otherwise the device will not be able to login successfully.
- 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.
- The login of NVR6 series devices (supports 16 and more HDD) can take long due to the large number of HDD. To avoid that, we recommend using CLIENT_SetOptimizeMode interface to obtain HDD information before device login. After above configuration, the returned parameter of HDD number when logging in the interface becomes invalid. You can obtain through CLIENT_QueryDevState (DH_DEVSTATE_DISK) interface. Example code of optimizing obtaining HDD information is shown below:

```
int opt = OPTTYPE_MOBILE_DISK_INFO;
CLIENT_SetOptimizeMode(EM_OPT_TYPE_MOBILE_OPTION, &opt);
```

2.2.2 Interface Overview

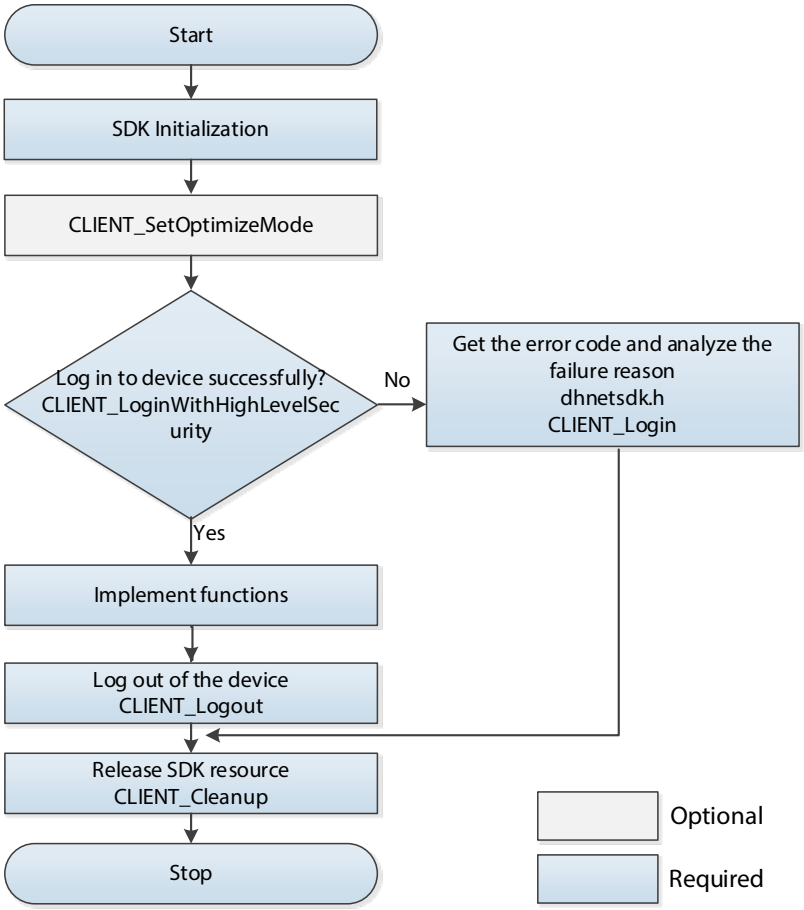
Table 2-2 Interfaces of device login

Interface	Implication
CLIENT_Init	SDK initialization
CLIENT_Cleanup	SDK cleaning up
CLIENT_LoginWithHighLevelSecurity	Log in to the device with high level security. CLIENT_LoginEx2 can still be used,but there are security risks,so it is highly recommended to use the interface CLIENT_LoginWithHighLevelSecurity to log in to the device.
CLIENT_Logout	Logout
CLIENT_GetLastError	Get error codes of other interfaces which fail to be called.
CLIENT_SetOptimizeMode	Optimize obtaining hard disk information.

2.2.3 Process

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.

Figure 2-2 Process of sync login



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Optional. Call **CLIENT_SetOptimizeMode** to optimize obtaining hard disk information.

Step 3 After initialization,call **CLIENT_LoginWithHighLevelSecurity** to log in to device.

Step 4 After login, users can realize business as needed.

Step 5 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 6 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.2.4 Example Code

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

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

static LLONG g_lLoginHandle = 0L;
static char g_szDevIp[32] = "172.32.4.25";
static WORD g_nPort = 37777; // Tcp connection port, which should be the same as tcp connection port of expected login device.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";
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 is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, 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 by 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 information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);
//This operation is optional. It is to optimize obtaining hard disk information.
    int opt = OPTTYPE_MOBILE_DISK_INFO;
CLIENT_SetOptimizeMode(EM_OPT_TYPE_MOBILE_OPTION, &opt);
```

```
    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);


    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);


        if(0 == g_lLoginHandle)
        {
          // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header file
 is decimal. Take care of conversion.
          // For example:
          // #define NET_NOT_SUPPORTED_EC(23)
          // Do not support this function. The corresponding error code is 0x80000017, and the
corresponding hexadecimal is 0x17.


          printf("CLIENT_LoginEx %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort ,
CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginEx %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business
function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
```

```
    // Task realizing operation
    ;
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // Log out of 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 initialization resources
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
{
    InitTest();
    RunTest();
    EndTest();
    return 0;
}
//**************************************************************************************
// Commonly used callback function 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 lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    printf("Call HaveReConnect\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");
}
```

2.3 Real-time Monitoring

2.3.1 Introduction

Real-time monitoring obtains the real-time stream from the storage device or front-end device, which is an important part of the surveillance system.

SDK can get the main stream and sub stream from the device once it logged.

- Support configuring bit stream resolution, encode, bit rate and other parameters of front-end devices.
- Support setting of image saturation, contrast, exposure and so on.
- Support conveying window handle from users, and SDK analyzes stream and play directly.
- Support calling back real-time stream data to users, and let users process by themselves.
- Support saving 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

Table 2-3 Interfaces of real-time monitoring

Interface	Implication
CLIENT_Init	SDK initialization
CLIENT_Cleanup	SDK cleaning up
CLIENT_LoginWithHighLevelSecurity	Log in to the device with high level security. CLIENT_LoginEx2 can still be used,but there are security risks,so it is highly recommended to use the interface CLIENT_LoginWithHighLevelSecurity to log in to the device.
CLIENT_RealPlay	Start real-time monitoring
CLIENT_StopRealPlay	Stop real-time monitoring
CLIENT_RealPlayEx	Extensive interface of starting real-time monitoring
CLIENT_StopRealPlayEx	Extensive interface of stopping real-time monitoring
CLIENT_StartRealPlay	Callback interface of starting real-time monitoring and supporting to set bit stream
CLIENT_SetRealDataCallBackEx	Extensive interface of setting real-time monitoring data callback
CLIENT_ClientGetVideoEffect	Get image attributes
CLIENT_ClientSetVideoEffect	Set image attributes
CLIENT_AdjustFluency	Adjust image playback fluency
CLIENT_Logout	Logout
CLIENT_GetLastError	Get error codes of other interfaces which fail to be called.

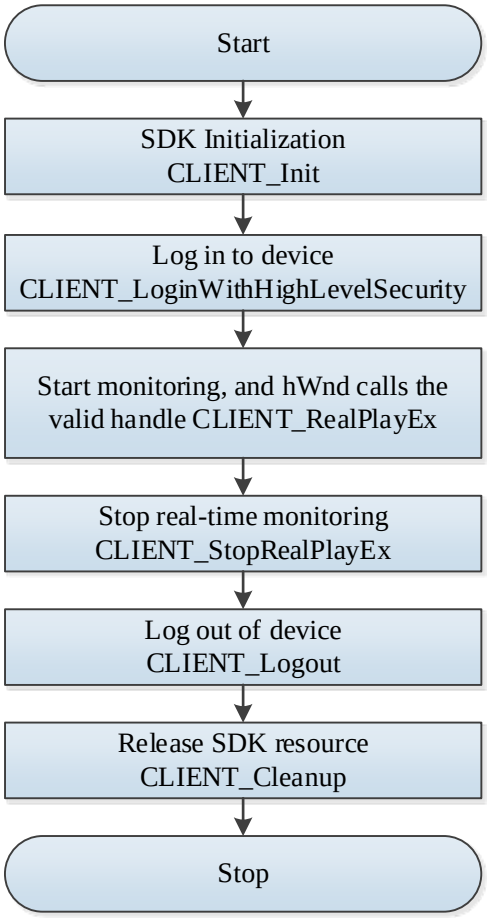
2.3.3 Process

There are two methods of real-time monitoring:

- SDK decoding play
  SDK realizes real-time play by calling playsdk library in aux library.
- The third party decoding play
  SDK only calls back real-time monitoring data stream to users, and then users decodes and plays with a third-party library.

2.3.3.1 SDK Decoding Play

Figure 2-3 Process of playing by SDK decoding library



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call **CLIENT_RealPlayEx** to enable the real-time monitoring. The parameter hWnd is a valid window handle.

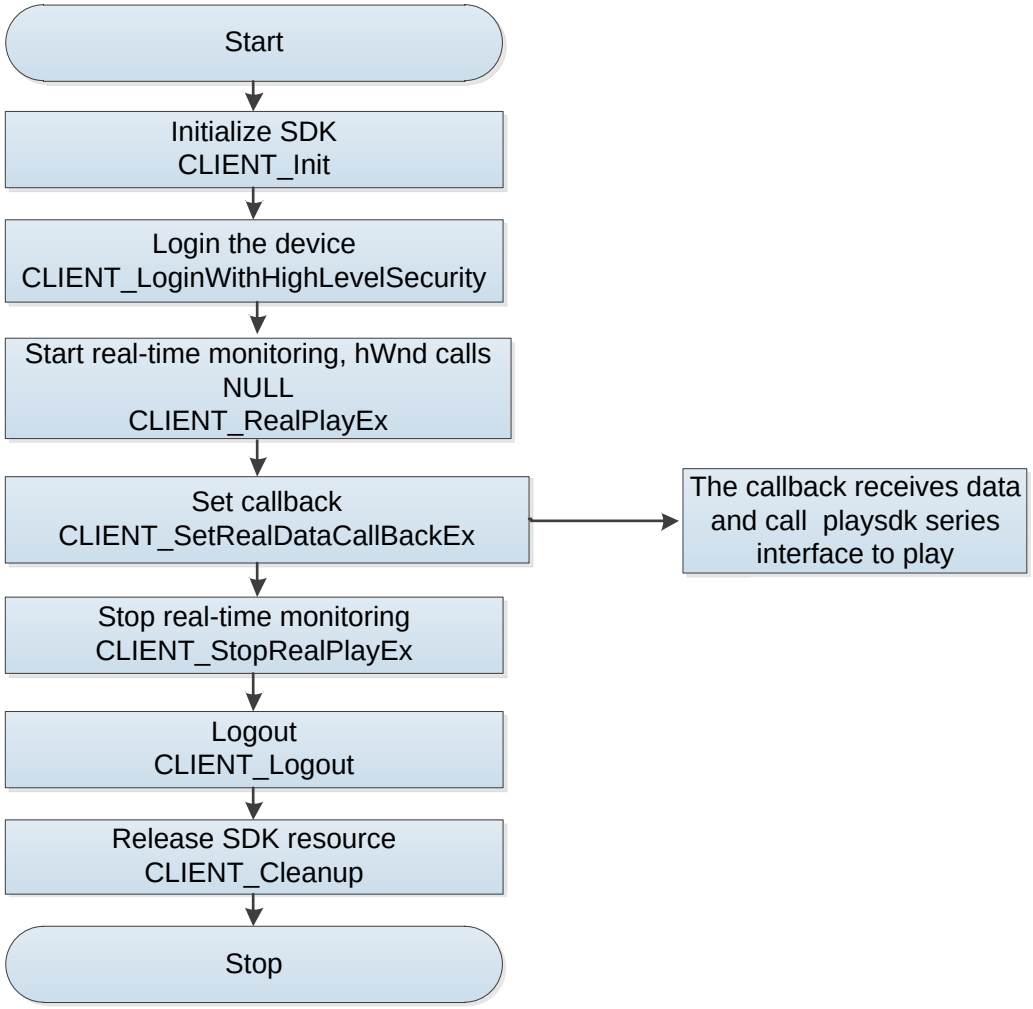
Step 4 After using the real-time function, call **CLIENT_StopRealPlayEx** to stop real-time monitoring**.**

Step 5 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 6 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.3.3.2 Third Party Decoding Play

Figure 2-4 Process of calling third party play library



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 After successful login, call **CLIENT_RealPlayEx** to enable real-time monitoring. The parameter hWnd is NULL.

Step 4 Call **CLIENT_SetRealDataCallBackEx** to set the real-time data callback.

Step 5 Save real-time data in the callback for further using. It is not recommended to do other operations in this callback other than data transfer and storage; otherwise, it will affect performance when there are many monitoring channels.

Step 6 After completing the real-time monitoring, call **CLIENT_StopRealPlayEx** to stop real-time monitoring.

Step 7 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 8 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.3.4 Example Code

2.3.4.1 SDK Decoding Play

```
#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; // Tcp connection port, which should be the same as tcp connection port of expected login device.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//************************************************************************
// 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 is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, 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 by 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 information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
```

```
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header file is decimal. Take care of conversion.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23) // Do not support this function. The corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
    // Check whether the initialization is success
    if (FALSE == g_bNetSDKInitFlag)
    {
        return;
    }
```

```
    // Check whether to log in to device
    if (0 == g_lLoginHandle)
    {
        return;
    }

    // Implement real-time monitoring
    // Get window handle of control unit
    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; // Live channel number
    DH_RealPlayType emRealPlayType =    DH_RType_Realplay; // Real-time monitoring
    g_lRealHandle = CLIENT_RealPlayEx(g_lLoginHandle, nChannelID, hWnd, emRealPlayType);
    if (0 == g_lRealHandle)
    {
        printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
    }
}

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

```
    }
    // Log out of 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 initialization resources
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

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

//**************************************************************************************
// Commonly used callback function 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 lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    printf("Call HaveReConnect\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");
}
```

2.3.4.2 Third Party Decoding Play

```
#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_lRealHandle = 0;
static char g_szDevIp[32] = "172.11.1.88";
static WORD g_nPort = 37777; // Tcp connection port, which should be the same as tcp connection port of expected login device.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//************************************************************************************
// Commonly used callback set definition.

// Callback function used when device disconnected.
// It is not recommended to call SDK interface in the SDK callback function.
// The callback is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
```

```
void CALLBACK DisConnectFunc(LLONG lLoginID, 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 by CLIENT_SetAutoReconnect. When offline device is reconnected successfully, SDK will call the function.
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// Real-time monitoring data callback
// It is not recommended to call SDK interface in the SDK callback function.
// Set the callback function by CLIENT_SetAutoReconnect. When offline device is reconnected successfully, SDK will call the function.
// It is recommended that users only do the data saving operation in this callback. You are not recommended encode and dedcode data directly.
// That is to copy the correspomding data to own storage space and then do operations such as encoding and edcodign data after leaving callback function.
void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser);

//*****************************************************************************
void InitTest()
{
    // SDK initilization
    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
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);
```

```
    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if (0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header
file is decimal. Take care of conversion.
            // For example:
```

```
                                // #define NET_NOT_SUPPORTED_EC(23) // Do not support this function. The
corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
                printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp ,
g_nPort , CLIENT_GetLastError());
            }
            else
            {
                printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
            }
            // When first time logging in, some data is needed to be initialized to enable normal business
function. It is recommended to wait for a while after login, and the waiting time varies by devices.
            Sleep(1000);
            printf("\n");
        }
}

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

    if (0 == g_lLoginHandle)
    {
        return;
    }

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

    // Start real-time monitoring
    int nChannelID = 0; // Live channel number
    DH_RealPlayType emRealPlayType =    DH_RType_Realplay; // Real-time monitoring
    g_lRealHandle = CLIENT_RealPlayEx(g_lLoginHandle, nChannelID, NULL, emRealPlayType);
    if (0 == g_lRealHandle)
    {
        printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
        return;
```

```
    }
    else
    {
        DWORD dwFlag = 0x00000001;
        if (FALSE == CLIENT_SetRealDataCallBackEx(g_lRealHandle, &RealDataCallBackEx, NULL, dwFlag))
        {
            printf("CLIENT_SetRealDataCallBackEx: failed! Error code: %x.\n", CLIENT_GetLastError());
            return;
        }
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // Stop live viewing
    if (0 != g_lRealHandle)
    {
        if (FALSE == CLIENT_StopRealPlayEx(g_lRealHandle))
        {
            printf("CLIENT_StopRealPlayEx Failed, g_lRealHandle[%x]!Last Error[%x]\n" , g_lRealHandle,
CLIENT_GetLastError());
        }
        else
        {
            g_lRealHandle = 0;
        }
    }
    // Log out of 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 initialization resources
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
}

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

//**************************************************************************************
// Commonly used callback function 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 lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    printf("Call HaveReConnect\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 RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LONG param, LDWORD dwUser)
{
  // If more than one real-time monitorings use the data callback, users can do one-to-one correspondence by
  lRealHandle.
    if (lRealHandle == g_lRealHandle)
    {
        switch(dwDataType)
        {
            case 0:
                // OriginalA/V hybrid data
                printf("receive real data, param: lRealHandle[%p], dwDataType[%d], pBuffer[%p],
dwBufSize[%d], param[%p], dwUser[%p]\n",
                        lRealHandle, 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 Introduction

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, draggering play and so on.

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.

2.4.2 Interface Overview

Table 2-4 Interfaces of record playback

Interface	Implication
CLIENT_Init	Interface for SDK initialization
CLIENT_Cleanup	Interface for cleaning up SDK resources
CLIENT_LoginWithHighLevelSecurity	Login with high level security
CLIENT_PlayBackByTimeEx	Extensive interface for playback by time
CLIENT_SetDeviceMode	Interface for setting work mode such as voice talk, playback, authority.
CLIENT_StopPlayBack	Interface for stopping record playback
CLIENT_GetPlayBackOsdTime	Interface for getting playback OSD time
CLIENT_PausePlayBack	Interface for pause or restoring playback
CLIENT_FastPlayBack	Interface for fast play.Increasing frame rate by 1x
CLIENT_SlowPlayBack	Interface for slow play.Decreasing frame rate by 1x
CLIENT_NormalPlayBack	Interface for restoring normal play speed
CLIENT_SeekPlayBack	Interface for positioning record playback start point
CLIENT_Logout	Interface for logout
CLIENT_GetLastError	Interface for getting error code after failed calling interface

2.4.3 Process

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

- SDK decoding playback

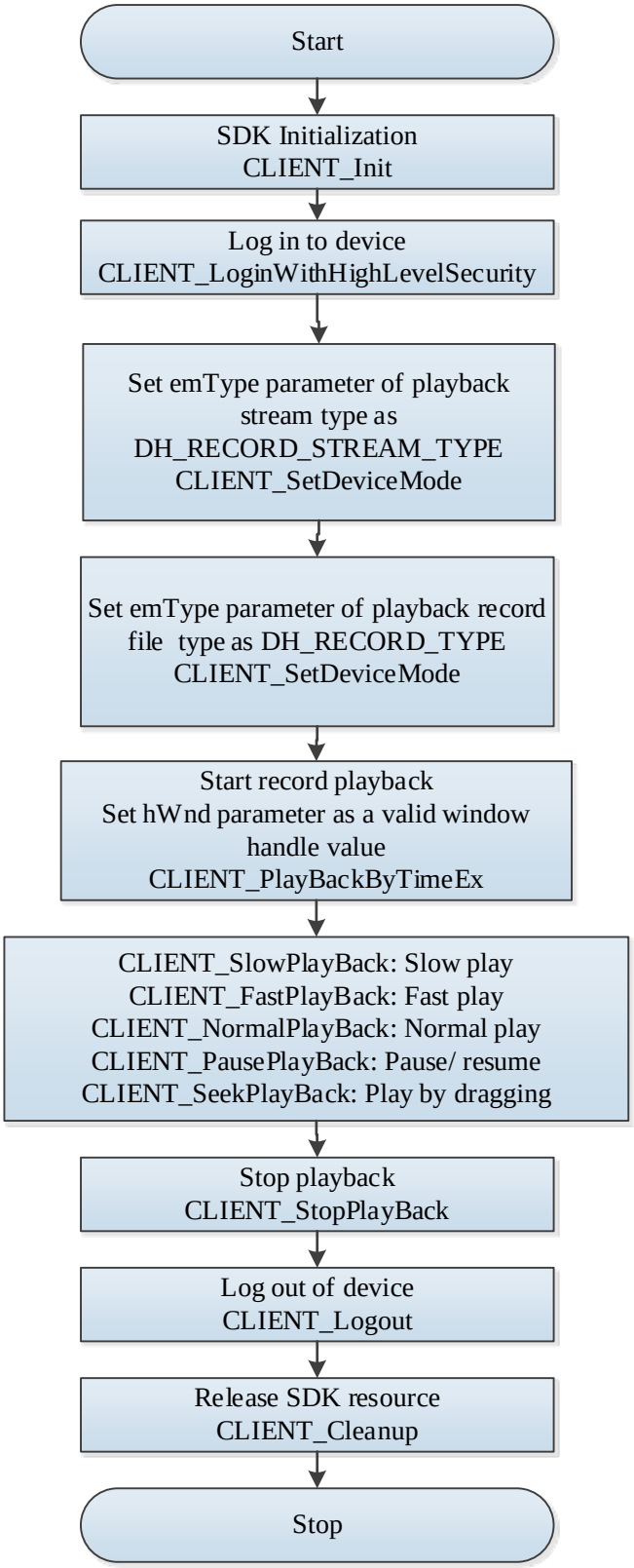
Firstly user inputs start time, end time and valid window handle of record, then SDK will call corresponding decoding library to analyze stream and show the video in display window.

- Third party decoding playback

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 Playback

Figure 2-5 Process of SDK decoding playback



Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 After successful login, call **CLIENT_SetDeviceMode** twice to separately set playback stream type and playback record file type.

Step 4 Call **CLIENT_PlayBackByTimeEx** to start playback, parameter hWnd is set to valid window handle value.

Step 5 During playback, call **CLIENT_SlowPlayBack** to slowly play, **CLIENT_FastPlayBack** to fast play, **CLIENT_NormalPlayBack** to play at normal speed, **CLIENT_PausePlayBack** to pause or resuem play, **CLIENT_SeekPlayBack** to play by dragging.

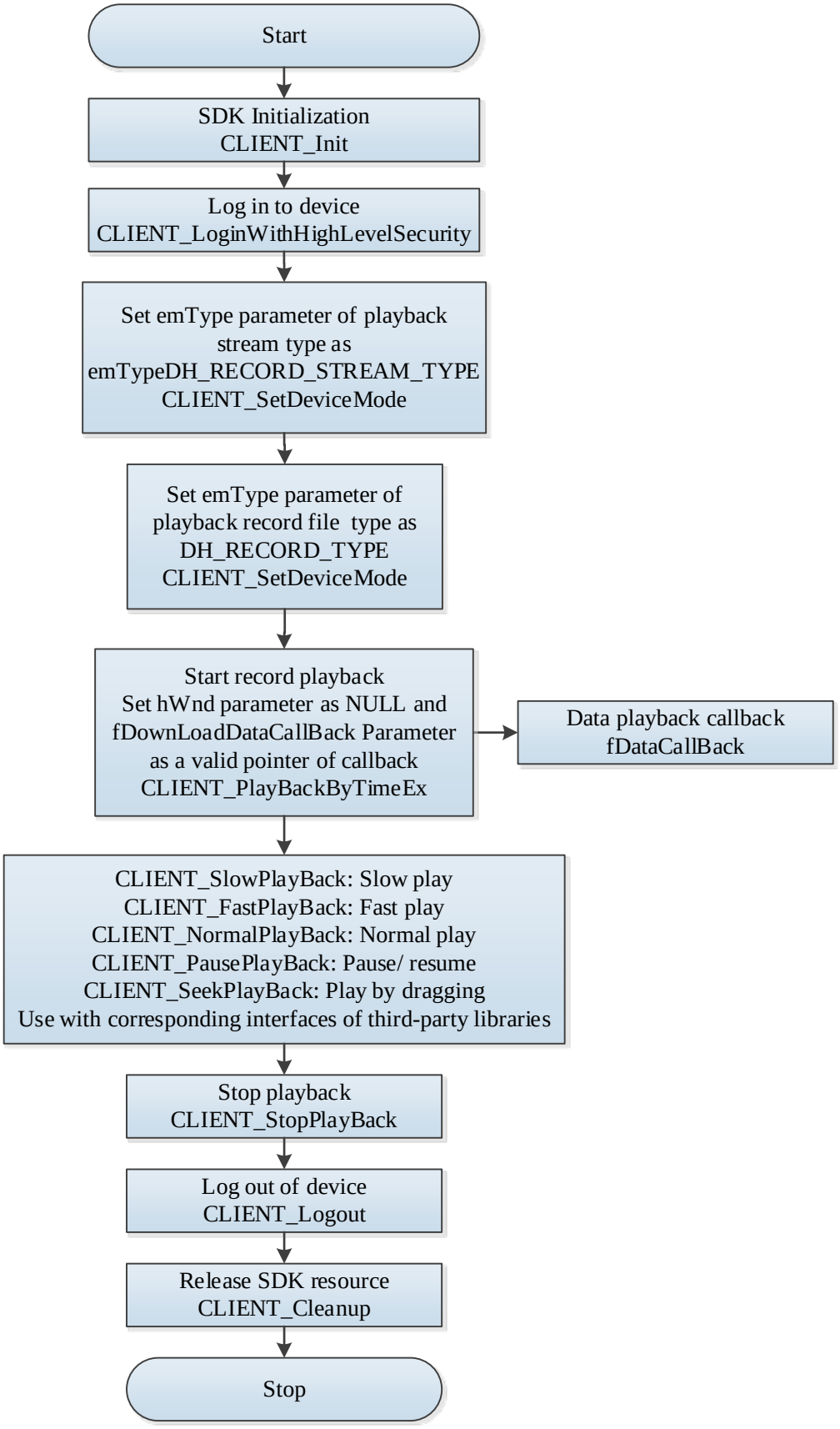
Step 6 After playback is done, call **CLIENT_StopPlayBack** to stop playback.

Step 7 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 8 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.4.3.2 Third Party Decoding Playback

Figure 2-6 Process of third party decoding playback



Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 After successful login, call **CLIENT_SetDeviceMode** twice to separately set playback stream type and playback record file type.

Step 4 After successful login, call **CLIENT_PlayBackByTimeEx** to start playback. The parameter hWnd is set to NULL, and parameter fDownLoadDataCallBack is a valid pointer pointing to a callback function.

Step 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.

Step 6 During playback, call **CLIENT_SlowPlayBack** to slowly play, **CLIENT_FastPlayBack** to fast play, **CLIENT_NormalPlayBack** to play at normal speed, **CLIENT_PausePlayBack** to pause or resuem play, **CLIENT_SeekPlayBack** to play by dragging and call the third-party interfaces at the same time.

Step 7 After playback is done, call **CLIENT_StopPlayBack** to stop playback.

Step 8 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 9 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.4.4 Example Code

2.4.4.1 SDK Decoding Playback

```
#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; // Tcp connection port, which should be the same as tcp connection port of expected login device.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//************************************************************************
// 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 is set by CLIENT_Init. When the device is offline, SDK will call this callback function..
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

/// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in the SDK callback function.
// Set the callback function by 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 information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

```
    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header
file is decimal. Take care of conversion.
            // For example:
           // #define NET_NOT_SUPPORTED_EC(23)          // Do not support this function. The
corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp ,
g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business
function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
```

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

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

        if (0 == g_lLoginHandle)
        {
            return;
        }

        // Record playback
        // Get window handle of control unit
        HWND hWnd = GetConsoleWindow();

        // Set bit stream of playback
        int nStreamType = 0; // 0-main and sub stream, 1-main stream, 2-sub stream
        CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

        // Set playback record file type
        NET_RECORD_TYPE emFileType = NET_RECORD_TYPE_ALL; // All recorded videos
        CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_TYPE, &emFileType);

        // Start record playback
        int nChannelID = 0; // Channel number
        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_lPlayHandle = CLIENT_PlayBackByTimeEx(g_lLoginHandle, nChannelID, &stuStartTime, &stuStopTime,
hWnd, NULL, NULL, NULL, NULL);
    if (0 == g_lPlayHandle)
    {
        printf("CLIENT_PlayBackByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError());
    }

    // Implement playback controlling as needed
    // The example code is for reference because it is a control unit demo that cannot display the record
palyback and playback controlling at the same time.

    // CLIENT_SlowPlayBack To slow play
    /* Example code
    if (FALSE == CLIENT_SlowPlayBack (g_lPlayHandle))
    {
        printf("CLIENT_SlowPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
CLIENT_GetLastError());
    }
    */

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

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

    // CLIENT_PausePlayBack To pause and resume play
    /* Example code
```

```
    if (FALSE == CLIENT_PausePlayBack (g_lPlayHandle, TRUE))
    {
        printf("CLIENT_PausePlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
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 stuStartTime.
    if (FALSE == CLIENT_SeekPlayBack (g_lPlayHandle, nOffsetSeconds, 0))
    {
        printf("CLIENT_SeekPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
CLIENT_GetLastError());
    }
    */
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // Close playback
    if (0 != g_lPlayHandle)
    {
        if (FALSE == CLIENT_StopPlayBack(g_lPlayHandle))
        {
            printf("CLIENT_StopPlayBack Failed, g_lRealHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
CLIENT_GetLastError());
        }
        else
        {
            g_lPlayHandle = 0;
        }
    }
    // Log out of 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 initialization resources
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

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

//**************************************************************************************
// Commonly used callback function 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 lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
	printf("Call HaveReConnect\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");
}
```

2.4.4.2 Third Party Decoding Playback

```
#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_lPlayHandle = 0L;
static char g_szDevIp[32] = "172.11.1.6";
static WORD g_nPort = 37777; // Tcp connection port, which should be the same as tcp connection port of expected login device.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//************************************************************************************
// Commonly used callback set definition.

// Callback function used when device disconnected.
// It is not recommended to call SDK interface in the SDK callback function.
// The callback is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, 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 by CLIENT_SetAutoReconnect. When offline device is reconnected successfully, SDK will call the function.
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// Playback progress callback
// It is not recommended to call SDK interface in this function.
// Set the callback function by CLIENT_PlayBackByTimeEx. When you receive playback data from device, SDK will call the function.
void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);

// Playback data callback
// It is not recommended to call SDK interface in this function.
// When you set this callback, if hWnd is NULL, returned parameter 0 means that the callback failed and the next callingwill return the same data, and returned parameter means the callback succeeded and the next calling will return the following data.
// When you set this callback, if hWnd is not NULL, the callback succeeded no matter how much the return value and the next calling will return the following data.
// Set the callback function by CLIENT_PlayBackByTimeEx. When you receive playback data from device, SDK will call the function.
int CALLBACK DataCallBack(LLONG lRealHandle, 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 information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);
```

```
    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header
file is decimal. Take care of conversion.
```

```
                // For example:
                // #define NET_NOT_SUPPORTED_EC(23)                // Do not support this function. The corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
                printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
            }
            else
            {
                printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
            }
            // When first time logging in, some data is needed to be initialized to enable normal business function. It is recommended to wait for a while after login, and the waiting time varies by devices.
            Sleep(1000);
            printf("\n");
        }
}

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

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // Record playback

    // Set bit stream of playback
    int nStreamType = 0; // 0-main and sub stream, 1-main stream, 2-sub stream
    CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

    // Set playback record file type
    NET_RECORD_TYPE emFileType = NET_RECORD_TYPE_ALL; // All recorded videos
    CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_TYPE, &emFileType);

    // Start record playback
```

```
int nChannelID = 0; // Channel number
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;

// Function parameter hWnd should be NULL
// Function parameter fDownLoadDataCallBack should be a valid callback function pointer
g_lPlayHandle = CLIENT_PlayBackByTimeEx(g_lLoginHandle, nChannelID, &stuStartTime, &stuStopTime,
NULL, &DownLoadPosCallBack, NULL, &DataCallBack, NULL);
if (g_lPlayHandle == 0)
{
    printf("CLIENT_PlayBackByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}

// Implement playback controlling as needed
// Call the corresponding controlling interfaces of third party when call SDK playback controlling interface
because it is the third party library decoding.
// The example code is for reference because it is a control unit demo that cannot display the record
palyback and playback controlling at the same time.

// CLIENT_SlowPlayBack To slow play
/* Example code
if (FALSE == CLIENT_SlowPlayBack (g_lPlayHandle))
{
    printf("CLIENT_SlowPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
CLIENT_GetLastError());
}
// Call corresponding interface of third party library

*/

// CLIENT_FastPlayBack To fast play
/* Example code
if (FALSE == CLIENT_FastPlayBack (g_lPlayHandle))
```

```
    {
        printf("CLIENT_FastPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
CLIENT_GetLastError());
    }
    // Call corresponding interface of third party library

    */

    // CLIENT_NormalPlayBack To play at t normal speed
    /* Example code
    if (FALSE == CLIENT_NormalPlayBack (g_lPlayHandle))
    {
        printf("CLIENT_NormalPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
CLIENT_GetLastError());
    }
    // Call corresponding interface of third party library

    */

    // CLIENT_PausePlayBack To pause and resume play
    /* Example code
    if (FALSE == CLIENT_PausePlayBack (g_lPlayHandle, TRUE))
    {
        printf("CLIENT_PausePlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
CLIENT_GetLastError());
    }
    // Call corresponding interface of third party library

    */

    // CLIENT_SeekPlayBack To play by dragging
    /* Example code
    int nOffsetSeconds = 2 * 60 * 60; // Drag to 2*60*60s after stuStartTime to start play stuStartTime.
    if (FALSE == CLIENT_SeekPlayBack (g_lPlayHandle, nOffsetSeconds, 0))
    {
        printf("CLIENT_SeekPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
CLIENT_GetLastError());
    }
    // Call corresponding interface of third party library
```

```
    */
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // Close playback
    if (0 != g_lPlayHandle)
    {
        if (FALSE == CLIENT_StopPlayBack(g_lPlayHandle))
        {
            printf("CLIENT_StopPlayBack Failed, g_lRealHandle[%x]!Last Error[%x]\n" , g_lPlayHandle,
CLIENT_GetLastError());
        }
        else
        {
            g_lPlayHandle = 0;
        }
    }
    // Log out of 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 initialization resources
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}
```

```
int main()
{
	InitTest();

	RunTest();

	EndTest();

	return 0;
}

//****************************************************************************
/ Commonly used callback function 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 lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
	printf("Call HaveReConnect\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 DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize,
LDWORD dwUser)
{
    // If more than one playbacks or downloads use the progress callback, users can do one-to-one
correspondence by lPlayHandle.
    if (lPlayHandle == g_lPlayHandle)
    {
        printf("lPlayHandle[%p]\n", lPlayHandle);
        printf("dwTotalSize[%d]\n", dwTotalSize);
        printf("dwDownLoadSize[%d]\n", dwDownLoadSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
    }
}

int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize,
LDWORD dwUser)
{
    int nRet = 0;
    printf("call DataCallBack\n");
    // If more than one playbacks or downloads use the progress callback, users can do one-to-one
correspondence by lRealHandle.
    if(lRealHandle == g_lPlayHandle)
    {
        BOOL bSuccess = TRUE;
        // The following print will result in screen brushing during playback and download. Take care.
        printf("lPlayHandle[%p]\n", lRealHandle);
        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
            // Uses can save bit stream data here, and do other operations after leaving callback such as
decoding and forwarding.
            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 Introduction

Video surveillance system widely applies to city, airport, metro, bank and factory. When any event occurs, you need to download the video records and report to the leaders, public security bureau, or mass media. Therefore, record download is an important function.

The record download function helps you obtain the records saved on the device through SDK and save into the local. It allows you to download from the selected channels and export to the local disk or external USB flash drive.

Record download have two methods: download by file and download by time.

2.5.2 Interface Overview

Table 2-5 Interfaces of record download

Interface	Implication
CLIENT_Init	Interface for SDK initialization.
CLIENT_Cleanup	Interface for cleaning up SDK resources.
CLIENT_LoginWithHighLevelSecurity	Login with high level security
CLIENT_SetDeviceMode	Interface for setting work mode of device voice talk, playback and right
CLIENT_QueryRecordFile	Interface for searching all files in a specified time period
CLIENT_FindFile	Interface for opening record search handle
CLIENT_FindNextFile	Interface for searching record file
CLIENT_FindClose	Interface for closing record search handle
CLIENT_DownloadByRecordFileEx	Extensive interface for downloading record by file
CLIENT_DownloadByTimeEx	Extensive interface for downloading record by time
CLIENT_GetDownloadPos	Interface for searching record download process
CLIENT_StopDownload	Interface for stopping record download
CLIENT_Logout	Interface for logout
CLIENT_GetLastError	Interface for getting error code after failed calling interface.

2.5.3 Process

Record download includes the following two methods.

Download by file

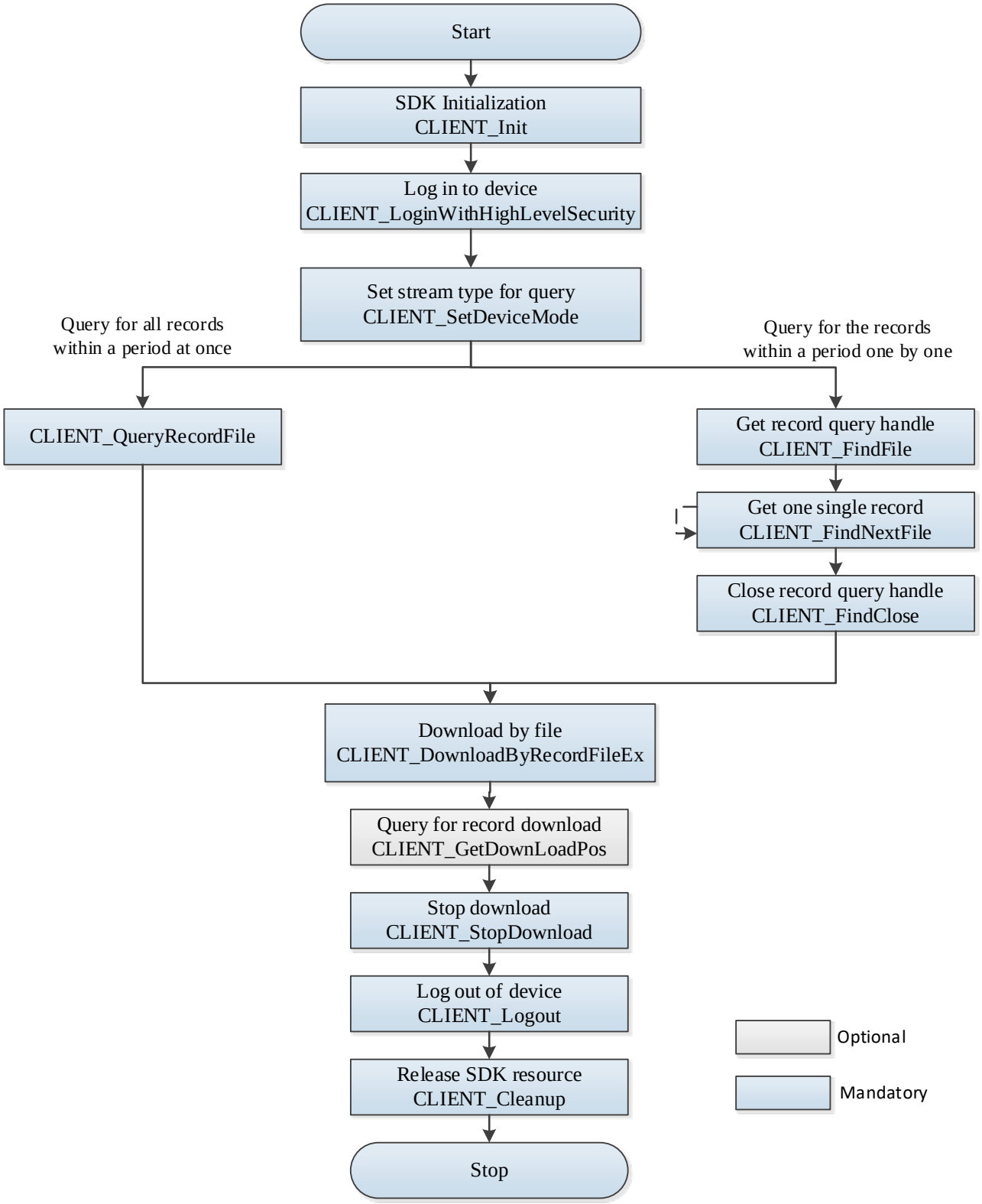
Users need to point the downloaded record file's information and SDK can download the specified file and save it to a specified file. At the same time, user can also provide a callback function pointer, so that SDK send the downloaded file info to users for further use by callbackfunction.

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

Figure 2-7 Process of download by file



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call **CLIENT_SetDeviceMode** to set the stream type, and set parameter emType as DH_RECORD_STREAM_TYPE. It is recommend to set stream as 0-mian ans sub stream, otherwise some devices might be unable to get results. If you only need main stream recordings, you can filter sub stream recordings of results.

Step 4 Query the record files by one of the following two ways:

- Call **CLIENT_FindFile** to obtain the record query handle, and then call **CLIENT_FindNextFile** several times to obtain the record file information and then call **CLIENT_FindClose** to close the record query handle at last.
- Call **CLIENT_QueryRecordFile** to obtain all the record files information for a period one time.

Step 5 After getting the record file information, call **CLIENT_DownloadByRecordFileEx** to start downloading record files. At least one of the sSavedFileName and fDownLoadDataCallBack should be valid.

Step 6 During downloading, call **CLIENT_GetDownloadPos** to query the record downloading progress.

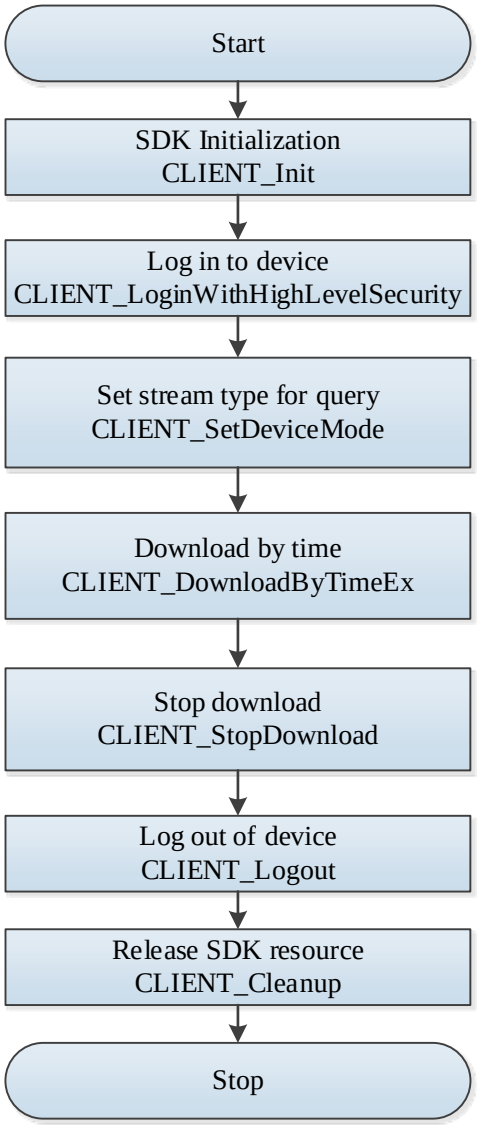
Step 7 Call **CLIENT_StopDownload** to stop download.

Step 8 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 9 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.5.3.2 Download by Time

Figure 2-8 Process of download by time



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call **CLIENT_SetDeviceMode** to set the stream type, and set parameter emType as DH_RECORD_STREAM_TYPE.

Step 4 Call **CLIENT_DownloadByTimeEx** to start downloading by time. At least one of the sSavedFileName and fDownLoadDataCallBack should be valid.

Step 5 Call **CLIENT_StopDownload** to stop download**.** You can close the download process after it is completed or it is just partially completed.

Step 6 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 7 After using all SDK functions, 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; // Tcp connection port, which should be the same as tcp connection port of expected login device.
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 disconnected.
// It is not recommended to call SDK interface in the SDK callback function.
// The callback is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in this function.
// Set the callback function by CLIENT_SetAutoReconnect. When offline device is reconnected successfully, SDK will call the function.
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// Playback/ download progress callback
// It is not recommended to call SDK interface in this 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 lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize,
LDWORD dwUser);

// Playback/ download progress callback
// It is not recommended to call SDK interface in this 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 lRealHandle, 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 information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);


    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);
```

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if (0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header
file is decimal. Take care of conversion.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23)                // Do not support this function. The
corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp ,
g_nPort , CLIENT_GetLastError());
        }
```

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

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

        if (0 == g_lLoginHandle)
        {
            return;
        }
        // Recorded files search
        // Set stream type of recordings
          int nStreamType = 0; // 0-main and sub stream, 1-main stream, 2-sub stream
          CLIENT_SetDeviceMode(g_lLoginHandle, 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, and the first method can see CLIENT_QueryRecordFile interface.
        int nChannelID = 0; // Channel number
        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 lFindHandle = CLIENT_FindFile(g_lLoginHandle, nChannelID, 0, NULL, &stuStartTime, &stuStopTime,
FALSE, 5000);
    if (0 == lFindHandle)
    {
        printf("CLIENT_FindFile Failed!Last Error[%x]\n",CLIENT_GetLastError());
        return;
    }
    // Example code of demo which takes max supported g_nMaxRecordFileCountrecorded files as an
example.
    std::vector<NET_RECORDFILE_INFO> bufFileInfo(g_nMaxRecordFileCount);

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

    // Stop searching
    if(0 != lFindHandle)
    {
        CLIENT_FindClose(lFindHandle);
    }
    // Set the first searched file as download file
    NET_RECORDFILE_INFO stuNetFileInfo;
    if (nFileIndex > 0)
    {
        memcpy(&stuNetFileInfo, (void *)&bufFileInfo[0], sizeof(stuNetFileInfo));
    }
    else
    {
        printf("no record, return\n");
        return;
```

```
    }

    // Recorded file download
    // Start recordings download
    // At least one of the function parameters sSavedFileName and fDownLoadDataCallBack shold be valid.
    // In pratical, save directly to sSavedFileName or call back to process data ass needed.
    g_lDownloadHandle = CLIENT_DownloadByRecordFileEx(g_lLoginHandle, &stuNetFileInfo, "test.dav",
DownLoadPosCallBack, NULL, DataCallBack, NULL);
    if (0 == g_lDownloadHandle)
    {
        printf("CLIENT_DownloadByRecordFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // Stop download, users can call this interface after download ends or during download.
    if (0 != g_lDownloadHandle)
    {
        if (FALSE == CLIENT_StopDownload(g_lDownloadHandle))
        {
            printf("CLIENT_StopDownload Failed, g_lDownloadHandle[%x]!Last Error[%x]\n" ,
g_lDownloadHandle, CLIENT_GetLastError());
        }
        else
        {
            g_lDownloadHandle = 0;
        }
    }
    // Log out of 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 initialization resources
        if (TRUE == g_bNetSDKInitFlag)
        {
            CLIENT_Cleanup();
            g_bNetSDKInitFlag = FALSE;
        }

        return;
}

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

//**************************************************************************************
// Commonly used callback function 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 lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
        printf("Call HaveReConnect\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 DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize,
LDWORD dwUser)
{
    // If more than one playbacks or downloads use the progress callback, users can do one-to-one
correspondence by lPlayHandle.
    if (lPlayHandle == g_lDownloadHandle)
    {
        printf("lPlayHandle[%p]\n", lPlayHandle);
        printf("dwTotalSize[%d]\n", dwTotalSize);
        printf("dwDownLoadSize[%d]\n", dwDownLoadSize);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
    }
}

int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize,
LDWORD dwUser)
{
    int nRet = 0;
    printf("call DataCallBack\n");
    // If more than one playbacks or downloads use the progress callback, users can do one-to-one
correspondence by lPlayHandle.
    if(lRealHandle == g_lDownloadHandle)
    {
        printf("lPlayHandle[%p]\n", lRealHandle);
        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; // Tcp connection port, which should be the same as tcp connection port of expected login device.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//*************************************************************************
// Commonly used callback set definition.

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

// Callback function used when device disconnected.
// It is not recommended to call SDK interface in the SDK callback function.
// The callback is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// Playback by time progress callback function
// It is 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 lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser);

// Playback/download data callback function
// It is 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 lRealHandle, 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 information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);


    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
   CLIENT_SetNetworkParam(&stuNetParm);
```

```
    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header
file is decimal. Take care of conversion.
            // For example:
           // #define NET_NOT_SUPPORTED_EC(23) // Do not support this function. The corresponding
error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp ,
g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business
function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{
```

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

    if (0 == g_lLoginHandle)
    {
        return;
    }
    // Recorded files search
    // Set stream type of recordings
      int nStreamType = 0; // 0-main and sub stream, 1-main stream, 2-sub stream
    CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

    int nChannelID = 0; // channel number
    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;

    // Implement record download

    // Start recordings download
    // At least one of the function parameters sSavedFileName and fDownLoadDataCallBack shold be valid.
    g_lDownloadHandle = CLIENT_DownloadByTimeEx(g_lLoginHandle, nChannelID,
EM_RECORD_TYPE_ALL, &stuStartTime, &stuStopTime, "test.dav", TimeDownLoadPosCallBack, NULL,
DataCallBack, NULL);
    if (g_lDownloadHandle == 0)
    {
        printf("CLIENT_DownloadByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError());
    }
}

void EndTest()
{
```

```
    printf("input any key to quit!\n");
    getchar();
    // Stop download, users can call this interface after download ends or during download.
    if (0 != g_lDownloadHandle)
    {
        if (FALSE == CLIENT_StopDownload(g_lDownloadHandle))
        {
            printf("CLIENT_StopDownload Failed, g_lDownloadHandle[%x]!Last Error[%x]\n" ,
g_lDownloadHandle, CLIENT_GetLastError());
        }
        else
        {
            g_lDownloadHandle = 0;
        }
    }
    // Log out of 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 initialization resources
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}

int main()
{
    InitTest();
    RunTest();
```

```
    EndTest();
    return 0;
}

//**************************************************************************************
// Commonly used callback function 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 lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    printf("Call HaveReConnect\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 TimeDownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser)
{
    // If more than one playbacks or downloads use the progress callback, users can do one-to-one
correspondence by lPlayHandle.
    if (lPlayHandle == g_lDownloadHandle)
    {
```

```
            printf("lPlayHandle[%p]\n", lPlayHandle);
            printf("dwTotalSize[%d]\n", dwTotalSize);
            printf("dwDownLoadSize[%d]\n", dwDownLoadSize);
            printf("index[%d]\n", index);
            printf("dwUser[%p]\n", dwUser);
            printf("\n");
        }
}

int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize,
LDWORD dwUser)
{
        int nRet = 0;
        printf("call DataCallBack\n");
        // If more than one playbacks or downloads use the progress callback, users can do one-to-one
correspondence by lPlayHandle.
        if(lRealHandle == g_lDownloadHandle)
        {
            printf("lPlayHandle[%p]\n", lRealHandle);
            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 Introduction

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 camere.

PTZ control is an important part of a surveillance system.Users have different demands for suiveillance 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

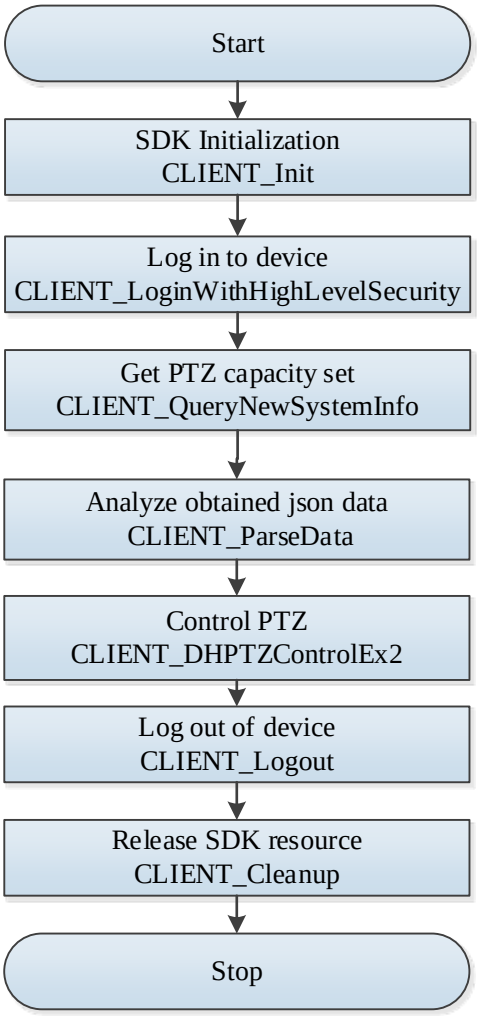
Table 2-6 Interfaces of PTZ control

Interface	Implication
CLIENT_Init	Interface for SDK initialization.
CLIENT_Cleanup	Interface for cleaning up SDK resources.
CLIENT_LoginWithHighLevelSecurity	Login with high level security.
CLIENT_ParseData	Interface for analyzing the obtained config info.
CLIENT_DHPTZControlEx2	Extensive interface for private PTZ control.
CLIENT_QueryNewSystemInfo	Interface for obtaining new system capacity set.
CLIENT_Logout	Interface for logout device.
CLIENT_GetLastError	Interface for getting error code after failed calling

Interface	Implication
	interface.

2.6.3 Process

Figure 2-9 Process of PTZ control



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 After successfully login, call **CLIENT_QueryNewSystemInfo** and obtain PTZ capacity set by CFG_CAP_CMD_PTZ; and then call **CLIENT_ParseData** and analyze PTZ capacity set by CFG_CAP_CMD_PTZ.

Step 4 Call **CLIENT_DHPTZControlEx2** as needed to operate PTZ. Different PTZ command may requires different parameters.,and some commands may require corresponding stopping command, such as left/right movement. For details, see example code.

Step 5 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 6 After using all SDK functions, 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_lLoginHandle = 0L;
static LLONG g_lDownloadHandle = 0L;
static char g_szDevIp[32] = "171.2.7.34";
static int    g_nPort = 37777; // Tcp connection port, which should be the same as tcp connection port of expected login device.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//************************************************************************************
// 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 is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in this function.
// Set the callback function by 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 information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
   CLIENT_SetNetworkParam(&stuNetParm);
```

```
    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header file is decimal. Take care of conversion.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23) // Do not support this function. The corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
        printf("\n");
    }
}
```

```
// Ptz control info structure
typedef struct tagPtzControlInfo
{
    tagPtzControlInfo():m_iCmd(-1), m_bStopFlag(false){}
    tagPtzControlInfo(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. Call corresponding stop operations after start.
}PtzControlInfo;

// 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<PtzControlInfo> 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.bIris)
    {
        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",
false));
        vecPtzControl.push_back(PtzControlInfo(int(DH_PTZ_POINT_SET_CONTROL), " set preset ", false));
    }

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

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

    if (TRUE == stuPtzCapsInfo.bRemoveTour)
    {
```

```
        vecPtzControl.push_back(PtzControlInfo(int(DH_EXTPTZ_CLOSELOOP), "clear tour", false));
    }

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

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

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

    vecPtzControl.push_back(PtzControlInfo(int(-2), "pause", false));
    vecPtzControl.push_back(PtzControlInfo(int(-1), "exit", true));
    PtzControlInfo cLastChoose;
    while(TRUE)
    {
        printf("PTZ control operation: \n");
        for (std::vector<PtzControlInfo>::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<PtzControlInfo>::iterator iterFind = vecPtzControl.begin();
        for (; iterFind != vecPtzControl.end(); ++iterFind)
        {
            if (nChoose == iterFind->m_iCmd)
            {
                break;
            }
        }

        if (iterFind == vecPtzControl.end())
        {
            printf("input operation within range\n");
            continue;
        }
        // Stop the last operation
        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)
        {
            // lParam2 is preset number
            printf("\t preset number (%2d-%2d):", stuPtzCapsInfo.wPresetMin,stuPtzCapsInfo.wPresetMax);
            scanf("%d", &lParam2);
        }
        else if (DH_PTZ_POINT_LOOP_CONTROL == iterFind->m_iCmd)
```

```
        {
            // lParam1 is scan path, lParam3: 76 sartt and 96 stop
            printf("\t scan path (%2d-%2d):", stuPtzCapsInfo.wTourMin,stuPtzCapsInfo.wTourMax);
            scanf("%d", &lParam1);
            printf("\t1:start \n\t2: stop \n\t select:");
            int nTmp = 0;
            scanf("%d", &nTmp);
            if (1 == nTmp)
            {
                lParam3 = 76;
            }
            else if (2 == nTmp)
            {
                lParam3 = 96;
            }
        }
        else if (DH_PTZ_LAMP_CONTROL == iterFind->m_iCmd)
        {
            // lParam1 is switch control
            printf("\t1:start \n\t2: stop \n\t select:");
            scanf("%d", &lParam1);
        }
        else if (DH_EXTPTZ_LEFTTOP <= iterFind->m_iCmd && iterFind->m_iCmd <=
DH_EXTPTZ_RIGHTDOWN)
        {
            // vertical speed, valid range (1-8)
            lParam1 = 1;
            // horizontal speed, valid range (1-8)
            lParam2 = 1;
        }
        else if (DH_EXTPTZ_ADDTOLOOP <= iterFind->m_iCmd && iterFind->m_iCmd <=
DH_EXTPTZ_DELFROMLOOP)
        {
            // lParam1 is tour path
            printf("\t scan path (%2d-%2d):", stuPtzCapsInfo.wTourMin,stuPtzCapsInfo.wTourMax);
```

```
            scanf("%d", &lParam1);
            // lParam2 is tour number
            printf("\t preset number (%2d-%2d):", stuPtzCapsInfo.wPresetMin,stuPtzCapsInfo.wPresetMax);
            scanf("%d", &lParam2);
        }
        else if (DH_EXTPTZ_CLOSELOOP == iterFind->m_iCmd)
        {
            // lParam1 is tour path
            printf("\t tour path (%2d-%2d):", stuPtzCapsInfo.wTourMin,stuPtzCapsInfo.wTourMax);
            scanf("%d", &lParam1);
        }
        else if (DH_EXTPTZ_FASTGOTO == iterFind->m_iCmd)
        {
            // Horizontal coordinate, valid range (-8191 ~ 8191)
            lParam1 = 2000;
            // Vertical coordinate, valid range (-8191 ~ 8191)
            lParam2 = 2000;
            // Zoom, valid range (-16 ~ 16)
            lParam3 = 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", &lParam1);
            // Vertical coordinate, valid range and accuracy is 10x of capacity set acquisition range.
            printf("\t vertical coordinate    (%2d-%2d):",
10*stuPtzCapsInfo.stuPtzMotionRange.nVerticalAngleMin,
10*stuPtzCapsInfo.stuPtzMotionRange.nVerticalAngleMax);
            scanf("%d", &lParam2);
            // zoom, valid range (1 ~ 128)
            lParam3 = 2;
        }
```

```
            if (FALSE == CLIENT_DHPTZControlEx2(g_lLoginHandle, nChannelId, iterFind->m_iCmd, lParam1,
lParam2, lParam3, 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();
    // Log out of 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 uo initilization resources
    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 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("lLoginID[0x%x]", lLoginID);
	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)
        {
                // Parameter error
                nError = -1;
        }
        else
        {
                nError = 0;
        }

        return nGet;
}
```

2.7 Voice Talk

2.7.1 Introduction

Voice talk realizes the voice interaction between the local platform and the environment where front-end devices are located.

This section introduces how to use SDK to realize the voice talk with the front-end devices.

Voice talk has two modes: client mode and server mode.

2.7.2 Interface Overview

Table 2-7 Interfaces of voice talk

Interface	Implication
CLIENT_Init	Interface for SDK Initialization.
CLIENT_Cleanup	Interface for cleaning up SDK resources.
CLIENT_LoginWithHighLevelSecurity	Login with high level security.
CLIENT_QueryDevState	Interface for searching device status.
CLIENT_GetDevConfig	Extensive interface for opening voice talk.
CLIENT_StartTalkEx	Extensive interface for stopping voice talk.
CLIENT_StopTalkEx	Extensive interface for starting client recording(valid in Windows platform only).
CLIENT_RecordStartEx	Extensive interface for stopping client recording(valid in Windowsplatform only).
CLIENT_RecordStopEx	Interface for sending audio data to device
CLIENT_TalkSendData	Extensive interface for decoding audio data(valid in Windows platform only).
CLIENT_AudioDecEx	Interface for logout.
CLIENT_Logout	Interface for getting error code after failed calling interface.
CLIENT_GetLastError	Interface for sending audio data to device.

2.7.3 Process

Voice talk has two modes.

Client mode

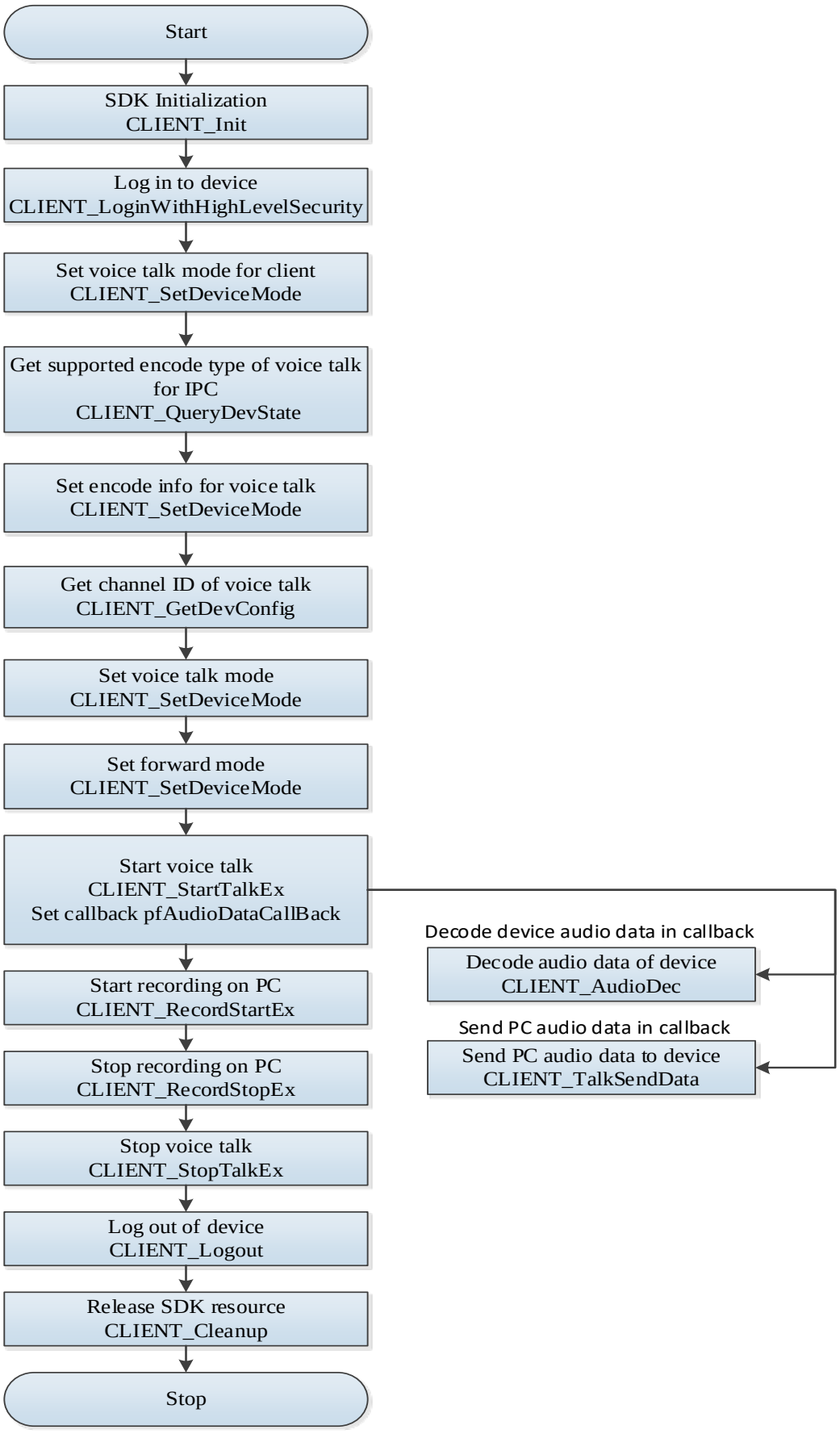
SDK allows user to provide a callback function.The callback function is called when SDK collects audio data from local sound card or receives data from the front-end.In callback function user can not only send collected local audio data to front-end device but decode and play the received front-end audio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

Figure 2-10 Process of client mode



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call **CLIENT_SetDeviceMode** to set voice talk mode as clinet mode, and set the parameter emType as DH_TALK_CLIENT_MODE.

Step 4 Call **CLIENT_QueryDevState** to get aupported voice talk encoding type list, and set the parameter nType as DH_DEVSTATE_TALK_ECTYPE.

Step 5 Call **CLIENT_SetDeviceMode** to set voice talk decoding info and set the parameter emType as DH_TALK_ENCODE_TYPE.

Step 6 Call **CLIENT_GetDevConfig** to get voice talk channel number and set parameter dwCommand as DH_DEV_DEVICECFG. If the acquired channel number is o, use o channel by default.

Step 7 Call **CLIENT_SetDeviceMode** to set voice talk parameter and set the parameter emType as DH_TALK_SPEAK_PARAM.

Step 8 Call **CLIENT_SetDeviceMode** to set voice talk transfer mode. No-transfer mode is to implement voice talk between local PC and logined device; and transfer mode is to implement voice talk between local PC and front-end device connected with specific channel of the logined device.

Step 9 Call **CLIENT_StartTalkEx** to set callback function and start voice talk.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.

Step 10 Call **CLIENT_RecordStartEx** to start PC recording.Only after this interface is called, can avoice talk callback function set by **CLIENT_StartTalkEx** will receive local audio data.

Step 11 After voice talk is finished, call **CLIENT_RecordStopEx** to stop PC recording.

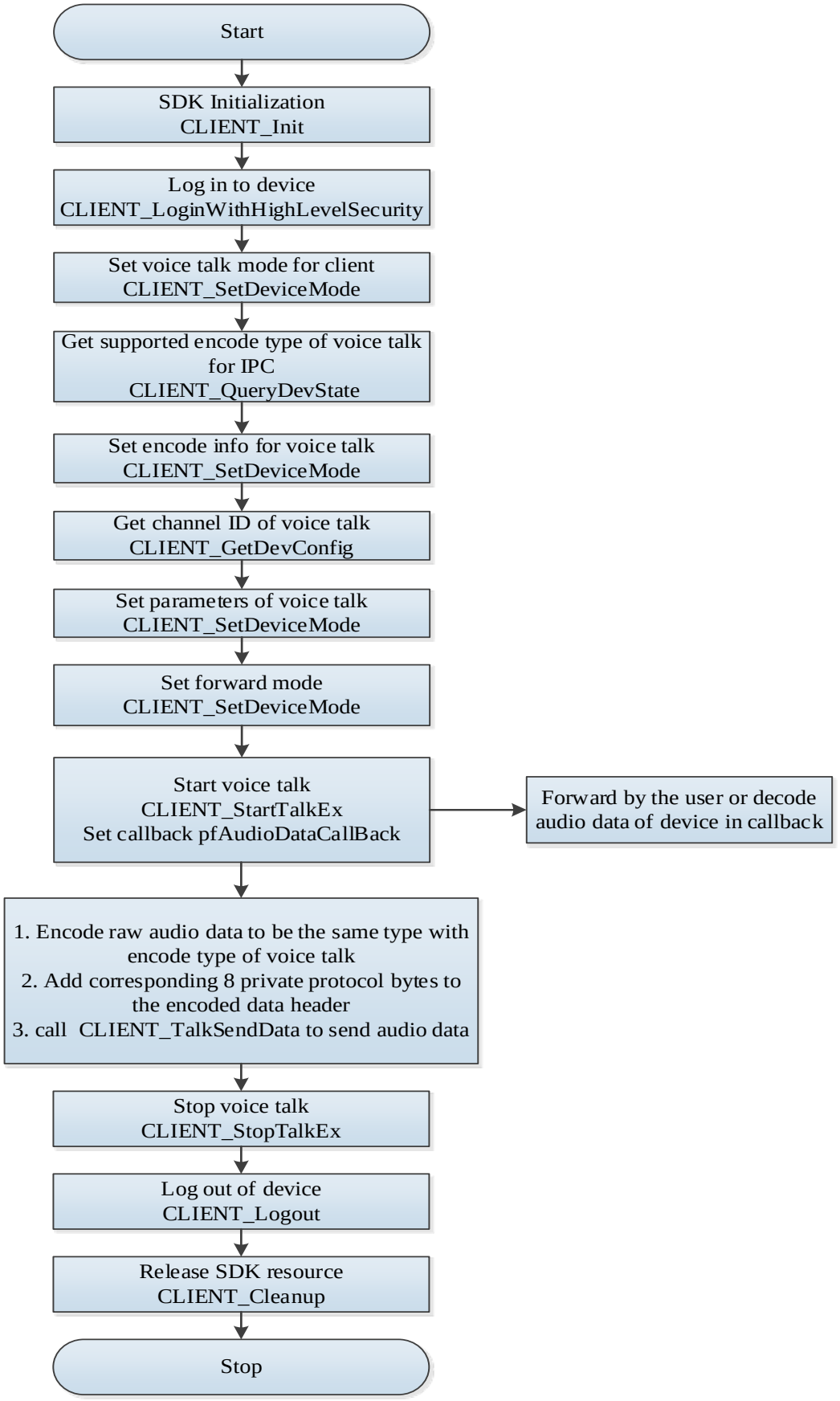
Step 12 Call **CLIENT_StopTalkEx** to stop voice talk.

Step 13 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 14 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.7.3.2 Server Mode

Figure 2-11 Process of server mode



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call **CLIENT_SetDeviceMode** to set voice talk mode as server mode, and set the parameter emType as DH_TALK_SERVER_MODE.

Step 4 Call **CLIENT_QueryDevState** to get aupported voice talk encoding type list, and set the parameter nType as DH_DEVSTATE_TALK_ECTYPE.

Step 5 Call **CLIENT_SetDeviceMode** to set voice talk decoding info and set the parameter emType as DH_TALK_ENCODE_TYPE.

Step 6 Call **CLIENT_GetDevConfig** to get voice talk channel number and set parameter dwCommand as DH_DEV_DEVICECFG. If the acquired channel number is o, use o channel by default.

Step 7 Call **CLIENT_SetDeviceMode** to set voice talk parameter and set the parameter emType as DH_TALK_SPEAK_PARAM.

Step 8 Call **CLIENT_SetDeviceMode** to set voice talk transfer mode. No-transfer mode is to implement voice talk between local PC and logined device; and transfer mode is to implement voice talk between local PC and front-end device connected with specific channel of the logined device.

Step 9 Call **CLIENT_StartTalkEx** to set callback function and start voice talk.In callback function, users can process audio data which is sent from device by themselves, such as transfer or decding for palying.

Step 10 Users decode original audio data to be the same type with talk encoding type, then add 8 corresponding private protocol bytes in front of encoded data, and call **CLIENT_TalkSendData** to send audio data to device.

Step 11 After voice talk is finished, call **CLIENT_RecordStopEx** to stop PC recording.

Step 12 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 13 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

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 g_lLoginHandle = 0L;
static LLONG g_lTalkHandle = 0L;
static BOOL g_bRecordFlag = FALSE;
```

```
static char g_szDevIp[32] = "172.23.2.66";
static WORD g_nPort = 37777; // Tcp connection port, which should be the same as tcp connection port of expected login device.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//**************************************************************************
// 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 is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in this function.
// Set the callback function by CLIENT_SetAutoReconnect. When offline device is reconnected successfully, SDK will call the function.
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

// Voice talk data callback function
// It is not recommended to call SDK interface in this 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 lTalkHandle, 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 information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
```

```
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header file is decimal. Take care of conversion.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23) // Do not support this function. The corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
        printf("\n");
    }
}

void RunTest()
{

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

```
    }

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // Set as voice talk 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 voice talk encoding 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 voice talk decoding info
    DHDEV_TALKDECODE_INFO curTalkMode;
    // Select the first encode method in the list, and users can select other encode method asneeded.
    curTalkMode = stulstTalkEncode.type[0];
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, 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;
    }

    // Get voice talk channel number of device
    DWORD dwRetBytes = 0;
    DHDEV_SYSTEM_ATTR_CFG stuAttr = { sizeof(stuAttr) };
    if (FALSE == CLIENT_GetDevConfig(g_lLoginHandle, DH_DEV_DEVICECFG, -1, &stuAttr, stuAttr.dwSize,
&dwRetBytes, 3000))
    {
        printf("CLIENT_GetDevConfig cmd[%d] Failed!Last Error[%x]\n" , DH_DEV_DEVICECFG,
CLIENT_GetLastError());
        return;
    }
    // Set voice talk parameter.
    NET_SPEAK_PARAM stuSpeak = {sizeof(stuSpeak)};
    stuSpeak.nMode = 0; // 0: talk（default mode）; 1: shout, reset when switch from shout to intercom.

    // Even if partial devices support voice talk, the returned channel number can be 0.
    // When stuAttr.byTalkOutChanNum is 0, the talk channel number is 0; otherwise, the range is 0 -
(stuAttr.byTalkOutChanNum-1).
    if (0 == stuAttr.byTalkOutChanNum)
    {
        stuSpeak.nSpeakerChannel = 0; // Voice talk channel number
    }
    else
    {
       // The example code select channel number as stuAttr.byTalkOutChanNum-1 to implement voice talk,
and userds can select the neededvalue from range 0 - (stuAttr.byTalkOutChanNum-1).
        stuSpeak.nSpeakerChannel = stuAttr.byTalkOutChanNum-1; // Voice talk channel number
    }
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, 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 transfer mode
    NET_TALK_TRANSFER_PARAM stuTransfer = {sizeof(stuTransfer)};
    stuTransfer.bTransfer = FALSE; // Close transfer mode because it is voice talk with logindevices.
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, 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_lTalkHandle = CLIENT_StartTalkEx(g_lLoginHandle, AudioDataCallBack, (DWORD)NULL);
    if(0 != g_lTalkHandle)
    {
        // Start local recording.It is no need to call this interface if it is one-way voice talk between DVR and
PC.
        BOOL bSuccess = CLIENT_RecordStartEx(g_lLoginHandle);
        if(TRUE == bSuccess)
        {
            g_bRecordFlag = TRUE;
        }
        else
        {
            if (FALSE == CLIENT_StopTalkEx(g_lTalkHandle))
            {
                printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
            }
            else
            {
                g_lTalkHandle = 0;
```

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

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

```
                g_lTalkHandle = 0;
            }
        }
        // Log out of 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 initialization resources
        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 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("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 AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag,
DWORD dwUser)
{
	// If more than one playbacks or downloads use the progress callback, users can do one-to-one
correspondence by lTalkHandle.
	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
    }
}
```

2.7.4.2 Server Mode

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

```
static LLONG    g_nPlayPort = 0;


#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; // Tcp connection port, which should be the same as tcp connection port of expected login device.
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 disconnected.
// It is not recommended to call SDK interface in the SDK callback function.
// The callback is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in this function.
// Set the callback function by CLIENT_SetAutoReconnect. When offline device is reconnected successfully, SDK will call the function.
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);


// Voice talk data callback function
// Only audio data sent by device can be received in server mode
// It is not recommended to call SDK interface in this 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 lTalkHandle, 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 voice talk 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 information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);
```

```
    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
```

```
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header file is decimal. Take care of conversion.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23) // Do not support this function. The corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
        printf("\n");
    }
}

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

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // Set as voice talk server mode.
    BOOL bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, 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 voice talk encoding 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 voice talk decoding info
    g_curTalkMode.encodeType = stulstTalkEncode.type[0];
    bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, 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;
    }

    // Get voice talk channel number of device
    DWORD dwRetBytes = 0;
    DHDEV_SYSTEM_ATTR_CFG stuAttr = { sizeof(stuAttr) };
    if (FALSE == CLIENT_GetDevConfig(g_lLoginHandle, DH_DEV_DEVICECFG, -1, &stuAttr, stuAttr.dwSize,
&dwRetBytes, 3000))
    {
        printf("CLIENT_GetDevConfig cmd[%d] Failed!Last Error[%x]\n" , DH_DEV_DEVICECFG,
CLIENT_GetLastError());
        return;
    }
```

```
// Set voice talk parameter.
NET_SPEAK_PARAM stuSpeak = {sizeof(stuSpeak)};
stuSpeak.nMode = 0; // 0: talk（default mode）; 1: shout, reset when switch from shout to intercom.

// Even if partial devices support voice talk, the returned channel number can be 0.
// When stuAttr.byTalkOutChanNum is 0, the talk channel number is 0; otherwise, the range is 0 - (stuAttr.byTalkOutChanNum-1).
if (0 == stuAttr.byTalkOutChanNum)
{
    stuSpeak.nSpeakerChannel = 0; // Voice talk channel number
}
else
{
    // The example code select channel number as stuAttr.byTalkOutChanNum-1 to implement voice talk, and userds can select the neededvalue from range 0 - (stuAttr.byTalkOutChanNum-1).
    stuSpeak.nSpeakerChannel = stuAttr.byTalkOutChanNum-1; // Voice talk channel number
}
bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, 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 transfer mode
NET_TALK_TRANSFER_PARAM stuTransfer = {sizeof(stuTransfer)};
stuTransfer.bTransfer = FALSE; // Close transfer mode because it is voice talk with logindevices.
bSuccess = CLIENT_SetDeviceMode(g_lLoginHandle, 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_lTalkHandle = CLIENT_StartTalkEx(g_lLoginHandle, AudioDataCallBack, (DWORD)NULL);
    if(0 != g_lTalkHandle)
    {
        bSuccess = StartAudioRecord();
        if(TRUE == bSuccess)
        {
            g_bOpenAudioRecord = TRUE;
        }
        else
        {
            printf("StartAudioRecord Failed!\n");
            CLIENT_StopTalkEx(g_lTalkHandle);
            g_lTalkHandle = 0;
        }
    }
    else
    {
        printf("CLIENT_StartTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
    }
}

void EndTest()
{
    printf("input any key to quit!\n");
    getchar();
    // Clean up talk resources of Dahua encoding/decodinglibrary.
    if(TRUE == g_bOpenAudioRecord)
    {
        if (TRUE == StopAudioRecord())
        {
            g_bOpenAudioRecord = FALSE;
        }
    }
```

```
    // Stop voice talk
    if (0 != g_lTalkHandle)
    {
        if(FALSE == CLIENT_StopTalkEx(g_lTalkHandle))
        {
            printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lTalkHandle = 0;
        }
    }
    // Log out of 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 initialization resources
    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 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("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 AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag,
DWORD dwUser)
{
    // If more than one playbacks or downloads use the progress callback, users can do one-to-one
correspondence by lTalkHandle.
    if (g_lTalkHandle != lTalkHandle)
    {
        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.
        PLAY_GetFreePort(&g_nPlayPort);
        //For PCM format withour header , please add 128.
        if (g_curTalkMode.encodeType == DH_TALK_DEFAULT)
        {
            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 formats.
        PLAY_InputData(g_nPlayPort,(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_lTalkHandle, (char *)pCbData, iCbLen);

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

```
//*****************************************************************************
BOOL StartAudioRecord()
{
    // It is the characteristics of Dahua encoding/decoding library.
    PLAY_GetFreePort(&g_nPlayPort);
    //    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 audiolength.
    {
        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(g_nPlayPort,0,0,1024*900);
    if(bOpenRet)
    {
        BOOL bPlayRet = PLAY_Play(g_nPlayPort,0);
        if(bPlayRet)
        {
            PLAY_PlaySoundShare(g_nPlayPort);
            BOOL bSuccess = PLAY_OpenAudioRecord(AudioCallFunction,g_curTalkMode.nAudioBit,
                        g_curTalkMode.dwSampleRate,nFrameLength,0,NULL);
            if(bSuccess)
            {
                bRet = TRUE;
            }
            else
            {
                PLAY_StopSoundShare(g_nPlayPort);
                PLAY_Stop(g_nPlayPort);
                PLAY_CloseStream(g_nPlayPort);
            }
        }
        else
        {
            PLAY_CloseStream(g_nPlayPort);
        }
    }

    return bRet;
}
```

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

    return bSuccess;
}
```

2.8 Video Snapshot

2.8.1 Introduction

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

Snapshot picture from device: Users 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 users, and users can configure interface by SDK to set some parameters, such as picture encoding type and resolution.

2.8.2 Interface Overview

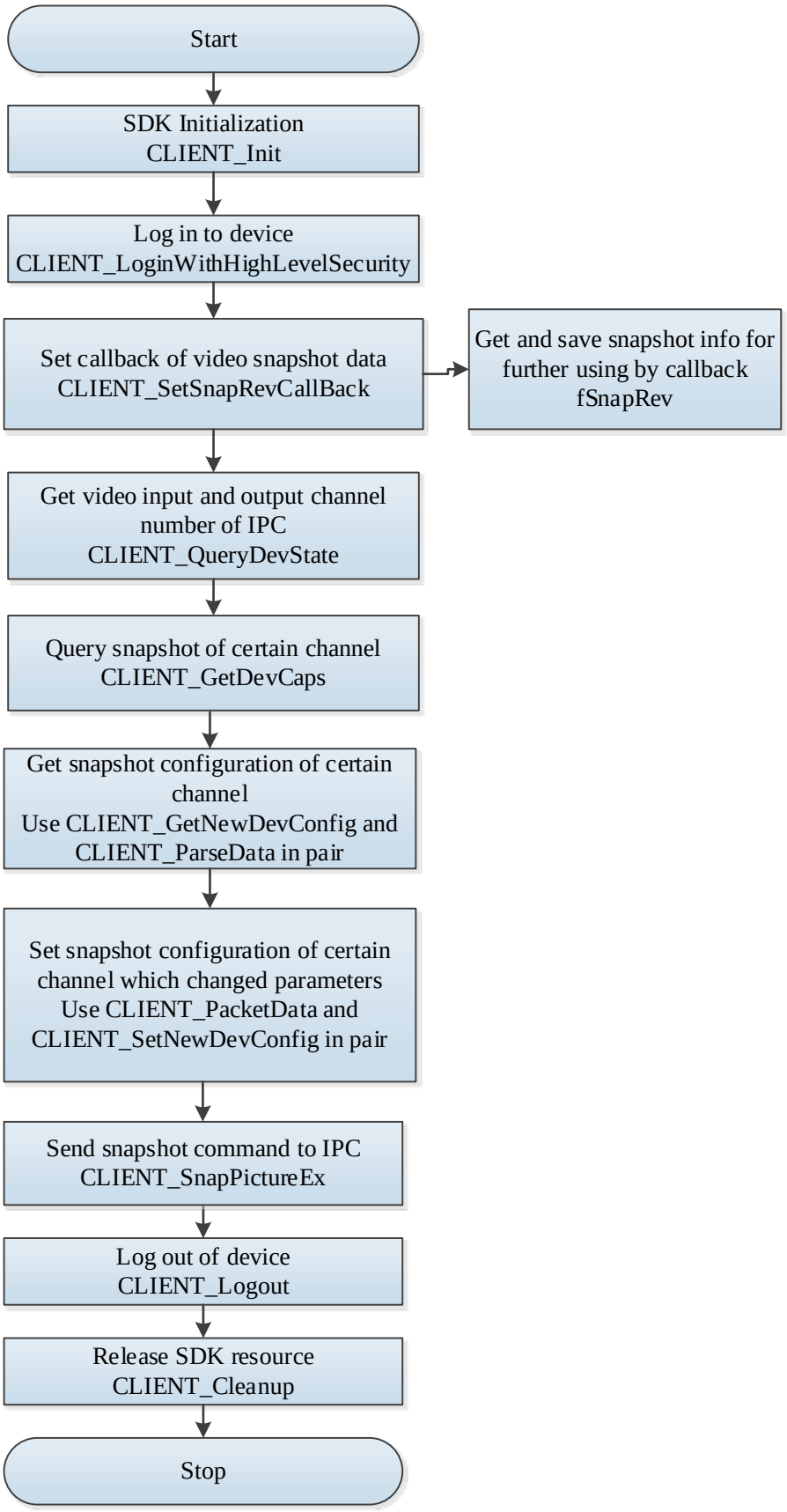
Table 2-8 Interfaces of video snapshot

Interface	Implication
CLIENT_Init	Interface for SDK Initialization.
CLIENT_Cleanup	Interface for cleaning up SDK resources.
CLIENT_LoginWithHighLevelSecurity	Login with high level security.
CLIENT_QueryDevState	Interface for querying device status.

Interface	Implication
CLIENT_GetDevCaps	Interface for getting device caplicity.
CLIENT_GetNewDevConfig	Interface for getting new device configurations.
CLIENT_ParseData	Interface for analyzing the acquired configuration info.
CLIENT_PacketData	Interface for packeting the set configuration info.
CLIENT_SetNewDevConfig	Inferface for setting new device configuration.
CLIENT_SnapPictureEx	Extensive interface for snapshot request.
CLIENT_Logout	Interface for logout device.
CLIENT_GetLastError	Interface for getting error code after failed calling interface.

2.8.3 Process

Figure 2-12 Process of video snapshot



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 **Call CLIENT_SetSnapRevCallBack** to set snapshot callback function. SDK will call fSnapRev callback function to recall picture information and data to users, when SDK receives snapshot data sent from device,

Step 4 Call **CLIENT_GetDevCaps** and set the cprresponding type parameter as NET_SNAP_CFG_CAPS, to query for the snapshot capalicity of secified channel.

Step 5 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData**, and set the cprresponding type parameter as CFG_CMD_ENCODE, to get the snapshot configuration of secified channel.

Step 6 Change the corresponding snapshot configuration, and then Call **CLIENT_PacketData** and **CLIENT_SetNewDevConfig**. Then set the cprresponding parameter type as CFG_CMD_ENCODE, to set the snapshot configuration of secified channel.

Step 7 Call **CLIENT_SnapPictureEx** to send snapshot command to the front–end devices, and wait for devices to reply picture information in fSnapRev callback.

Step 8 Call **CLIENT_Logout** to log out of the device.

Step 9 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resources.

2.8.4 Example Code

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

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

static BOOL g_bNetSDKInitFlag = FALSE;
static LLONG g_lLoginHandle = 0L;
static char g_szDevIp[32] = "172.23.1.27";
static WORD g_nPort = 37777; // Tcp connection port, which should be the same as tcp connection port of expected login device.
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 disconnected.
// It is not recommended to call SDK interface in the SDK callback function.
// The callback is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in this function.
// Set the callback function by CLIENT_SetAutoReconnect. When offline device is reconnected successfully, SDK will call the function.
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);

//Snapshot callback function.
// It is not recommended to call SDK interfaces in this callback.
//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 lLoginID, BYTE *pBuf, UINT RevLen, UINT EncodeType, DWORD CmdSerial, LDWORD dwUser);

//*************************************************************************************
// Commonly used funvtion det 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 the SDK version information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
```

```
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if(0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header file is decimal. Take care of conversion.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23) // Do not support this function. The corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
        printf("\n");
    }
}


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

    if (0 == g_lLoginHandle)
    {
        return;
    }

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

    int nChannelId = 0;
```

```
    unsigned int i = 0;
    unsigned int nRealNum = 0;
    // Get the front-end video input/output channel number.
    NET_DEV_CHN_COUNT_INFO stuChnCountInfo = {sizeof(stuChnCountInfo)};
    stuChnCountInfo.stuVideoIn.dwSize = sizeof(stuChnCountInfo.stuVideoIn);
    stuChnCountInfo.stuVideoOut.dwSize = sizeof(stuChnCountInfo.stuVideoOut);
    int nRetLen = 0;
    int nRet = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_CHN_COUNT, (char
*)&stuChnCountInfo, sizeof(NET_DEV_CHN_COUNT_INFO), &nRetLen);
    if(nRet == FALSE || nRetLen != sizeof(NET_DEV_CHN_COUNT_INFO))
    {
        printf("CLIENT_QueryDevState cmd[DH_DEVSTATE_DEV_CHN_COUNT] Failed!Last Error[%x]\n",
CLIENT_GetLastError());
        return;
    }

    char szUserChoose[128] = "";
    do
    {
        printf("Select snapshot channel (%d-%d)\n", 0, stuChnCountInfo.stuVideoIn.nMaxLocal-1);
        int nError = 0;
        unsigned int nTmp = GetIntInput("\t Select: ", nError);
        if (0 != nError || nTmp >= stuChnCountInfo.stuVideoIn.nMaxLocal)
        {
            printf("Inout error !  \n");
            continue;
        }
        unsigned int nSnapChannelId = nTmp;

        // Query for the snapshot capacility of specified channel.
        NET_IN_SNAP_CFG_CAPS stuSnapCapInParam = {0};
        stuSnapCapInParam.nChannelId = nSnapChannelId;
        NET_OUT_SNAP_CFG_CAPS stuSnapCapOutParam = {0};
        if (FALSE == CLIENT_GetDevCaps(g_lLoginHandle, NET_SNAP_CFG_CAPS, &stuSnapCapInParam,
&stuSnapCapOutParam, 5000))
        {
            printf("CLIENT_GetDevCaps cmd[NET_SNAP_CFG_CAPS] Failed!Last Error[%x]\n",
CLIENT_GetLastError());
            return;
        }
```

```
        // Get the snapshot configuration of specified channel.
        char * pszSnapAttr = new char[1024*100];
        if (NULL == pszSnapAttr)
        {
            printf("pszSnapAttr new fail!\n");
            return;
        }
        memset(pszSnapAttr, 0, 1024*100);
        DWORD dwRetLen = 0;
        if(FALSE == CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ENCODE, nSnapChannelId,
pszSnapAttr, 1024*100, NULL, 5000))
        {
            printf("CLIENT_GetNewDevConfig cmd[CFG_CMD_ENCODE] Failed!Last Error[%x]\n",
CLIENT_GetLastError());
            delete []pszSnapAttr;
            return;
        }

        CFG_ENCODE_INFO stuEncodeInfo = {0};
        if(FALSE == CLIENT_ParseData(CFG_CMD_ENCODE, pszSnapAttr, (LPVOID)&stuEncodeInfo,
sizeof(CFG_ENCODE_INFO), NULL))
        {
            printf("CLIENT_ParseData cmd[CFG_CMD_ENCODE] Failed!Last Error[%x]\n",
CLIENT_GetLastError());
            delete []pszSnapAttr;
            return;
        }
        delete []pszSnapAttr;
        pszSnapAttr = NULL;

        nTmp = GetIntInput("Select anpshot method: \n\t0 manuallysnapshot \n\t1 Snapshot by time \n\t
Select:", nError);
        if (0 != nError || nTmp >= 2)
        {
            printf("Input error !  \n");
            continue;
        }

        // Change the snapshot configuration of specified channel.
```

```
        unsigned int nSnapType = nTmp;
        if (1 == nTmp)
        {
            stuEncodeInfo.stuSnapFormat[0].abSnapEnable = true;
            stuEncodeInfo.stuSnapFormat[0].bSnapEnable = TRUE;
            printf("Support snapshot interval:\n");
            nRealNum = min(stuSnapCapOutParam.dwFramesPerSecNum, DH_MAX_FPS_NUM);
            for (i = 0; i < nRealNum; ++i)
            {
                if (stuSnapCapOutParam.nFramesPerSecList[i] < 0)
                {
                    printf("\t[%2d]: [%d]Second of one frame \n", i,
abs(stuSnapCapOutParam.nFramesPerSecList[i]));
                }
                else
                {
                    printf("\t[%2d]: [%d]Frame of one second \n", i,
stuSnapCapOutParam.nFramesPerSecList[i]);
                }
            }
            nTmp = GetIntInput("\t Select:", nError);
            if (0 != nError || nTmp >= nRealNum)
            {
                printf("Input error !  \n");
                continue;
            }

            double dbFps = 0;
            if (stuSnapCapOutParam.nFramesPerSecList[nTmp] >= 0)
            {
                dbFps = stuSnapCapOutParam.nFramesPerSecList[nTmp];
            }
            else
            {
                dbFps = 1 / (double)(0-stuSnapCapOutParam.nFramesPerSecList[nTmp]);
            }
            stuEncodeInfo.stuSnapFormat[0].stuVideoFormat.nFrameRate = (float)dbFps;
        }

        printf("Supported resolution:\n");
```

```
        nRealNum = min(stuSnapCapOutParam.nResolutionTypeNum, DH_MAX_CAPTURE_SIZE_NUM);
        for (i = 0; i < nRealNum; ++i)
        {
            printf("\t[%2d]:[%dx%d]\n", i, stuSnapCapOutParam.stuResolutionTypes[i].snWidth,
stuSnapCapOutParam.stuResolutionTypes[i].snHight);
        }
        nTmp = GetIntInput("\t Select:", nError);
        if (0 != nError || nTmp >= nRealNum)
        {
            printf("Input error !  \n");
            continue;
        }

        // Set the related snapshot configuration
        stuEncodeInfo.stuSnapFormat[0].stuVideoFormat.nWidth =
stuSnapCapOutParam.stuResolutionTypes[nTmp].snWidth;
        stuEncodeInfo.stuSnapFormat[0].stuVideoFormat.nHeight =
stuSnapCapOutParam.stuResolutionTypes[nTmp].snHight;

        printf("Supported image quality (higer value, higer quality) :\n");
        nRealNum = min(stuSnapCapOutParam.dwQualityMun, DH_MAX_QUALITY_NUM);
        for (i = 0; i < nRealNum; ++i)
        {
            printf("\t[%2d]:quality level[%d]\n", i, stuSnapCapOutParam.nQualityList[i]);
        }
        nTmp = GetIntInput("\t Select:", nError);
        if (0 != nError || nTmp >= nRealNum)
        {
            printf("Input error !  \n");
            continue;
        }

        stuEncodeInfo.stuSnapFormat[0].stuVideoFormat.emImageQuality =
(CFG_IMAGE_QUALITY)stuSnapCapOutParam.nQualityList[nTmp];

        // Set snapshot configuration of specified configuration
        if (NULL == pszSnapAttr)
        {
            pszSnapAttr = new char[1024*100];
            if (NULL == pszSnapAttr)
```

```
        {
            printf("pszSnapAttr new fail!\n");
            return;
        }
    }

    memset(pszSnapAttr, 0, 1024*100);
    if (FALSE == CLIENT_PacketData(CFG_CMD_ENCODE, &stuEncodeInfo, sizeof(CFG_ENCODE_INFO),
pszSnapAttr, 1024*100))
    {
        printf("CLIENT_PacketData cmd[CFG_CMD_ENCODE] Failed!Last Error[%x]\n",
CLIENT_GetLastError());
        delete []pszSnapAttr;
        return;
    }

    int nRestart = 0;
    if (FALSE == CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ENCODE, nSnapChannelId,
pszSnapAttr, 1024*100, &nError, &nRestart, 3000))
    {
        printf("CLIENT_SetNewDevConfig cmd[CFG_CMD_ENCODE] Failed!Last Error[%x]\n",
CLIENT_GetLastError());
        delete []pszSnapAttr;
        return;
    }

    delete []pszSnapAttr;
    pszSnapAttr = NULL;

    //Send snapshot command to the front-end device
    SNAP_PARAMS stuSnapParams;
    stuSnapParams.Channel = nChannelId;
    stuSnapParams.mode = nSnapType;
    stuSnapParams.CmdSerial = ++g_nCmdSerial; // Ask for SN. The valid range is 0~65535, and the over
range part will be cut off as unsigned short.
    if (FALSE == CLIENT_SnapPictureEx(g_lLoginHandle, &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();

        // Log ou t of 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 initialization resources
        if (TRUE == g_bNetSDKInitFlag)
        {
            CLIENT_Cleanup();
            g_bNetSDKInitFlag = FALSE;
        }

        exit(0);
}

int main()
{
```

```
	InitTest();

	RunTest();

	EndTest();

	return 0;
}

//****************************************************************************
// Commonly used callback set 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 lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
	printf("Call HaveReConnect\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 SnapRev(LLONG lLoginID, BYTE *pBuf, UINT RevLen, UINT EncodeType, DWORD CmdSerial,
LDWORD dwUser)
```

```
{
    printf("[SnapRev] -- receive data!\n");
    if(lLoginID == g_lLoginHandle)
    {
        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 callback function definition
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)
    {
        // Inpu t parameter error
        nError = -1;
    }
    else
    {
        nError = 0;
    }

    return nGet;
}

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

2.9 Alarm Report

2.9.1 Introduction

Alarm report, is to send alarm to platform-end and notify the platform, when front-end device detects special event set previously.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

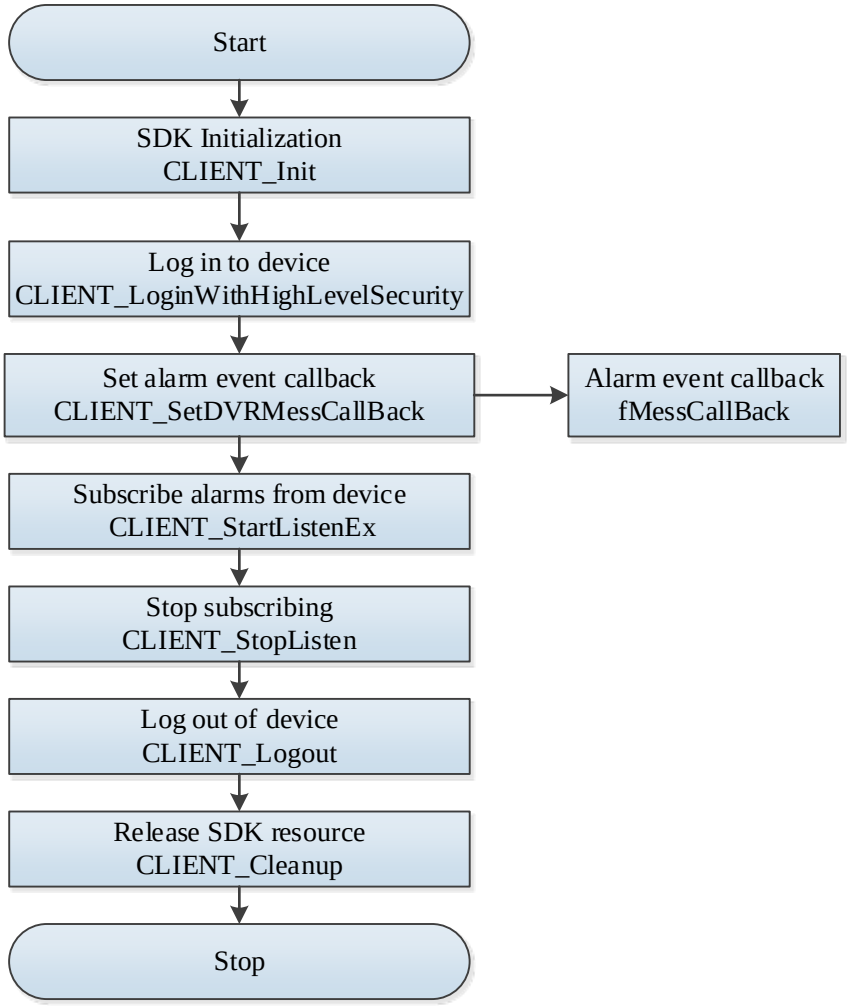
Table 2-9 Interfaces of alarm listening

Interface	Implication
CLIENT_Init	Interface for initilization.
CLIENT_Cleanup	Interface for cleaning up SDK resources.
CLIENT_LoginWithHighLevelSecurity	Login with high level security.
CLIENT_SetDVRMessCallBack	Interface for setting alarm callback function.
CLIENT_StartListenEx	Extensive interface for subscribing alarm event from device.
CLIENT_StopListen	Interface for stopping subscribing alarm.
CLIENT_Logout	Interface for logout device.

Interface	Implication
CLIENT_GetLastError	Interface for getting error code after failed calling.

2.9.3 Process

Figure 2-13 Process of alarm report



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call **CLIENT_SetDVRMessCallBack** to set alarm callback function which should be called before alarm subscription.

Step 4 Call **CLIENT_StartListenE** to subscribe alarms fro mdevice. After susbcribtion, alarm event reported by device is sent to user via callback function set in **CLIENT_SetDVRMessCallBack**.

Step 5 After using the function module, Call **CLIENT_StopListen** to stop susbscbing alarm from device.

Step 6 Call **CLIENT_Logout** to log out of the device.

Step 7 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

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; // Tcp connection port, which should be the same as tcp connection port of expected login device.
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 disconnected.
// It is not recommended to call SDK interface in the SDK callback function.
// The callback is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in this function.
// Set the callback function by 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 is 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 lCommand, LLONG lLoginID, 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 the SDK version information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
```

```
    CLIENT_SetNetworkParam(&stuNetParm);


    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);


        if(0 == g_lLoginHandle)
        {
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header
file is decimal. Take care of conversion.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23) // Do not support this function. The corresponding
error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp ,
g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business
function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
        printf("\n");
```

```
    }
}

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

    if (0 == g_lLoginHandle)
    {
        return;
    }
    // Set alarm event callback
    CLIENT_SetDVRMessCallBack(MessCallBack , NULL);

    // Subscribe alarm fro m device
    if( TRUE == CLIENT_StartListenEx(g_lLoginHandle))
    {
        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 fro m device
```

```
    if (TRUE == g_bStartListenFlag)
    {
        if (FALSE == CLIENT_StopListen(g_lLoginHandle))
        {
            printf("CLIENT_StopListen Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_bStartListenFlag = FALSE;
        }
    }
    // Log ou t of 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 initialization resources
    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 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("lLoginID[0x%x]", lLoginID);
	if (NULL != pchDVRIP)
	{
		printf("pchDVRIP[%s]\n", pchDVRIP);
	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
```

```
}

BOOL CALLBACK MessCallBack(LONG lCommand, LLONG lLoginID, 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(lCommand)
    {
    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 number = [%s]\n" , pInfo->szCardNo);
            }
        }
        break;
    default:
        printf("\n[MessCallBack] – Other alarms Get lCommand = 0x%x\n" , lCommand);
        break;
    }
    return TRUE;
}
```

2.10 Device Search

2.10.1 Introduction

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

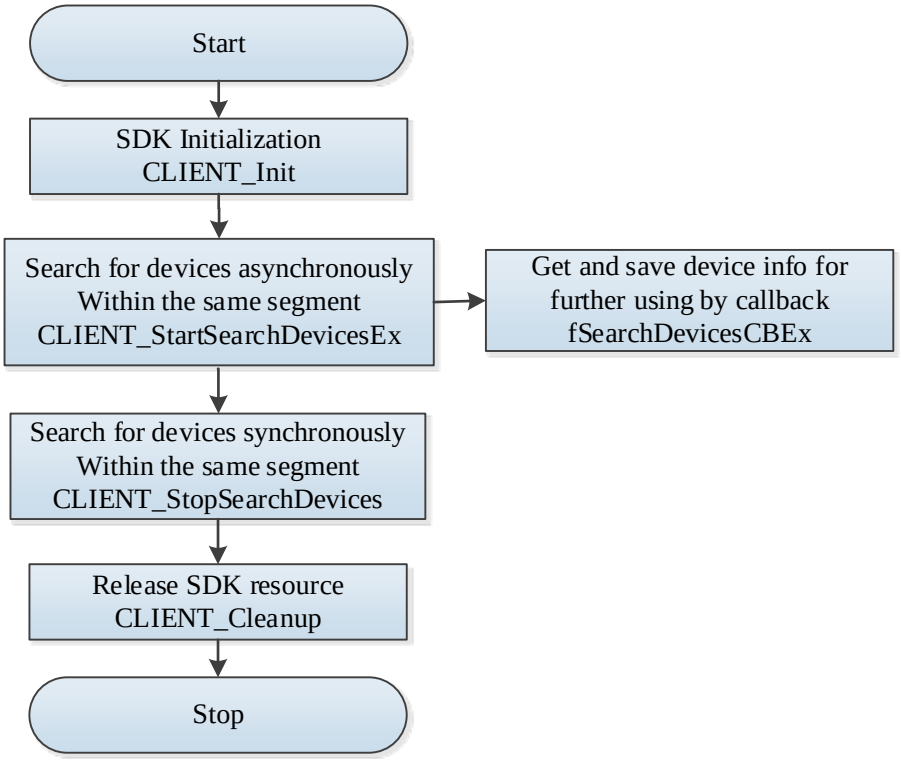
Table 2-10 Interfaces of device search

Interface	Implication
CLIENT_Init	Interface for initilization.
CLIENT_Cleanup	Interface for cleaning up SDK resources.
CLIENT_StartSearchDevicesEx	Interface for async searching for devices within same segment,such as IPC and NVS.
CLIENT_StopSearchDevices	Interface for stopping async search for devices within same segment,such as IPC and NVS.
CLIENT_SearchDevicesByIPs	Interface for sync searching cross-segment devices.
CLIENT_GetLastError	Interface for getting error code after failed calling interface.

2.10.3 Process

2.10.3.1 Async Searching within Same Segment

Figure 2-14 Process of async searching within same segment



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

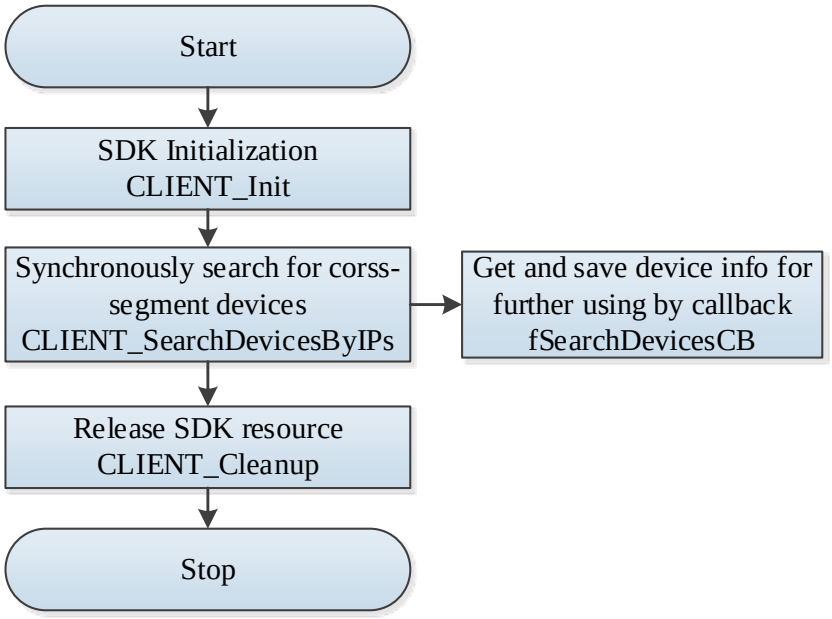
Step 2 Call **CLIENT_StartSearchDevicesEx** to async search for devices within same segment. Users get the obtained device info by fSearchDevicesCB which is set in this interface.The search operation has no timeout, so usera need to stop searching by calling interface **CLIENT_StopSearchDevices**.

Step 3 Call **CLIENT_StopSearchDevices** to stop sync searchingfor devices within same segment.

Step 4 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.10.3.2 Sync Searching in Cross-segment

Figure 2-15 Process of sync searching in cross-segment



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call **CLIENT_SearchDevicesByIPs** to sync search for devices in cross-segment. Users get the obtained device info by fSearchDevicesCB which is set in this interface. Only when searching time is out or searching all the devices cross the segment, the interface return.Users can decide the timeout as needed.

Step 3 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resources.

2.10.4 Example Code

2.10.4.1 Async Searching within Same Segment

```
#include <windows.h>
#include <stdio.h>
#include <vector>
#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::vector<DEVICE_NET_INFO_EX> g_lDeviceVec;          // Device list

//*************************************************************************
// Commonly used callback set.

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

// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in this function.
// Set the callback function by 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 is not recommended to call SDK interfaces in this callback function
// Set this callback function in CLIENT_StartSearchDevices/
CLIENT_StartSearchDevicesEx/CLIENT_SearchDevicesByIPs. SDK will call this function when device is found.
void CALLBACK SearchDevicesCBEx(LLONG lSearchHandle,DEVICE_NET_INFO_EX2 *pDevNetInfo, void* pUserData);

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

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

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);
}

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

```
    }
    // Start async searching within same segment
    NET_IN_STARTSERACH_DEVICE pInBuf = { 0 };
    NET_OUT_STARTSERACH_DEVICE pOutBuf = { 0 };
    LLONG seachHandle = 0;
    pInBuf.dwSize = sizeof(NET_IN_STARTSERACH_DEVICE);
    pInBuf.cbSearchDevices = cbSearchDevicesEx;
    pInBuf.pUserData = this;
    int nMaxCopyLen = MAX_LOCAL_IP_LEN - 1;
    strncpy(pInBuf.szLocalIp, "192.168.1.10", sizeof(pInBuf.szLocalIp) - 1);
    pOutBuf.dwSize = sizeof(NET_OUT_STARTSERACH_DEVICE);

    seachHandle = CLIENT_StartSearchDevicesEx(&pInBuf, &pOutBuf);
    if (NULL == seachHandle)
    {
        printf("CLIENT_StartSearchDevicesEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
        return;
    }
    int nIndex = 0;
    int nSearchTime = 0;
    int nSearchLimit = 10;// Search lasts for 10 seconds, and users can change the value according to network condition
    Sleep(nSearchLimit * 1000);
    EnterCriticalSection(&g_mDeviceListLock);
    for (std::vector<DEVICE_NET_INFO_EX>::iterator iter = g_lDeviceVec.begin(); iter != g_lDeviceVec.end(); ++iter)
    {
        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_lDeviceVec.clear();
    LeaveCriticalSection(&g_mDeviceListLock);
```

```
}

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

    // Stop async searching within same segment
    if (NULL != g_lSearchHandle)
    {
        if (FALSE == CLIENT_StopSearchDevices(g_lSearchHandle))
        {
            printf("CLIENT_StopSearchDevices Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
    }

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

int main()
{
    InitTest();

    RunTest();

    EndTest();
```

```
    return 0;
}

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 lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    printf("Call HaveReConnect\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 SearchDevicesCBEx(LLONG lSearchHandle,DEVICE_NET_INFO_EX2 *pDevNetInfo, void*
pUserData)
{
    if ((NULL == pDevNetInfo) || (NULL == pUserData))
    {
        printf("warming param is null\n");
```

```
        return;
    }

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

2.10.4.2 Sync Searching in Cross-segment

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

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

BOOL g_bNetSDKInitFlag = FALSE;
std::vector<DEVICE_NET_INFO_EX> g_lDeviceVec;        // Device list

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

//*****************************************************************************
// 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 is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in this function.
```

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

// Sync search for device callback
// It is not recommended to call SDK interfaces in this callback function.
// Set this callback function in CLIENT_StartSearchDevices/ CLIENT_StartSearchDevicesEx /CLIENT_SearchDevicesByIPs. SDK will call this function when device is found.
void CALLBACK SearchDevicesCBEx(LLONG lSearchHandle,DEVICE_NET_INFO_EX2 *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 the SDK version information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);
}

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

    DEVICE_IP_SEARCH_INFO stuTmp = {sizeof(stuTmp)};
    stuTmp.nIpNum = 256;// Number of valid searched IP address
    for (unsigned int i = 0; i < stuTmp.nIpNum; ++i)
    {
        // Users need to guarantee the validity of IP address
        _snprintf(stuTmp.szIP[i], sizeof(stuTmp.szIP[i]) - 1, "172.11.1.%d", i);
    }

    DWORD dwWaitTime = 5000;
    // Only when searching time is out, the interface return.Users can decide the timeout as needed.
    if (FALSE == CLIENT_SearchDevicesByIPs(&stuTmp, SearchDevicesCB, (LDWORD)&g_lDeviceVec, szLocalIp,
dwWaitTime))
```

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

    int nIndex = 0;
    for (std::vector<DEVICE_NET_INFO_EX>::iterator iter = g_lDeviceVec.begin(); iter != g_lDeviceVec.end();
++iter)
    {
        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_lDeviceVec.clear();
}

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

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

int main()
{
    InitTest();
```

```
	RunTest();

	EndTest();

	return 0;
}

//****************************************************************************
// Commonly used callback set 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("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 SearchDevicesCBEx(LLONG lSearchHandle,DEVICE_NET_INFO_EX2 *pDevNetInfo, void*
pUserData)
{
	if(pDevNetInfo != NULL)
	{
	CDevInitDlg *dlg = (CDevInitDlg *)pUserData;
	DEVICE_NET_INFO_EX2 *pData = NEW DEVICE_NET_INFO_EX2;
	memcpy(pData, pDevNetInfo, sizeof(DEVICE_NET_INFO_EX2));
	LONG bIsUnicast = dlg->m_IsUnicast;
	}
}

// ************************************* Get Local IP interface
std::string GetLocalIpAddress()
{
	WSADATA wsaData;
	if (0 != WSAStartup(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 Smart Event Report and Snapshot

2.11.1 Introduction

Smart event report: Devices make smart analysis by real-time stream. Devices judge whether to report events and to send pictures to users according to event trigger rules configured by users. Smart events include 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 and so on.

Smar tevent snapshot: Users manually send a command to device after subscribing event successfully. Device snapshots pictute of current scene and reports it to users by smart event.

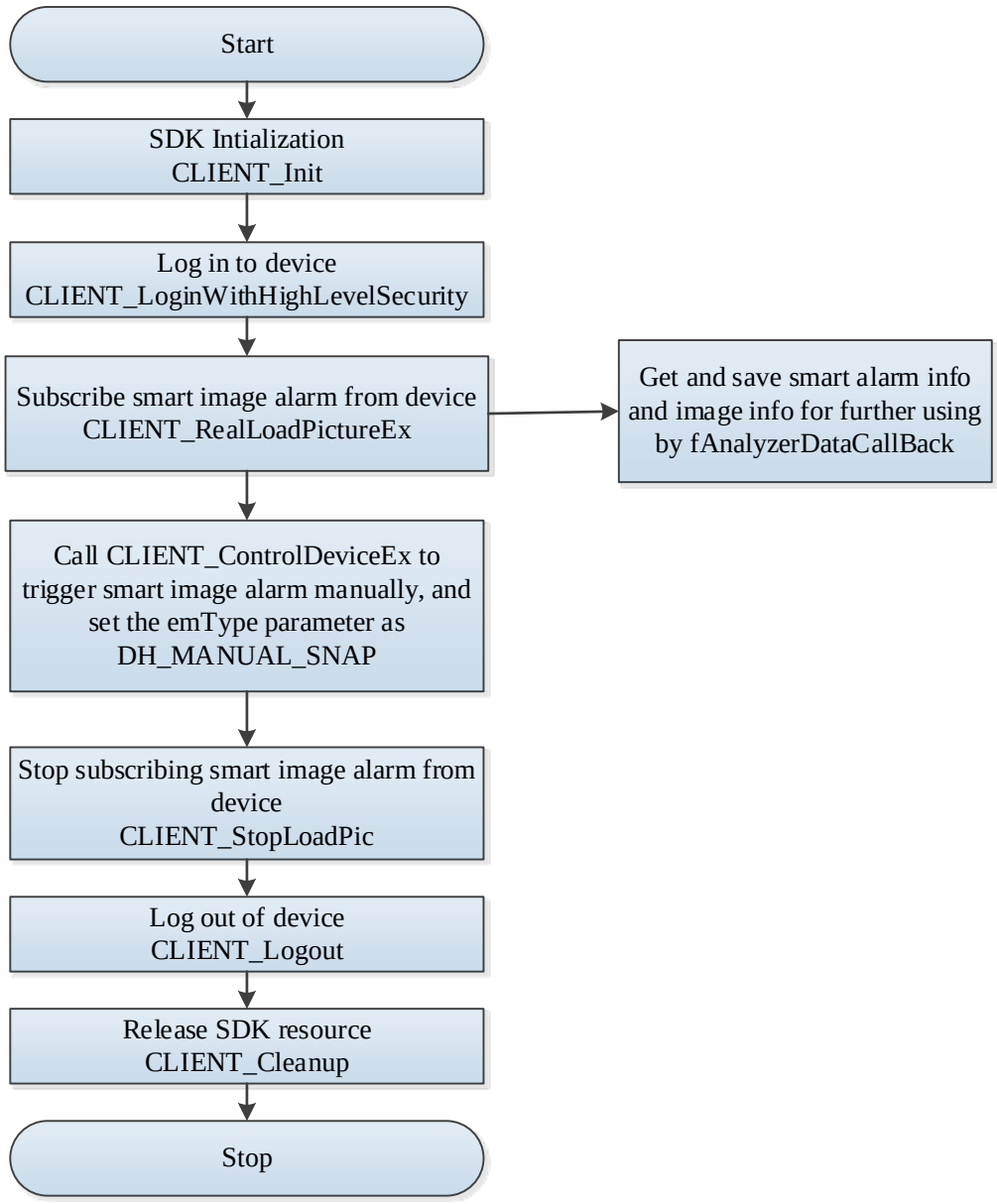
2.11.2 Interface Overview

Table 2-11 Interfaces of smart event report and snapshot

Interface	Implication
CLIENT_Init	Interface for initilization.
CLIENT_Cleanup	Interface for cleaning up SDK resources.
CLIENT_LoginWithHighLevelSecurity	Extensive interface 2 for sync login.
CLIENT_RealLoadPictureEx	Interface for smart snapshot alarm subscription.
CLIENT_ControlDeviceEx	Extensive interface for device control.
CLIENT_Logout	Interface for logout device.
CLIENT_GetLastError	Interface for getting error code after failed calling interface.

2.11.3 Process

Figure 2-16 Process of smart event report and snapshot



Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call **CLIENT_RealLoadPictureEx** to subscribe smart snapshot alarm from device. After successful subscription, the smart snapshot alarm event reported by device will be sent to users by fAnalyzerDataCallBack.In callback function, users should converts input character to corresponding structyure according to the instructions in SDK header files, and then display and save event as needed. Due to SDK receving buffer is 2M by default, when callback picture info exceed 2M, users need to call **CLIENT_SetNetworkParam** to set receiving buffer again, otherwise SDK will abandon data pack over 2M.

Step 4 If users want to manually trigger smart snapshot alarm, call **CLIENT_ControlDeviceEx** with parameter emType DH_MANUAL_SNAP. SDK will send command to device, and then device snapshots current monitoring video and reports it to users.

Step 5 Call **CLIENT_StopLoadPic** to to stop subscribing smart snapshot alarm from device**.**

Step 6 After using the function module, call **CLIENT_Logout** to log out of the device.

Step 7 After using all SDK functions, 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; // Tcp connection port, which should be the same as tcp connection port of expected login device.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin";

//************************************************************************************
// 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 is set by CLIENT_Init. When the device is offline, SDK will call this callback function.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);

// Callback function is set after the device reconnected successfully
// It is not recommended to call SDK interface in this function.
// Set the callback function by CLIENT_SetAutoReconnect. When offline device is reconnected successfully, SDK will call the function.
```

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


// Smart analyzing data callback
// It is not recommended to call SDK interfaces in this callback function.
// Set this callback function in CLIENT_RealLoadPictureEx/CLIENT_RealLoadPicture, and SDK will call this function when device-end has smart snapshot 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, without special meaning.
// Due to SDK receving buffer is 2M by default, when callback snapshot info exceeds 2M, users need to call CLIENT_SetNetworkParam interface to set receiving buffer again, otherwise SDK will abandon data pack over 2M.
int    CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, 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 the SDK version information
    // Optional operation
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);
```

```
    // Set reconnection callback. Internal SDK auto connects when the device disconnected.
    // This operation is optional but recommended.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

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

    // Set more network parameters. The nWaittime and nConnectTryNum of NET_PARAM have the same
    meaning with the timeout and trial time set in interface CLIENT_SetConnectTime.
    // Optional operation
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // The timeout of connection when login.
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, csIp.GetBuffer(0), sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, csPwd.GetBuffer(0), sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, csName.GetBuffer(0), sizeof(stInparam.szUserName) - 1);
stInparam.nPort = sPort;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
    while(0 == g_lLoginHandle)
    {
        // Log in to device
        LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

        if (0 == g_lLoginHandle)
        {
```

```
            // Find the meanings of error codes in dhnetsdk.h. Here the print is hexadecimal and the header file is decimal. Take care of conversion.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23)            // Do not support this function. The corresponding error code is 0x80000017, and the corresponding hexadecimal is 0x17.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp , g_nPort);
        }
        // When first time logging in, some data is needed to be initialized to enable normal business function. It is recommended to wait for a while after login, and the waiting time varies by devices.
        Sleep(1000);
        printf("\n");
    }
}

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

    if (0 == g_lLoginHandle)
    {
        return;
    }

    // Subscribe smart snapshot 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, call CLIENT_RealLoadPictureEx twice and set different event type.
    g_lRealLoadHandle = CLIENT_RealLoadPictureEx(g_lLoginHandle, nChannel, EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, dwUser, NULL);
    if (0 == g_lRealLoadHandle)
    {
        printf("CLIENT_RealLoadPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
        return;
    }

    // Manually snapshot to trigger smart snapshot alarm
    while(1)
    {
        char szGetBuf[64] = "";
        printf("manual snap, \'q\': quit, other: yes\n");
        gets(szGetBuf);
        // Input 'q' to exit manually 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;
        memcpy(stuSanpParam.bySequence, "just for test", sizeof(stuSanpParam.bySequence) - 1);
        // Manually snapshot trigger alarm function, and this function is only valid for ITC device.
        if (FALSE == CLIENT_ControlDeviceEx(g_lLoginHandle, 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 snapshot alarm.
    if (0 != g_lRealLoadHandle)
    {
        if (FALSE == CLIENT_StopLoadPic(g_lRealLoadHandle))
        {
            printf("CLIENT_StopLoadPic Failed!Last Error[%x]\n", CLIENT_GetLastError());
        }
        else
        {
            g_lRealLoadHandle = 0;
        }
    }

    // Log ou t of 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 initialization resources
    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 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("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
```

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

int CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE
*pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    if (lAnalyzerHandle != g_lRealLoadHandle)
    {
        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 snapshots being received at the same time, mark with i than saving snapshots
by 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_%s.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

3.1 fDisConnect

Table 3-1 fDisConnect

Item	Description	
Name	Disconnection callback function.Informing users by this callback when logined device gets disconnected.	
Precondition	None.	
Function	typedef void(CALLBACK *fDisConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
Parameter	lLoginID	Logined device ID.Return value of interface CLIENT_LoginWithHighLevelSecurity.
	pchDVRIP	Device IP.Disconnected device IP which is the input IP of login interface.
	nDVRPort	Device port.Disconnected device port which is the input port of login interface.
	dwUser	User data which should be the same with the imported value when setting fDisConnect.
Return value	None.	
Note	Set this callback in interface CLIENT_Init. Users can identify which device gets disconnected by parameters lLoginID, pchDVRIP and nDVRPort.	

3.2 fHaveReConnect

Table 3-2 fHaveReConnect

Item	Description	
Name	Successful reconnection callback function. When the disconnected device gets reconnected, call this interface to inform users.	
Precondition	None.	
Function	typedef void (CALLBACK *fHaveReConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
Parameter	lLoginID	Logined device ID.Return value of interface CLIENT_LoginWithHighLevelSecurity.
	pchDVRIP	Device IP.Disconnected device IP

Item	Description	
		which is the input IP of login interface.
	nDVRPort	Device port.Disconnected device port which is the input port of login interface.
	dwUser	User data which should be the same with the imported value when setting fDisConnect.
Return value	None.	
Note	Set this callback in interface CLIENT_SetAutoReconnect. Users can identify which device gets reconnected by parameters lLoginID, hDVRIP and nDVRPort.	

3.3 fRealDataCallBackEx

Table 3-3 fRealDataCallBackEx

Item	Description	
Name	Real-time monitoring data callback function prototype extension	
Precondition	None.	
Function	typedef void (CALLBACK *fRealDataCallBackEx)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser);	
Parameter	lRealHandle	Real-time monitoring handle. Return value of interfaces pulling real-time monitoring bitstream, such as CLIENT_RealPlayEx.
	dwDataType	Data type call backed by mark. It is determined by dwFlag of CLIENT_SetRealDataCallBackEx. 0: Original data which is consistent with data saved by SaveRealData. 1: Frame data. 2: Yuv data. 3: Pcm audio data.
	pBuffer	Buffer for callback data. Data of different length will be called back according to different data type. The data are called back by frame for every type but type 0, and each time one frame is called back.
	dwBufSize	Callback data length. The data buffers are diffreent for different types. The unit is byte.

Item	Description	
	param	Callback parameter structure. Different type value corresponds to different parameter structure. The structure is 0 when type is 0 or 2. When dwDataType is 1, param is a pointer of structure tagVideoFrameParam. For details, see tagVideoFrameParam. When dwDataType is 3, param is a pointer of structure tagCBPCMDataParam. For details, see tagCBPCMDataParam.
	dwUserData	User data which should be the same with the imported value when setting fRealDataCallBackEx.
Return value	None.	
Note	Set this callback in interface CLIENT_SetRealDataCallBackEx. In this callback, users can indentify which callback data is monitored in real time by lRealHandle.	

3.4 fDownLoadPosCallBack

Table 3-4 fDownLoadPosCallBack

Item	Description	
Name	Playback progress callback function	
Precondition	None.	
Function	typedef int (CALLBACK *fDataCallBack)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);	
Parameter	lPlayHandle	Playback handle. Return value of playback interfaces such as CLIENT_PlayBackByTimeEx.
	dwTotalSize	Total size of the current play. The unit is KB.
	dwDownLoadSize	The size that has been played. The unit is KB. When the value is -1, it means the end of the playback; and -2 means it failed to write the file.
	dwUser	User data which should be the same with the imported value when setting fDownLoadPosCallBack.
Return value	None.	
Note	Set this callback in record palyback interfaces, such as CLIENT_PlayBackByTimeEx. In this callback, users can indentify which progress callback corresponding to the current stream by lRealHandle.	

3.5 fDataCallBack

Table 3-5 fDataCallBack

Item	Description	
Name	Playback data callback function	
Precondition	None.	
Function	typedef int (CALLBACK *fDataCallBack)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);	
Parameter	lPlayHandle	Playback handle. Return value of playback interfaces such as CLIENT_PlayBackByTimeEx.
	dwDataType	Data type.The value remains 0, which means data of original type.
	pBuffer	Data buffer which is used to store video data of this callback.
	dwBufSize	Data stored buffer length. The unit is byte.
	dwUser	User data which should be the same with the imported value when setting fDataCallBackEx.
Return value	None.	
Note	Set the callback function in record playback interfaces such as CLIENT_PlayBackByTimeEx. If parameter, if hWnd is not NULL, no matter what value returns, the callback is being considered successful and next callback will return follow-up data. In this callback, users can indentify which progress callback corresponding to the current stream by lRealHandle.	

3.6 fTimeDownLoadPosCallBack

Table 3-6 fTimeDownLoadPosCallBack

Item	Description	
Name	Callback of download by time.	
Precondition	None.	
Function	typedef void (CALLBACK *fTimeDownLoadPosCallBack) (LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser);	
Parameter	lPlayHandle	Download handle. Return value of playback interfaces such as CLIENT_DownloadByTimeEx.
	dwTotalSize	Total size of playback. The unit is KB.

Item	Description	
	dwDownLoadSize	The size that has been played. The unit is KB.
	index	Sequence number of the currently downloaded video file, starting from 0.
	recordfileinfo	Current downloaded files information. For details, see structure NET_RECORDFILE_INFO.
	dwUser	User data which should be the same with the imported value when setting fTimeDownLoadPosCallBack.
Return value	None.	
Note	Set the callback function in interfaces downloading by time, such as CLIENT_PlayBackByTimeEx. In this callback, users can indentify which progress callback corresponding to the record download by lRealHandle.	

3.7 fMessCallBack

Table 3-7 fMessCallBack

Item	Description	
Name	Alarm report callback function prototype	
Precondition	None.	
Function	typedef BOOL (CALLBACK *fMessCallBack)(LONG lCommand, LLONG lLoginID, char *pBuf, DWORD dwBufLen, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
Parameter	lCommand	Alarm event type of callback which is matched with pBuf for usage. Different lCommands have different types of pBuf. For details, see the following descriptions.
	lLoginID	Device login ID. Return value of device login interfaces such as CLIENT_LoginWithHighLevelSecurity.
	pBuf	Alarm data received buffer. pBuf points to different data type according to different listen interface and different lCommand.
	dwBufLen	Length of alarm data received buffer. The unit is byte.
	pchDVRIP	Device IP which reports alarm.
	nDVRPort	Device port which reports alarm.
	dwUser	User data which should be the same with the imported value when setting fMessCallBack.
Return value	None.	

Item	Description
Note	All the logined device use the same alarm report callback function. Users indentify which login report the alarm by parameter lLoginID. pBuf points to different data type according to different listen interface and different lCommand. As there are too many alarm events, here does not introduce them all, and users can search the following key section in dhnetsdk.h: // Extensive alarm type, corresponding to CLIENT_StartListenEx #define DH_ALARM_ALARM_EX 0x2101 // External alarm To find the corresponding descriptions.

3.8 fSearchDevicesCB

Table 3-8 fSearchDevicesCB

Item	Description	
Name	Device search callback prototype	
Precondition	None.	
Function	typedef void (CALLBACK *fSearchDevicesCB)(DEVICE_NET_INFO_EX *pDevNetInfo, void* pUserData);	
Parameter	pDevNetInfo	Device info structure. For details, see structure DEVICE_NET_INFO_EX.
	pUserData	User data which should be the same with the imported value when setting fSearchDevicesCB.
Return value	None.	
Note	Device search callback function. It is not recommended to call SDK interfaces in this callback function. Set the callback function by CLIENT_StartSearchDevices/ CLIENT_SearchDevicesByIPs. When device is searched out, SDK will call this function.	

3.9 fSearchDevicesCBEx

Table 3-9 fSearchDevicesCBEx

Item	Description	
Name	Device search callback prototype	
Precondition	None.	
Function	typedef void(CALLBACK * fSearchDevicesCBEx)(LLONG lSearchHandle, DEVICE_NET_INFO_EX2 *pDevNetInfo, void* pUserData);	
Parameter	lSearchHandle	Returned serach handle of CLIENT_StartSearchDevicesEx.

Item	Description	
	pDevNetInfo	Device inforamtion structure. For details, see structure definition of DEVICE_NET_INFO_EX2.
	pUserData	User data which should be the same with the imported value when setting fSearchDevicesCBEx.
Return value	None.	
Note	Device search callback function. It is not recommended to call SDK interfaces in this callback function. Set the callback function by CLIENT_StartSearchDevicesEx. When device is searched out, SDK will call this function.	

3.10 fAnalyzerDataCallBack

Table 3-10 fAnalyzerDataCallBack

Item	Description	
Name	Smart snapshot alarm callback function prototype	
Precondition	None.	
Function	typedef int (CALLBACK *fAnalyzerDataCallBack)(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved);	
Parameter	lAnalyzerHandle	Smart snapshot alarm subscription handle. When multiple samrt snapshot alarm subscriptions use the same callback function, users can find the corresponding subscription operation by lAnalyzerHandle.
	dwAlarmType	Smart snapshot alarm type which is mathed with pAlarmInfo to use. pAlarmInfo points to different data type according to different dwAlarmType value.
	pAlarmInfo	Structure pointer which is mathed with dwAlarmType to use. pAlarmInfo points to different data type according to different dwAlarmType value.
	pBuffer	Smart snapshot info buffer.
	dwBufSize	Smart snapshot info size.
	dwUser	User data which should be the same with the imported value when setting fSearchDevicesCB.
	nSequence	Whether the sanpshot is repeated 0: It is the first time that the picture shows up, and the follow-up alarms may use the picture.

Item	Description	
		1: This picture is the same as the one shown in the last alarm, and the follow-up alarms may use the picture. 2: This picture is the same as the one shown in the last alarm. It will never show up again or it is the only time that the picture shows up. (Most of the alarms have an unique snapshot ,and nSequence valus is 2 generally.)
	reserved	Satatus of the current callback. Reserved is the int pointer. *(int *)reserved value: 0: Current data is real-time data. 1: Current data is off-line data. 2: Off-line transfer is finished. (Most of the smart snapshot alarm data is real-time data, and the value of *(int *)reserved is 0 generally.)
Return value	The return value has no meaning. Users can return 0.	
Note	Smart snapshot alarm callback function. It is not recommended to call SDK interfaces in this callback function. Set the callback function by CLIENT_RealLoadPictureEx/ CLIENT_RealLoadPicture. When smart snapshot alarm is reported by device, SDK will call this function. SDK receving buffer is 2M by default, so that users need to call CLIENT_SetNetworkParam to set receiving buffer again when callback snapshot info exceeds 2M; otherwise SDK will abandon data pack over 2M. Different dwAlarmType matches with different pointer. As there are too many alarm events, here does not introduce them all, and users can search the following key section in dhnetsdk.h: // Smart analysis event type #define EVENT_IVS_ALL 0x00000001// Subscribe all events To find the corresponding descriptions.	

3.11 fSnapRev

Table 3-11 fSnapRev

Item	Description
Name	Front-end video snapshot callback function prototype
Precondition	None.
Function	typedef void (CALLBACK *fSnapRev)(LLONG lLoginID, BYTE *pBuf, UINT RevLen, UINT EncodeType, DWORD CmdSerial, LDWORD dwUser);

Item	Description	
Parameter	lLoginID	Device login ID. When multiple front-end video snapshots use the same callback function, users can indentify which snapshot is by this parameter.
	pBuf	Sanpshot info buffer. Used to store the sanpshot info returned by storage device.
	RevLen	Snapshot info buffer size.
	EncodeType	Encode type 10: jpeg picture 0: i frame of mpeg4
	CmdSerial	Serial number of snapshot. It is input by CLIENT_SnapPictureEx input parameter
	dwUser	User data which should be the same with the imported value when setting fSnapRev.
Return value	None.	
Note	Snapshot callback function. It is not recommended to call SDK interfaces in this callback function. Set the callback function by CLIENT_SetSnapRevCallBack. When there are snapshot data sent by device, SDK will call this function. SDK receving buffer is 2M by default, so that users need to call CLIENT_SetNetworkParam to set receiving buffer again when callback snapshot info exceeds 2M; otherwise SDK will abandon data pack over 2M.	

3.12 fRealPlayDisConnect

Table 3-12 fRealPlayDisConnect

Item	Description	
Name	Real-time monitoring disconnection callback function prototype	
Precondition	None.	
Function	typedef void (CALLBACK *fRealPlayDisConnect)(LLONG lOperateHandle, EM_REALPLAY_DISCONNECT_EVENT_TYPE dwEventType, void* param, LDWORD dwUser);	
Parameter	lOperateHandle	Real-time monitoring handle. When multiple real-time monitoring devices use the same callback function, users can identify the cprresonding operation by this parameter.
	dwEventType	Event which causes disconnection. For details, see enum description of EM_REALPLAY_DISCONNECT_EVENT_TYPE.
	param	Reserved paramerer, and the default value is NULL.
	dwUser	User data which should be the same with the imported value when setting fRealPlayDisConnect.
Return value	None.	

Item	Description
Note	Real-time monitoring disconnection callback function. It is not recommended to call SDK interfaces in this callback function. Set the callback function by CLIENT_StartRealPlay. When eal-time monitoring isdisconnected, SDK will call this function.

3.13 pfAudioDataCallBack

Table 3-13 pfAudioDataCallBack

Item	Description	
Name	Audion data callback function protptype	
Precondition	None.	
Function	typedef void (CALLBACK *pfAudioDataCallBack)(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag, LDWORD dwUser);	
Parameter	lTalkHandle	Voice talk handle. Return value of voice talk interfaces such as CLIENT_StartTalkEx.
	pDataBuf	Audio data being called back Where the data from is decided by parameter byAudioFlag
	dwBufSize	Length of audio data being called back. The unit is byte.
	byAudioFlag	Flag indicates where the audio data from. 0: Receive PC audio data collected by local audio library. Only CLIENT_RecordStartEx is called, can the data be called back. 1: Receive audio data sent by device.
	dwUser	User data which should be the same with the imported value when setting pfAudioDataCallBack.
Return value	None.	
Note	Set the callback function in interfaces voice talk, such as CLIENT_StartTalkE.	

4 Structure Definition

4.1 NET_DEVICEINFO

Table 4-1 NET_DEVICEINFO

Option	Instruction
Struct description	Device info

Structure	typedef struct
	{
	BYTE sSerialNumber[DH_SERIALNO_LEN];
	BYTE byAlarmInPortNum;
	BYTE byAlarmOutPortNum;
	BYTE byDiskNum;
	BYTE byDVRType;
	union
	{
	BYTE byChanNum;
	BYTE byLeftLogTimes;
	};
	} NET_DEVICEINFO, *LPNET_DEVICEINFO;

Members	sSerialNumber SN byAlarmInPortNum DVR alarm input amount byAlarmOutPortNum DVR alarm output amount byDiskNum DVR HDD amount byDVRType DVR type.Refer to NET_DEVICE_TYPE. byChanNum DVR channel amount. It is valid after user logged in. byLeftLogTimes When login failed due to password error, prompt user by this parameter. Remaining login times 0 means this parameter is invalid.

4.2 NET_PARAM

Table 4-2 NET_PARAM

Option	Instruction
Struct description	Relevant parameters of login
Struct	typedef struct { int nWaittime; int nConnectTime; int nConnectTryNum; int nSubConnectSpaceTime; int nGetDevInfoTime; int nConnectBufSize;

Option	Instruction
	int nGetConnInfoTime; int nSearchRecordTime; int nsubDisconnetTime; BYTE byNetType; BYTE byPlaybackBufSize; BYTE bDetectDisconnTime; BYTE bKeepLifeInterval; int nPicBufSize; BYTE bReserved[4]; } NET_PARAM;
Members	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 Get device information timeout, 0:default 1000ms. nConnectBufSize Receiving data buffer size of each connection(Bytes), 0:default 250*1024 nGetConnInfoTime Getting sub-connect information timeout(Unit is ms), 0:default 1000ms. nSearchRecordTime Timeout value of search video (unit ms), default 3000ms nsubDisconnetTime Waiting time of sub-link offline detection (unit ms), default 6000ms byNetType Network type,0-LAN,1-WAN. byPlaybackBufSize Playback data receiving buffer size(Unit;M). 0: default 4M. bDetectDisconnTime Pulse detection offline time(second) .When it is 0, the default setup is 60s, and the min time is 2s. bKeepLifeInterval Pulse sending out interval(second).When it is 0, the default setup is 10s, the min internal is 2s. nPicBufSize Receiving buffer size of real-time piciture(Unit: byte). 0: default 2*1024*1024. bReserved Reserved byte

4.3 NET_DEVICEINFO_Ex

Table 4-3 NET_DEVICEINFO_Ex

Option	Instruction
Struct description	Device info extension
Struct	typedef struct { BYTE sSerialNumber[DH_SERIALNO_LEN]; int nAlarmInPortNum; int nAlarmOutPortNum; int nDiskNum;

Option	Instruction
	int nDVRType; int nChanNum; BYTE byLimitLoginTime; BYTE byLeftLogTimes; BYTE bReserved[2]; int nLockLeftTime; char Reserved[24]; } NET_DEVICEINFO_Ex, *LPNET_DEVICEINFO_Ex;
Members	sSerialNumber Device SN nAlarmInPortNum DVR alarm input amount nAlarmOutPortNum DVR alarm output amount nDiskNum DVR HDD amount nDVRType DVR type.Refer to NET_DEVICE_TYPE. nChanNum DVR channel amount. It is valid after user logged in. byLimitLoginTime Online timeout. 0: no login limit. If it is not 0,it means the login limit time (Unit: Minute). byLeftLogTimes When login failed due to password error, prompt user by this parameter. bReserved Reserved byte. Byte alignment. nLockLeftTime Once login failed, it means the user unlock remaining time (Unit: second). -1: Current parameter is null. Reserved Reserved byte

4.4 NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY

Table 4-4 NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY

Option	Instruction
Struct description	CLIENT_LoginWithHighLevelSecurity input parameters

Struct	typedef struct tag NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY
	{
	DWORD dwSize; // Structure size
	char szIP[64]; // IP
	int nPort; // Port
	char szUserName[64]; // User name
	char szPassword[64]; // Password
	EM_LOGIN_SPAC_CAP_TYPE emSpecCap; // Login mode
	BYTE byReserved[4]; // Byte alignment
	void* pCapParam; //A complementary function of emSpecCap
	} NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY;

Members	dwSize Structure size. Assign a value when using sizeof(NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY). szIp

Option	Instruction
	Device IP nPort Login port szUserName User name szPassword Password emSpecCap Login mode. The capabilities the device supports. Refer to enumeration note of EM_LOGIN_SPAC_CAP_TYPE byReserved Byte alignment pCapParam The complementary function of emSpecCap, working with emSpecCap. Refer to enumeration note of EM_LOGIN_SPAC_CAP_TYPE. Input NULL if the value of pCapParam has no corresponding note.

4.5 NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY

Table 4-5 NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY

Item	Description
Struct description	Output parameters of CLIENT_LoginWithHighLevelSecurity

Struct	typedef struct tagNET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY
	{
	DWORD dwSize; // Structure size
	NET_DEVICEINFO_Ex stuDeviceInfo; // Device information
	int nError; // error code. Refer to error code of CLIENT_Login
	BYTE byReserved[132]; // Reserved field
	}NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY;

Members	dwSize Structure size. Assign a value when using sizeof(NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY). stuDeviceInfo When the device successfully logged in, it saves the some information of the logged in device. When the device failed to login, it saves the information about the login such as remaining login attempts. Refer to structure note of NET_DEVICEINFO. nError （It is null if the function returned successfully）, return login error code. Refer to the following contents. 1-Incorrect password 2 - User name does not exist 3 - Login timeout. 4 - Already logged in to this account. 5 - Account locked. 6 - The account is blocklisted 7 - System is busy. Resources are insufficient 8 - Sub-connection failed. 9 - Main connection failed. 10 - Exceeded maximum connections. byReserved Reserved field

4.6 NET_IN_STARTSERACH_DEVICE

Table 4-6 NET_IN_STARTSERACH_DEVICE

Option	Instruction
Struct description	Input parameters of CLIENT_StartSearchDevicesEx

Struct	typedef struct tagNET_IN_STARTSERACH_DEVICE
	{
	DWORD dwSize; // Structure size
	char szLocalIp[MAX_LOCAL_IP_LEN]; // The local IP that starts searching
	fSearchDevicesCBEx cbSearchDevices; // Device information
	void* pUserData; // User self-defined data
	EM_SEND_SEARCH_TYPE emSendType; // Sending out search type
	}NET_IN_STARTSERACH_DEVICE;

Members	dwSize Structure size. Assign a value when using sizeof(NET_IN_STARTSERACH_DEVICE). szLocalIp The local IP that starts searching Do not need to input. The default value is NULL. cbSearchDevices Device info callback function When there is corresponding package from the device, NetSDK parses the package to valid information. And then use callback function to notify the user. Refer to the callback function note of fSearchDevicesCBEx. Callback address cannot be null. pUserData User self-defined data NetSDK searches device callback function fSearchDevicesCB to return the data to user so that the user can continue the following operations. emSendType Search type enumeration including multicast and broadcast. Refer to the enumeration definition of EM_SEND_SEARCH_TYPE

4.7 NET_OUT_STARTSERACH_DEVICE

Table 4-7 NET_OUT_STARTSERACH_DEVICE

Item	Description
Struct description	Output parameters of CLIENT_StartSearchDevicesEx

Struct	typedef struct tagNET_OUT_STARTSERACH_DEVICE
	{
	DWORD dwSize; // Structure size
	}NET_OUT_STARTSERACH_DEVICE;

Members	dwSize Structure size. Assign a value when using sizeof(NET_OUT_STARTSERACH_DEVICE).

4.8 tagVideoFrameParam

Table 4-8 tagVideoFrameParam

Item	Description
Struct description	Frame structure of calling video data frame
Struct	typedef struct _tagVideoFrameParam { BYTE encode; BYTE frametype; BYTE format; BYTE size; DWORD fourcc; WORD width; WORD height; NET_TIME struTime; } tagVideoFrameParam;
Members	encode Encode Type Different values have different encode types. As follows: MPEG4 encode – 1 Dahua H.264 encode -2 ADI H.264 encode – 3 Standard H.264 encode - 4 frametype Frame type Different values have different frame types. As follows: I frame - 0 P frame - 1 B frame - 2 format Video format Different values have different video formats. As follows: RAID 0 RAID 1 size Resolution Different values have different resolutions. As follows: RAID 0 HD1 – 1 2CIF2 D1 – 3 VGA – 4 QCIF – 5 QVGA – 6 SVCD – 7 QQVGA – 8 SVGA – 9 XVGA – 10 WXGA – 11 SXGA – 12 WSXGA – 13 UXGA – 14 WUXGA – 15 LFT – 16 720 – 17 1080 - 18 fourcc If it is H.264 encode, the total amount is 0. Otherwise the value is *(DWORD*)"DIVX",it is 0x58564944. width Width, unit is pixel. It is valid when size =255. height Height, unit is pixel. It is valid when size =255. struTime

Item	Description
	Time info Refer to the structure note of the NET_TIME

4.9 tagCBPCMDataParam

Table 4-9 tagCBPCMDataParam

Item	Description
Struct description	Frame structure of callback audio data
Struct	typedef struct _tagCBPCMDataParam { BYTE channels; BYTE samples; BYTE depth; BYTE param1; DWORD reserved; } tagCBPCMDataParam;
Members	channels Sound track amount samples Sampling rate Different values have different sampling rates. As follows: 0 – 8000 1 – 11025 2 - 16000 3 - 22050 4 - 32000 5 - 44100 6 - 48000 depth Sampling depth Value is 8,16 and so on. param1 Audio data type 0 - Indication without icon 1 - Indication with icon reserved Reserve

4.10 NET_TIME

Table 4-10 NET_TIME

Item	Description
Struct description	Time structure. Unit is second.
Struct	typedef struct { DWORD dwYear; DWORD dwMonth; DWORD dwDay; DWORD dwHour; DWORD dwMinute; DWORD dwSecond; } NET_TIME,*LPNET_TIME;
Members	dwYear Year dwMonth Month

Item	Description
	dwDay Day dwHour Hour dwMinute Minute dwSecond Second

4.11 NET_RECORDFILE_INFO

Table 4-11 NET_RECORDFILE_INFO

Item	Description
Struct description	Structure description
Struct	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 bImportantRecID; BYTE bHint; BYTE bRecType; } NET_RECORDFILE_INFO, *LPNET_RECORDFILE_INFO;
Members	ch Channel number filename File name framenum File total frame amount size File length starttime Start time endtime End time driveno Disk No. （Distinguishes network record file and local record file,0～127:local record file, 64 means disc 1,128 means network record file.） startcluster Begin cluster No. nRecordFileType Record file types 0:General record;1:Alarm record;2:Motion detection; 3:Card number record;4:picture;5:IVS record bImportantRecID The flag to be the important record or not 0:General record;1:Important record bHint File positioning index （When nRecordFileType==4<picture>,bImportantRecID<<8 +bHint ,creating picture positioning index）

Item	Description
	bRecType Device record stream type 0 - Main stream record 1 - Sub stream 1 Record 2 - Sub stream 2 Record 3 - Sub stream 3 Record

4.12 CFG_PTZ_PROTOCOL_CAPS_INFO

Table 4-12 CFG_PTZ_PROTOCOL_CAPS_INFO

Item	Description
Struct description	PTZ capability set information structure
Struct	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; 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;

Item	Description
	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; BOOL bSupportPresetTimeSection; BOOL bFocus; }CFG_PTZ_PROTOCOL_CAPS_INFO;
Members	nStructSize Assign value as sizeof(CFG_PTZ_PROTOCOL_CAPS_INFO) bPan Supports PTZ horizontal movement or not. bTile Supports PTZ vertical movement or not. bZoom Supports PTZ zoom or not bIris Supports PTZ iris adjustment or not. bPreset Supports preset or not bRemovePreset Supports to delete preset or not bTour Supports tour or not bRemoveTour Supports to delete tour or not bPattern Support pattern or not bAutoPan Supports auto horizontal movement or not. bAutoScan Supports auto scan or not bAux Supports AUX function or not bAlarm Supports alarm or not bLight Supports light or not. Refer to the "stuPtzLightingControl" member listed below. bWiper Supports wiper or not bFlip Supports lens flip or not bMenu Supports PTZ built-in menu or not bMoveRelatively Supports to positioning PTZ according to the relative coordinates bMoveAbsolutely Supports to positioning PTZ according to the absolute coordinates bReset Supports reset PTZ or not bGetStatus Supports to get PTZ moving status and directional coordinates bSupportLimit

Item	Description
	Supports PTZ limit or not bPtzDevice Supports PTZ device or not. bIsSupportViewRange Supports PTZ visual range wCamAddrMin The min. value of the channel address wCamAddrMax The max. value of the channel address wMonAddrMin The min. value of the monitor address wMonAddrMax The max. value of the monitor address wPresetMin The min. value of preset wPresetMax The max. value of preset wTourMin The min. value of auto tour wTourMax The max. value of auto tour wPatternMin The min. value of pattern wPatternMax The max. value of pattern wTileSpeedMin The min. value of vertical speed wTileSpeedMax The max. value of vertical speed wPanSpeedMin The min. value of horizontal speed wPanSpeedMax The max. value of horizontal speed wAutoScanMin The min. value of auto scan wAutoScanMax The max. value of auto scan wAuxMin The min. value of aux function wAuxMax The max. value of aux function dwInterval Time interval of sending command. dwType Protocol type,0-Local PTZ,1-Remote PTZ dwAlarmLen Protocol alarm length dwNearLightNumber Near light groups amount,0~4,0: does not support this function dwFarLightNumber Far light groups amount,0~4,0: does not support this function dwSupportViewRangeType The submask of getting the supported visual range type. From low bit to high bit. Right now supports The 1st bit:1: supports ''ElectronicCompass'' dwSupportFocusMode The submask of supported focus mode. From low bit to high bit. Refer to the enumeration note of EM_SUPPORT_FOCUS_MODE szName Operation protocol name

Item	Description
	szAuxs PTZ AUX function list stuPtzMotionRange PTZ movement angles range. The unit is degree. Refer to the structure note of CFG_PTZ_MOTION_RANGE stuPtzLightingControl Light control contents. Refer to the structure note of CFG_PTZ_LIGHTING_CONTROL bSupportPresetTimeSection Supports preset period setup or not bFocus Supports PTZ focus or not

4.13 CFG_PTZ_MOTION_RANGE

Table 4-13 CFG_PTZ_MOTION_RANGE

Item	Description
Struct description	PTZ movement angles range structure
Struct	typedef struct tagCFG_PTZ_MOTION_RANGE { int nHorizontalAngleMin; int nHorizontalAngleMax; int nVerticalAngleMin; int nVerticalAngleMax; }CFG_PTZ_MOTION_RANGE;
Members	nHorizontalAngleMin Min. horizontal angle value. The unit : degree nHorizontalAngleMax Max. horizontal angle value. The unit : degree nVerticalAngleMin Min. vertical angle value. The unit : degree nVerticalAngleMax Max. vertically angle value. The unit : degree

4.14 CFG_PTZ_LIGHTING_CONTROL

Table 4-14 CFG_PTZ_LIGHTING_CONTROL

Item	Description
Struct description	Light control contents structure
Struct	typedef struct tagCFG_PTZ_LIGHTING_CONTROL { char szMode[CFG_COMMON_STRING_32]; DWORD dwNearLightNumber; DWORD dwFarLightNumber; }CFG_PTZ_LIGHTING_CONTROL;
Members	szMode Manual light control mode "on-off": on-off mode "adjustLight": Manually adjusting light dwNearLightNumber NearLight group amount dwFarLightNumber FarLight group amount

4.15 DHDEV_TALKFORMAT_LIST

Table 4-15 DHDEV_TALKFORMAT_LIST

Item	Description
Struct description	The audio talk type supported by the device
Struct	typedef struct { Int nSupportNum; DHDEV_TALKDECODE_INFO type[64]; Char reserved[64]; } DHDEV_TALKFORMAT_LIST;
Members	nSupportNum Supported amount type Encode type Refer to the structure note of DHDEV_TALKDECODE_INFO reserved Reserved byte

4.16 DHDEV_TALKDECODE_INFO

Table 4-16 DHDEV_TALKDECODE_INFO

Item	Description
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;
Members	encodeType Encode type Refer to the enumeration note of DH_TALK_CODING_TYPE nAudioBit Bit such as 8, 16. dwSampleRate Sampling rate such as 8000,16000 nPacketPeriod Packet interval. The unit. ms reserved Reserved byte

4.17 DHDEV_SYSTEM_ATTR_CFG

Table 4-17 DHDEV_SYSTEM_ATTR_CFG

Item	Description
Struct description	System Information
Struct	typedef struct { DWORD dwSize; /* The following contents are the read-only part of the device */ DH_VERSION_INFO stVersion; DH_DSP_ENCODECAP stDspEncodeCap; BYTE szDevSerialNo[DH_DEV_SERIALNO_LEN];

Item	Description
	BYTE byDevType; BYTE szDevType[DH_DEV_TYPE_LEN]; BYTE byVideoCaptureNum; BYTE byAudioCaptureNum; BYTE byTalkInChanNum; BYTE byTalkOutChanNum; BYTE byDecodeChanNum; BYTE byAlarmInNum; BYTE byAlarmOutNum; BYTE byNetIONum; BYTE byUsbIONum; BYTE byIdeIONum; BYTE byComIONum; BYTE byLPTIONum; BYTE byVgaIONum; BYTE byIdeControlNum; BYTE byIdeControlType; BYTE byCapability; BYTE byMatrixOutNum; /* The following contents are the writable part of the device */ 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;
Members	dwSize Structure size. Assign a value sizeof(DHDEV_SYSTEM_ATTR_CFG) /* The following contents are the read-only part of the device */ stVersion Device version information. Refer to the structure note of DH_VERSION_INFO stDspEncodeCap DSP capability description. Refer to the structure note of DH_DSP_ENCODECAP szDevSerialNo Device serial number byDevType Device type. Refer to the enumeration of NET_DEVICE_TYPE szDevType Device detailed model,. String format. It can be null sometimes. byVideoCaptureNum Video port amount byAudioCaptureNum Audio port amount byTalkInChanNum Audio talk input port amount byTalkOutChanNum Audio talk output port amount byDecodeChanNum Decode port amount byAlarmInNum Alarm input port amount byAlarmOutNum Alarm output port amount byNetIONum Network port amount

Item	Description
	byUsbIONum USB port amount byIdeIONum IDE amount byComIONum Serial port amount byLPTIONum LPT port amount byVgaIONum VGA port amount byIdeControlNum IDE control amount byIdeControlType IDE control type byCapability Device capabilities, extension description byMatrixOutNum Video matrix output port /* The following contents are the writable part of the device */ byOverWrite When HDD is full (1: Stop. 0: Overwrite) byRecordLen Record file pack duration byDSTEnable Enable DST or not. 1: enable. 0: disable. wDevNo Device SN. For remote control. byVideoStandard Video format :0-PAL,1-NTSC byDateFormat Date format byDateSprtr Date separator (0:".",1:"-",2:"/") byTimeFmt Time format (（0～24H,1～12H) byLanguage Language type. Refer to the enumeration of DH_LANGUAGE_TYPE.

4.18 NET_SPEAK_PARAM

Table 4-18 NET_SPEAK_PARAM

Item	Description
Struct description	Audio parameter structure
Struct	typedef struct __NET_SPEAK_PARAM { DWORD dwSize; int nMode; int nSpeakerChannel; BOOL bEnableWait; } NET_SPEAK_PARAM;
Members	dwSize Structure size. The assign value is sizeof(NET_SPEAK_PARAM) nMode Mode type, 0: audio talk （default）, 1: broadcast; resetting required if switching from broadcast to audio talk. nSpeakerChannel Speaker channel number. It is valid in broadcast mode. bEnableWait

Item	Description
	Waiting for device to respond or not when enabling the audio talk. The default value is FALSE. TRUE:Wait;FALSE:Do not wait. The timeout time is set by CLIENT_SetNetworkParam,corresponding to nWaittime of NET_PARAM.

4.19 NET_TALK_TRANSFER_PARAM

Table 4-19 NET_TALK_TRANSFER_PARAM

Item	Description
Struct description	Enable the transfer mode of the audio talk.
Struct	typedef struct tagNET_TALK_TRANSFER_PARAM { DWORD dwSize; BOOL bTransfer; }NET_TALK_TRANSFER_PARAM;
Members	dwSize Structure the size. The assign value is sizeof(NET_TALK_TRANSFER_PARAM) bTransfer Enable audio talk transfer mode or not. TRUE:Enable transfer,FALSE: Disable transfer

4.20 DEVICE_NET_INFO_EX

Table 4-20 DEVICE_NET_INFO_EX

Item	Description
Struct description	Device search callback message structure
Struct	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; char cReserved[244]; }DEVICE_NET_INFO_EX;
Members	iIPVersion IP protocol,4 for IPV4, 6 for IPV6

Item	Description
	szIP IP string format,IP IPV4 such as "192.168.0.1" IPV6 such as "2008::1/64" nPort TCP Port szSubmask Subnet mask. IPV6 has no subnet mask szGateway Device gateway szMac Device MAC address szDeviceType Device type byManuFactory The manufacturer of the target device. Refer to EM_IPC_TYPE byDefinition 1-Standard definition 2-High definition bDhcpEn DCHP enable status, true-Enable,false-Disable byReserved1 Byte alignment verifyData Verify data. Asynchronously search callback to get. (Uses the information to verify when modifying device IP.) szSerialNo Serial number szDevSoftVersion Device software version szDetailType Device model szVendor OEM customer type szDevName Device name szUserName Logged in device user name (Input when modifying device IP) szPassWord Logged in device password (Input when modifying device IP) nHttpPort HTTP service port number. wVideoInputCh Video input channel amount wRemoteVideoInputCh Remote video input channel amount wVideoOutputCh Video output channel amount wAlarmInputCh Alarm input channel amount wAlarmOutputCh Alarm output channel amount cReserved Reserved byte

4.21 MANUAL_SNAP_PARAMETER

Table 4-21 MANUAL_SNAP_PARAMETER

Item	Description
Struct description	Manual Snapshot Structure
Struct	typedef struct _MANUAL_SNAP_PARAMETER{

Item	Description
	int nChannel; BYTE bySequence[64]; BYTE byReserved[60]; }MANUAL_SNAP_PARAMETER;
Members	nChannel Snapshot channel. Start from 0. bySequence Snapshot SN string. Returns current field when uploading corresponding intelligent picture alarm. Uses this string to check one by one when there are several manual snapshot events at the same time. byReserved Reserved field

4.22 OPR_RIGHT_EX

Table 4-22 OPR_RIGHT_EX

Item	Description
Struct description	Rights info structure
Struct	typedef struct _OPR_RIGHT_EX { DWORD dwID; char name[DH_RIGHT_NAME_LENGTH]; char memo[DH_MEMO_LENGTH]; } OPR_RIGHT_EX;
Members	dwID Right ID Each right has its own ID name Right name memo Right note

4.23 OPR_RIGHT_NEW

Table 4-23 OPR_RIGHT_NEW

Item	Description
Struct description	Rights info structure
Struct	typedef struct _OPR_RIGHT_NEW { DWORD dwSize; DWORD dwID; char name[DH_RIGHT_NAME_LENGTH]; char memo[DH_MEMO_LENGTH]; } OPR_RIGHT_NEW;
Members	dwSize Structure size. The assign value is sizeof(OPR_RIGHT_NEW) dwID Right ID Each right has its own ID name Right name memo Right note

4.24 NET_DEV_CHN_COUNT_INFO

Table 4-24 NET_DEV_CHN_COUNT_INFO

Item	Description
Struct description	Device channel amount information structure
Struct	typedef struct tagNET_DEV_CHN_COUNT_INFO { DWORD dwSize; NET_CHN_COUNT_INFO stuVideoIn; NET_CHN_COUNT_INFO stuVideoOut; } NET_DEV_CHN_COUNT_INFO;
Members	dwSize Structure size. The assign value is sizeof(NET_DEV_CHN_COUNT_INFO) stuVideoIn Video input channel Refer to the structure note of NET_CHN_COUNT_INFO stuVideoOut Video output channel Refer to the structure note of NET_CHN_COUNT_INFO

4.25 NET_CHN_COUNT_INFO

Table 4-25 NET_CHN_COUNT_INFO

Item	Description
Struct description	Channel amount information structure
Struct	typedef struct tagNET_CHN_COUNT_INFO { DWORD dwSize; int nMaxTotal; int nCurTotal; int nMaxLocal; int nCurLocal; int nMaxRemote; int nCurRemote; } NET_CHN_COUNT_INFO;
Members	dwSize Structure size. The assign value is sizeof(NET_CHN_COUNT_INFO) nMaxTotal Device total channel amount (The total quantity of the valid channels) nCurTotal Configured channel amount nMaxLocal Max. local channel amount. It includes the main board and then removable sub-card channel. nCurLocal Configured local channel amount nMaxRemote Max. remote channel amount nCurRemote Configured remote channel amount

4.26 NET_IN_SNAP_CFG_CAPS

Table 4-26 NET_IN_SNAP_CFG_CAPS

Item	Description
Struct description	Gets input parameter structure of the snapshot configuration

Item	Description
Struct	typedef struct tagNET_IN_SNAP_CFG_CAPS { int nChannelId; BYTE bReserved[1024]; }NET_IN_SNAP_CFG_CAPS;
Members	nChannelId Channel number bReserved Reserved byte

4.27 NET_OUT_SNAP_CFG_CAPS

Table 4-27 NET_OUT_SNAP_CFG_CAPS

Item	Description
Struct description	Gets output parameter structure of the snapshot configuration
Struct	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;
Members	nResolutionTypeNum Supported video resolution information Works with stuResolutionTypes stuResolutionTypes Video resolution information structure Works with nResolutionTypeNum dwFramesPerSecNum Supported frame rate information Works with nFramesPerSecList nFramesPerSecList Supported frame rate list Works with dwFramesPerSecNum dwQualityMun Supported video quality Works with nQualityList nQualityList Supported video quality List Works with dwQualityMun dwMode Mode. By bit:The 1st bit:schedule. The 2nd bit:manual dwFormat Picture format mode. By bit:The 1st bit:BMP. The 2nd bit:JPG bReserved Reserved byte

4.28 DH_RESOLUTION_INFO

Table 4-28 DH_RESOLUTION_INFO

Item	Description
Struct description	Picture resolution structure
Struct	typedef struct { unsigned short snWidth; unsigned short snHight; }DH_RESOLUTION_INFO;
Members	snWidth Width snHight Height

4.29 CFG_VIDEOENC_OPT

Table 4-29 CFG_VIDEOENC_OPT

Item	Description
Struct description	Video encode parameter structure
Struct	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;
Members	abVideoEnable Indicate the bVideoEnable is valid or not When getting, indicates support enable video or not When setting, indicates support modify video or not abAudioEnable Indicate the bAudioEnable is valid or not When getting, indicates support enable audio or not When setting, indicates support modify audio or not abSnapEnable Indicate the bSnapEnable is valid or not When getting, indicates support schedule snapshot or not When setting, indicates support modify schedule snapshot or not abAudioAdd Indicate the bAudioAddEnable is valid or not When getting, indicates support overlay audio or not When setting, indicates support modify overlay audio or not abAudioFormat Indicate the stuAudioFormat is valid or not When getting, indicates support audio format or not When setting, indicates support modify audio format or not bVideoEnable Enable video Works with abVideoEnable stuVideoFormat Video file format Refer to the structure note of NET_CHN_COUNT_INFO bAudioEnable Enable audio

Item	Description
	Works with abAudioEnable bSnapEnable Enable scheduled snapshot Works with abSnapEnable bAudioAddEnable Enable audio overlay Works with abAudioAdd stuAudioFormat Audio format Works with abAudioFormat Refer to the structure note of CFG_AUDIO_ENCODE_FORMAT

4.30 CFG_VIDEO_FORMAT

Table 4-30 CFG_VIDEO_FORMAT

Item	Description
Struct description	Struct
Struct	typedef struct tagCFG_VIDEO_FORMAT { bool abCompression; bool abWidth; 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;
Members	abCompression TRUE:emCompression is valid;FALSE:emCompression is null The string is read-only. Uses the getting value. Do not change. abWidth TRUE:nWidth is valid;FALSE:nWidth is null The string is read-only. Uses the getting value. Do not change. abHeight TRUE:nHeight 字 is valid;FALSE:nHeight 字 is null The string is read-only. Uses the getting value. Do not change. abBitRateControl TRUE:emBitRateControl is valid;FALSE:emBitRateControl is null The string is read-only. Uses the getting value. Do not change. abBitRate TRUE:nBitRateis valid;FALSE:nBitRate is null The string is read-only. Uses the getting value. Do not change. abFrameRate TRUE:nFrameRate is valid;FALSE:nFrameRate is null The string is read-only. Uses the getting value. Do not change.

Item	Description
	abIFrameInterval TRUE:nIFrameIntervalis valid;FALSE:nIFrameInterval is null The string is read-only. Uses the getting value. Do not change. abImageQuality TRUE:emImageQuality is valid;FALSE:emImageQuality is null The string is read-only. Uses the getting value. Do not change. abFrameType TRUE:nFrameTypeis valid;FALSE:nFrameType is null The string is read-only. Uses the getting value. Do not change. abProfile TRUE:emProfile is valid;FALSE:emProfile is null The string is read-only. Uses the getting value. Do not change. emCompression Video compression format The string is valid or not depending on abCompression Refer to the enumeration note of CFG_VIDEO_COMPRESSION nWidth Video Width The string is valid or not depending on abWidth nHeight Video Height The string is valid or not depending on abHeight emBitRateControl Bit Rate Control Mode The string is valid or not depending on abBitRateControl Refer to the enumeration note of CFG_BITRATE_CONTROL nBitRate Video bit stream (kbps) The string is valid or not depending on abBitRate nFrameRate Video Frame Rate The string is valid or not depending on abFrameRate nIFrameInterval I frame interval (1-100). For example, 50 means there is one I frame each 49 B frames or P frames. The string is valid or not depending on abIFrameInterval emImageQuality Image Quality The string is valid or not depending on abImageQuality Refer to the enumeration note of CFG_IMAGE_QUALITY nFrameType Packet mode. 0—DHAV,1—"PS" The string is valid or not depending on abFrameType emProfile H.264 encode mode The string is valid or not depending on abProfile Refer to the enumeration note of CFG_H264_PROFILE_RANK

4.31 CFG_AUDIO_ENCODE_FORMAT

Table 4-31 CFG_AUDIO_ENCODE_FORMAT

Item	Description
Struct description	Audio format structure
Struct	typedef struct tagCFG_AUDIO_FORMAT { bool abCompression; bool abDepth; bool abFrequency;

Item	Description
	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;
Members	abCompression TRUE:emCompression is valid;FALSE:emCompression is null The string is read-only. Uses the getting value. Do not change. abDepth TRUE:nDepthis valid;FALSE:nDepth is null The string is read-only. Uses the getting value. Do not change. abFrequency TRUE:nFrequency is valid;FALSE:nFrequency is null The string is read-only. Uses the getting value. Do not change. abMode TRUE:nMode is valid;FALSE:nMode is null The string is read-only. Uses the getting value. Do not change. abFrameType TRUE:nFrameType is valid;FALSE:nFrameType is null The string is read-only. Uses the getting value. Do not change. abPacketPeriod TRUE:nPacketPeriod is valid;FALSE:nPacketPeriod is null The string is read-only. Uses the getting value. Do not change. emCompression Audio Compression Mode The string is valid or not depending on abCompression Refer to the enumeration note of CFG_AUDIO_FORMAT nDepth Audio Sampling Depth The string is valid or not depending on abDepth nFrequency Audio Sampling Frequency The string is valid or not depending on abFrequency nMode Audio Encode Mode The string is valid or not depending on abMode nFrameType Audio package mode. 0-DHAV, 1-PS The string is valid or not depending on abFrameType nPacketPeriod Audio Packet Period (ms) The string is valid or not depending on abPacketPeriod

4.32 CFG_VIDEO_COVER

Table 4-32 CFG_VIDEO_COVER

Item	Description
Struct description	Multiple-zone Tampering Configuration Structure
Struct	typedef struct tagCFG_VIDEO_COVER { int nTotalBlocks; int nCurBlocks; CFG_COVER_INFO stuCoverBlock[MAX_VIDEO_COVER_NUM];

Item	Description
	} CFG_VIDEO_COVER;
Members	nTotalBlocks Supported tampering block amount nCurBlocks Configured block amount stuCoverBlock Tampering zone Refer to the structure note of CFG_COVER_INFO

4.33 CFG_COVER_INFO

Table 4-33 CFG_COVER_INFO

Item	Description
Struct description	Tampering Info Structure
Struct	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;
Members	abBlockType TRUE:nBlockType is valid;FALSE:nBlockType is null The string is read-only. Uses the getting value. Do not change. abEncodeBlend TRUE:nEncodeBlend is valid;FALSE:nEncodeBlend is null The string is read-only. Uses the getting value. Do not change. abPreviewBlend TRUE:nPreviewBlend is valid;FALSE:nPreviewBlend is null The string is read-only. Uses the getting value. Do not change. stuRect Tampering zone coordinates Refer to the structure note of CFG_RECT stuColor Tampering color Refer to the structure note of CFG_RGBA nBlockType Tampering mode. 0—Black—black block,1—mosaic The string is valid or not depending on abBlockType nEncodeBlend Encoding-level tampering. 1—valid,0—null The string is valid or not depending on abEncodeBlend nPreviewBlend Tampering when previewing. 1—valid,0—null The string is valid or not depending on abPreviewBlend

4.34 CFG_RECT

Table 4-34 CFG_RECT

Item	Description
Struct description	Area information structure
Struct	typedef struct tagCFG_RECT

Item	Description
	{ int nLeft; int nTop; int nRight; int nBottom; } CFG_RECT;
Members	nLeft Left Area nTop Top Area nRight Right Area nBottom Bottom Area
Struct description	RGBA information structure
Struct	typedef struct tagCFG_RGBA { int nRed; int nGreen; int nBlue; int nAlpha; } CFG_RGBA;
Members	nRed Red nGreen Green nBlue Blue nAlpha Transparency

4.35 CFG_ENCODE_INFO

Table 4-35 CFG_ENCODE_INFO

Item	Description
Struct description	Image channel attribute information structure
Struct	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;
Members	nChannelID Channel number, starting from 0 When getting the value, current field is valid. When setting, current field is null. szChnName Invalid field

Item	Description
	stuMainStream Main Stream Attribute Information stuMainStream[0]—Main stream general record attribute information stuMainStream[1] — Main stream motion detection record attribute information stuMainStream[2]—Main stream alarm record attribute information Refer to the structure note of CFG_VIDEOENC_OPT stuExtraStream Sub Stream Attribute Information stuMainStream[0]—Sub stream general record attribute information stuMainStream[1]—Sub stream general record attribute information stuMainStream[2]—Sub stream tampered alarm record attribute information Refer to the structure note of CFG_VIDEOENC_OPT stuSnapFormat Snapshot Attribute Information stuSnapFormat[0]—General snapshot attribute information stuSnapFormat[1]—Motion detection snapshot attribute information stuSnapFormat[2]—Alarm snapshot attribute information Refer to the structure note of CFG_VIDEOENC_OPT dwCoverAbilityMask Invalid Field dwCoverEnableMask Invalid Field stuVideoCover Invalid Field stuChnTitle Invalid Field stuTimeTitle Invalid Field stuVideoColor Invalid Field emAudioFormat Invalid Field nProtocolVer Protocol version number. Read-only. When getting the value, current field is valid. When setting, current field is null.

4.36 SNAP_PARAMS

Table 4-36 SNAP_PARAMS

Item	Description
Struct description	Snapshot parameters structure
Struct	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;
Members	Channel Snapshot channel Quality Image quality. Value range 1-6. The larger the value is, the better the image

Item	Description
	quality is. ImageSize Image size;0:QCIF,1:CIF,2:D1 mode Snapshot Mode -1: stop timing snapshot;0: require one frame; 1: timing send request. 2:Continuous request InterSnap Time interval. The unit: second;if mode=1, device sends out timing request. It is for some special devices such as (mobile device) to use this field to set schedule snapshot interval. We recommend the user uses stuSnapFormat[nSnapMode].stuVideoFormat.nFrameRate of CFG_CMD_ENCODE to realize this function. CmdSerial Snapshot request SN. The value ranges from 0 to 65535. Once the value is out of range, it is unsigned short. Reserved Reserved byte

4.37 DH_VERSION_INFO

Table 4-37 DH_VERSION_INFO

Item	Description
Struct description	Device software version information. The higher 16-bit is main version number and then lower 16-bit is the minor version number.
Struct	typedef struct { DWORD dwSoftwareVersion; DWORD dwSoftwareBuildDate; DWORD dwDspSoftwareVersion; DWORD dwDspSoftwareBuildDate; DWORD dwPanelVersion; DWORD dwPanelSoftwareBuildDate; DWORD dwHardwareVersion; DWORD dwHardwareDate; DWORD dwWebVersion; DWORD dwWebBuildDate; } DH_VERSION_INFO, *LPDH_VERSION_INFO;
Members	dwSoftwareVersion Software Version No. dwSoftwareBuildDate Software Built Version No. dwDspSoftwareVersion DSP Software Version dwDspSoftwareBuildDate DSP Software Built Version dwPanelVersion It is null right now dwPanelSoftwareBuildDate It is null right now dwHardwareVersion Hardware Version dwHardwareDate It is null right now dwWebVersion Web Version dwWebBuildDate Web Built Version No.

4.38 DH_DSP_ENCODECAP

Table 4-38 DH_DSP_ENCODECAP

Item	Description
Struct description	DSP capability description
Struct	typedef struct { DWORD dwVideoStandardMask; DWORD dwImageSizeMask; DWORD dwEncodeModeMask; DWORD dwStreamCap; DWORD dwImageSizeMask_Assi[8]; DWORD dwMaxEncodePower; WORD wMaxSupportChannel; WORD wChannelMaxSetSync; } DH_DSP_ENCODECAP, *LPDH_DSP_ENCODECAP;
Members	dwVideoStandardMask Video format mask. Uses bit to indicate the video format device supported. dwImageSizeMask Resolution mask Uses bit to indicate the resolution device supported. Refer to the following list. 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 14 1920*1200 15 240*192 16 1280*720 17 1920*1080 18 1280*960 19 1872*1408 20 3744*1408 21 2048*1536 22 2432*2050 23 1216*1024 24 1408*1024 25 3296*2472 26 2560*1920(5M) 27 The region setting interface is divided to 960×576(PAL)(NTSC) 28 960 (H) × 720 (V) dwEncodeModeMask

Item	Description
	Compression mode mask bit. Uses bit to indicate the compression mode device supported. dwStreamCap Uses bit to indicate the multi-media function devices supported , The 1st bit :supports main stream, The 2nd bit: supports sub stream1, The 3rd bit: supports sub stream2, The 5th bit : supports snapshot(.JPG) dwImageSizeMask_Assi For main stream resolution, it is the supported mask bit of sub stream resolution dwMaxEncodePower DSP max. supports encode capability wMaxSupportChannel The max. input video channel amount of each DSP wChannelMaxSetSync The max. encode setting of each DSP is synchronized or not. 0: No. 1: Yes

5 Enumeration Definition

5.1 NET_DEVICE_TYPE

Table 5-1 NET_DEVICE_TYPE

Item	Description
Enumeration Description	Device type enumeration. For different device types.

Enumeration Definition	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-channel DVR
	NET_DVR_MPEG1_8, // MPEG1 8-channel DVR
	NET_DVR_MPEG4_8, // MPEG4 8-channel DVR
	NET_DVR_MPEG4_16, // MPEG4 16-channel 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, // The 2nd Network Video Server
	NET_DVR_STD_NEW, // New standard configuration protocol
	NET_DVR_DDNS, // DDNS server
	NET_DVR_ATM, // ATM
	NET_NB_SERIAL, // The 2nd non-real time NB series DVR
	NET_LN_SERIAL, // LN series DVR
	NET_BAV_SERIAL, // BAV series DVR
	NET_SDIP_SERIAL, // SDIP series products
	NET_IPC_SERIAL, // IPC series products
	NET_NVS_B, // NVS B series
	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 by string mode
	NET_NVD_SERIAL, // Network video decoder
	NET_DVR_N5, // N5
	NET_DVR_MIX_DVR, // Hybrid DVR
	NET_SVR_SERIAL, // SVR series
	NET_SVR_BS, // SVR-BS
	NET_NVR_SERIAL, // NVR series

Item	Description
	NET_DVR_N51, // N51 NET_ITSE_SERIAL, // ITSE intelligent analytics box NET_ITC_SERIAL, // ITC (Intelligent Traffic Camera) NET_HWS_SERIAL, // HWS (Radar Speed Measurement Device) NET_PVR_SERIAL, // Portable video recorder NET_IVS_SERIAL, // IVS（Intelligent Video Server） NET_IVS_B, // General intelligent video analytics server NET_IVS_F, // Human recognition device NET_NVS_MPEG1, // Video Quality Diagnosis Server NET_MATRIX_SERIAL, // Matrix NET_DVR_N52, // N52 NET_DVR_N56, // N56 NET_ESS_SERIAL, // ESS NET_IVS_PC, // People Counting Server NET_PC_NVR, // pc-nvr NET_DSCON, // Video wall controller NET_EVS, // Embedded video storage server NET_EIVS, // Embedded intelligent video server NET_DVR_N6, // DVR-N6 NET_UDS, // Universal decoder NET_AF6016, // Bank alarm host NET_AS5008, // Video network alarm server NET_AH2008, // Network alarm server NET_A_SERIAL, // Alarm host series NET_BSC_SERIAL, // Access control series products NET_NVS_SERIAL, // NVS NET_VTO_SERIAL, // VTO NET_VTNC_SERIAL, // VTNC NET_TPC_SERIAL, // TPC (Thermal devices) }NET_DEVICE_TYPE ;

5.2 EM_OPTIMIZE_TYPE

Table 5-2 EM_OPTIMIZE_TYPE

Item	Description
Enumeration Description	Set up internal optimization options to choose different optimization methods.

Item	Description
Enumeration Definition	typedef enum tagEmOptimizeType { EM_OPT_TYPE_DEFAULT = 0, /// Not optimized by default EM_OPT_TYPE_MOBILE_V1 = 1, /// Conflicts with EM_OPT_TYPE_MOBILE_OPTION; only one can be used. Asynchronous login optimization is not supported. When mobile optimization is enabled, it will filter device attributes and hard disk information by default. EM_OPT_TYPE_P2P_NETPARAM_V1 = 2, /// Configure P2P network parameters. pParam is NET_PARAM* EM_OPT_TYPE_MOBILE_OPTION = 3, /// Conflicts with EM_OPT_TYPE_MOBILE_V1; only one can be used. Asynchronous login optimization is supported. It is a mobile optimization option with parameter int, specified as a bitmask ranging from low to high. The input values come from the enumeration type EM_OPTTYPE_MOBILE_TYPE. EM_OPT_TYPE_LOGIN_GET_ENCRYPT = 4, /// Get key information during login to reduce stream loading time. pParam is int*. Use 1 to enable optimization. }EM_OPTIMIZE_TYPE;

5.3 EM_LOGIN_SPAC_CAP_TYPE

Table 5-3 EM_LOGIN_SPAC_CAP_TYPE

Item	Description
Enumeration Description	Login mode enumeration description. To select different login mode.

Enumeration Definition	typedef enum tagEM_LOGIN_SPAC_CAP_TYPE
	{
	EM_LOGIN_SPEC_CAP_TCP= 0, // TCP, default mode
	EM_LOGIN_SPEC_CAP_ANY = 1, // Login unconditionally
	EM_LOGIN_SPEC_CAP_SERVER_CONN = 2, // Login of auto registration
	EM_LOGIN_SPEC_CAP_MULTICAST = 3, // Multicast login, default
	EM_LOGIN_SPEC_CAP_UDP= 4, // UDP login
	EM_LOGIN_SPEC_CAP_MAIN_CONN_ONLY= 6,
	// Only main connection
	EM_LOGIN_SPEC_CAP_SSL= 7, // SSL encryption mode login
	EM_LOGIN_SPEC_CAP_INTELLIGENT_BOX= 9,
	// Log in to the smart box device
	EM_LOGIN_SPEC_CAP_NO_CONFIG= 10,
	// Do not get configuration after login device
	EM_LOGIN_SPEC_CAP_U_LOGIN= 11, // Login by USB key
	EM_LOGIN_SPEC_CAP_LDAP= 12, // Login by LDAP
	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, // The 2nd verification login
	EM_LOGIN_SPEC_CAP_TS = 18, // TS bit stream client login
	EM_LOGIN_SPEC_CAP_P2P = 19, // P2P login
	EM_LOGIN_SPEC_CAP_MOBILE= 20, // Cellphone client login
	EM_LOGIN_SPEC_CAP_INVALID// Invalid login
	}EM_LOGIN_SPAC_CAP_TYPE;

5.4 DH_RealPlayType

Table 5-4 DH_RealPlayType

Item	Description
Enumeration Description	Live view type. Corresponding value of CLIENT_RealPlayEx

Enumeration Definition	typedef enum _RealPlayType
	{
	DH_RType_Realplay = 0, // real-time live view
	DH_RType_Multiplay,, //Multi-screen Preview
	DH_RType_Realplay_0 , / / Real-time monitoring—main stream, equivalent to DH_RType_Realplay
	DH_RType_Realplay_1 , / / Real-time monitoring—sub stream 1
	DH_RType_Realplay_2 , / / Real-time monitoring—sub stream 2
	DH_RType_Realplay_3 , / / Real-time monitoring—sub stream 3
	DH_RType_Multiplay_1 , / / Multi-picture live view—1-window
	DH_RType_Multiplay_4 , / / Multi-picture live view—4-window
	DH_RType_Multiplay_8 , / / Multi-picture live view—8-window
	DH_RType_Multiplay_9 , / / Multi-picture live view—9-window
	DH_RType_Multiplay_16 , / / Multi-picture live view—16-window
	DH_RType_Multiplay_6 , / / Multi-picture live view—6-window
	DH_RType_Multiplay_12 , / / Multi-picture live view—12-window
	DH_RType_Multiplay_25 , / / Multi-picture live view—25-window
	DH_RType_Multiplay_36 , / / Multi-picture live view—36-window
	} DH_RealPlayType;

5.5 EM_QUERY_RECORD_TYPE

Table 5-5 EM_QUERY_RECORD_TYPE

Item	Description
Enumeration Description	Record search type

Enumeration Definition	typedef enum tagEmQueryRecordType
	{
	EM_RECORD_TYPE_ALL = 0, // All records
	EM_RECORD_TYPE_ALARM = 1, // External alarm record
	DPSDK_RECORD_TYPE_ALARM = 2, // Motion detect alarm record
	EM_RECORD_TYPE_ALARM_ALL = 3, // All alarm records
	EM_RECORD_TYPE_CARD = 4, // Search card number
	EM_RECORD_TYPE_CONDITION = 5, // Search by criteria
	EM_RECORD_TYPE_JOIN = 6, // Combined search
	EM_RECORD_TYPE_CARD_PICTURE = 8,
	// Search picture by card number. For HB-U, NVS and so on.
	EM_RECORD_TYPE_PICTURE = 9, // Search picture. For HB-U, NVS and so on.
	EM_RECORD_TYPE_FIELD = 10, // Search by field.
	EM_RECORD_TYPE_INTELLI_VIDEO = 11, // Search IVS record
	EM_RECORD_TYPE_TRANS_DATA = 16, // Search transparent serial port record
	EM_RECORD_TYPE_IMPORTANT = 17,,// Search important record files
	EM_RECORD_TYPE_TALK_DATA = 18, // Search audio file
	EM_RECORD_TYPE_INVALID = 256, // Invalid search type
	}EM_QUERY_RECORD_TYPE;

5.6 EM_USEDEV_MODE

Table 5-6 EM_USEDEV_MODE

Item	Description
Enumeration Description	Device working mode type (Some modes are not included in this manual so there is no corresponding note about the extension data type)

Enumeration Definition	typedef enum __EM_USEDEV_MODE
	{
	DH_TALK_CLIENT_MODE,
	// Set to use client-end mode to begin audio talk (The extension data is NULL)
	DH_TALK_SERVER_MODE,
	// Set to use server mode to begin audio talk (The extension data is NULL)
	DH_TALK_ENCODE_TYPE,
	/ / Configure the encode format of audio talk (The extension data is DHDEV_TALKDECODE_INFO*)
	DH_ALARM_LISTEN_MODE,
	// Set alarm subscription mode (The extension data is NULL))
	DH_CONFIG_AUTHORITY_MODE,
	// Set to use right to realize configuration management(The extension data is NULL)
	DH_TALK_TALK_CHANNEL,
	// Set audio talk channel. (The extension data is int*, pointer address is 0~MaxChannel-1)
	DH_RECORD_STREAM_TYPE,
	// Set the record bit stream type of the file to be searched and file searching by time (The extension data is int*,pointer address is 0-main stream/sub stream,1-main stream,2-sub stream)
	DH_TALK_SPEAK_PARAM,
	/ / Set broadcast parameters of audio talk
	DH_RECORD_TYPE,
	// Set record file type of the file play and download by time (Refer to NET_RECORD_TYPE)
	DH_TALK_MODE3
	/ / Set audio talk parameters of the third-generation devices（The extension data is ???）
	DH_PLAYBACK_REALTIME_MODE ,
	// Set real-time playback function （ The extension data is int*,pointer address :0-Disable,1-Enable）
	DH_TALK_TRANSFER_MODE,
	// Set audio talk is transfer mode or not (The extension data is NET_TALK_TRANSFER_PARAM*)
	DH_TALK_VT_PARAM,
	//Set VT audio talk parameters, corresponding structure is NET_VT_TALK_PARAM
	DH_TARGET_DEV_ID,
	//Set object device identifier,to search new system capability (Not 0-Transfer system capability message)
	} EM_USEDEV_MODE;

5.7 EM_SUPPORT_FOCUS_MODE

Table 5-7 EM_SUPPORT_FOCUS_MODE

Item	Description
Enumeration Description	The enumeration of the supported focus mode
Enumeration Definition	typedef enum tagSUPPORT_FOCUS_MODE { ENUM_SUPPORT_FOCUS_CAR= 1,// Focus on card mode ENUM_SUPPORT_FOCUS_PLATE= 2,// Focus on plate number mode ENUM_SUPPORT_FOCUS_PEOPLE= 3,// Focus on human mode ENUM_SUPPORT_FOCUS_FACE= 4,// Focus on human face

Item	Description
	}EM_SUPPORT_FOCUS_MODE;

5.8 DH_PTZ_ControlType

Table 5-8 DH_PTZ_ControlType

Item	Description
Enumeration Description	General PTZ control commands enumeration

Enumeration Definition	typedef enum _PTZ_ControlType
	{
	DH_PTZ_UP_CONTROL = 0, // Up,lParam2:pan/tilt movement speed. Valid range (1-8)
	DH_PTZ_DOWN_CONTROL, // Down,lParam2:pan/tilt movement speed. Valid range(1-8)
	DH_PTZ_LEFT_CONTROL, // Left,lParam2:pan/tilt movement speed. Valid range (1-8)
	DH_PTZ_RIGHT_CONTROL, // Right,lParam2:pan/tilt movement speed. Valid range(1-8)
	DH_PTZ_ZOOM_ADD_CONTROL, // Zoom+,lParam2:speed,Valid range(1-8)
	DH_PTZ_ZOOM_DEC_CONTROL, // Zoom-,lParam2:speed,Valid range(1-8)
	DH_PTZ_FOCUS_ADD_CONTROL, // Focus+,lParam2:speed,Valid range(1-8)
	DH_PTZ_FOCUS_DEC_CONTROL, // Focus-,lParam2:speed,Valid range(1-8)
	DH_PTZ_APERTURE_ADD_CONTROL, // Iris +,lParam2:speed,Valid range(1-8)
	DH_PTZ_APERTURE_DEC_CONTROL, // Iris-,lParam2:speed,Valid range(1-8)
	DH_PTZ_POINT_MOVE_CONTROL, // Go to preset,lParam2:Preset No.
	DH_PTZ_POINT_SET_CONTROL, // Set,lParam2:Preset No.
	DH_PTZ_POINT_DEL_CONTROL, // Delete,lParam2:Preset No.
	DH_PTZ_POINT_LOOP_CONTROL, // Tour,lParam1:Tour path,lParam3:76 start;96 stop
	DH_PTZ_LAMP_CONTROL// Light and wiper,lParam1:On-off control,1:Enable,0:Disable
	} DH_PTZ_ControlType;

5.9 DH_EXTPTZ_ControlType

Table 5-9 DH_EXTPTZ_ControlType

Item	Description
Enumeration Description	PTZ control extension commands
Enumeration 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,// Adds a preset to tour, lParam1: tour No.;lParam2: preset No. DH_EXTPTZ_DELFROMLOOP,// Deletes a preset from the tour,lParam1:tour No.;lParam2: preset No. DH_EXTPTZ_CLOSELOOP,// Delete a tour. lParam1: tour No. DH_EXTPTZ_STARTPANCRUISE,//Begin pan rotation DH_EXTPTZ_STOPPANCRUISE,// Stop pan rotation DH_EXTPTZ_SETLEFTBORDER,// Set left limit DH_EXTPTZ_SETRIGHTBORDER,// Set right limit DH_EXTPTZ_STARTLINESCAN,// Start scanning DH_EXTPTZ_CLOSELINESCAN,// Stop scanning

Item	Description

	DH_EXTPTZ_SETMODESTART,// Start mode Mode line
	DH_EXTPTZ_SETMODESTOP,// Stop mode Mode line
	DH_EXTPTZ_RUNMODE,// Running mode Mode line
	DH_EXTPTZ_STOPMODE,// Stop mode Mode line
	DH_EXTPTZ_DELETEMODE,// Clear mode Mode line
	DH_EXTPTZ_REVERSECOMM,// Flip command
	DH_EXTPTZ_FASTGOTO,// Fast positioning lParam1:Horizontal coordinates,valid range(-8191 ~ 8191);lParam2:vertical coordinates,valid range (-8191 ~ 8191);lParam3:zoom,valid range(-16 ~ 16)
	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 triggers PTZ parm1:Alarm input channel;parm2:Alarm trigger type 1-preset 2-scan 3-tour;parm3:trigger 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 accurate 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 structure PTZ_CONTROL_ABSOLUTELY
	DH_EXTPTZ_MOVE_CONTINUOUSLY,
	// Continuous motion control commands,param4 corresponding structure PTZ_CONTROL_CONTINUOUSLY
	DH_EXTPTZ_GOTOPRESET,
	// PTZ control commands, at a certain speed to go to a preset ,parm4 corresponding structure //PTZ_CONTROL_GOTOPRESET
	DH_EXTPTZ_SET_VIEW_RANGE = 0x49,
	// Set visual field (param4 corresponding structure PTZ_VIEW_RANGE_INFO)
	DH_EXTPTZ_FOCUS_ABSOLUTELY = 0x4A,
	// Absolute focus (param4 corresponding structure PTZ_FOCUS_ABSOLUTELY)
	DH_EXTPTZ_HORSECTORSCAN = 0x4B,
	// Horizontal scan (param4 corresponding structure PTZ_CONTROL_SECTORSCAN,param1, param2, param3 are null)
	DH_EXTPTZ_VERSECTORSCAN = 0x4C,
	// Vertical scan (param4 corresponding structure PTZ_CONTROL_SECTORSCAN,param1, param2, param3 are null)
	DH_EXTPTZ_SET_ABS_ZOOMFOCUS = 0x4D,
	// Set absolute focus distance, focus value,param1 is focus distance,range:[0,255],param2 is focus,range :[0,255],param3, param4 are null.
	DH_EXTPTZ_SET_FISHEYE_EPTZ = 0x4E,
	// Control fish eye e-PTZ,param4 corresponding structure PTZ_CONTROL_SET_FISHEYE_EPTZ
	DH_EXTPTZ_UP_TELE = 0x70,// up + TELE param1=step (1-8). similarly hereinafter
	DH_EXTPTZ_DOWN_TELE,// Down + TELE
	DH_EXTPTZ_LEFT_TELE,// Left + TELE
	DH_EXTPTZ_RIGHT_TELE,// Right + TELE

Item	Description
	DH_EXTPTZ_LEFTUP_TELE,// Upper left + TELE 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=step (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;

5.10 DH_TALK_CODING_TYPE

Table 5-10 DH_TALK_CODING_TYPE

Item	Description
Enumeration Description	Audio Encode Type
Enumeration Definition	typedef enum __TALK_CODING_TYPE { DH_TALK_DEFAULT = 0,// No-head PCM DH_TALK_PCM = 1,// PCM with head 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 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;

5.11 CtrlType

Table 5-11 CtrlType

Item	Description
Enumeration Description	Device control type. Corresponding to interface CLIENT_ControlDeviceEx

Enumeration Definition	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,

Item	Description

	DH_KEYBOARD_DOWN,
	DH_KEYBOARD_LEFT,
	DH_KEYBOARD_RIGHT,
	DH_KEYBOARD_BTN0,
	DH_KEYBOARD_BTN1,
	DH_KEYBOARD_BTN2,
	DH_KEYBOARD_BTN3,
	DH_KEYBOARD_BTN4,
	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,
	DH_KEYBOARD_SHIFT,
	DH_KEYBOARD_BACK,
	DH_KEYBOARD_LOGIN , // New network keyboard functions
	DH_KEYBOARD_CHNNEL , // Switch video channel
	DH_TRIGGER_ALARM_IN = 100, // Trigger alarm input
	DH_TRIGGER_ALARM_OUT, // Trigger alarm output
	DH_CTRL_MATRIX, // Matrix control
	DH_CTRL_SDCARD, // SD card control (IPC products) Parameters are the same as that of the HDD control.
	DH_BURNING_START, // Burner control, start burning
	DH_BURNING_STOP, // Burner control, stop burning
	DH_BURNING_ADDPWD, // Burner control, overlay password (String ended with '\0'. Max length is 8-bit)
	DH_BURNING_ADDHEAD, // Burner control, overlay title (String ended with '\0'. Max length is 1024-bit. 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, //Trigger device to snapshot
	DH_CTRL_CLEARLOG, // Clear log
	DH_TRIGGER_ALARM_WIRELESS = 200, // Trigger wireless

Item	Description
	alarm (IPC series) DH_MARK_IMPORTANT_RECORD, // Mark important record DH_CTRL_DISK_SUBAREA, // Network hard disk partition DH_BURNING_ATTACH, // Burner control, burn the attachment DH_BURNING_PAUSE, // Pause burn DH_BURNING_CONTINUE, // Resume burn DH_BURNING_POSTPONE, //Postpone burn DH_CTRL_OEMCTRL, // OEM control DH_BACKUP_START, // Device starts backing up DH_BACKUP_STOP, // Device stops backing up DH_VIHICLE_WIFI_ADD, // Manually adds Wi-Fi configuration for mobile devices DH_VIHICLE_WIFI_DEC, // Manually deletes Wi-Fi configuration for mobile devices 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, // Intelligent traffic,wiper control DH_MANUAL_SNAP, // Intelligent traffic, manual snapshot (MANUAL_SNAP_PARAMETER) DH_MANUAL_NTP_TIMEADJUST, // Manual NTP DH_NAVIGATION_SMS, // Navigation info and message DH_CTRL_ROUTE_CROSSING, // Route info DH_BACKUP_FORMAT, // Format backup device DH_DEVICE_LOCALPREVIEW_SLIPT, // Control local live view split (DEVICE_LOCALPREVIEW_SLIPT_PARAMETER) DH_CTRL_INIT_RAID, // RAID initialization DH_CTRL_RAID, // RAID operation DH_CTRL_SAPREDISK, // Hotspare operation DH_WIFI_CONNECT, // Manually start Wi-Fi connection (WIFI_CONNECT) DH_WIFI_DISCONNECT, // Manually stop Wi-Fi connection (WIFI_CONNECT) DH_CTRL_ARMED, //Arm and disarm operation DH_CTRL_IP_MODIFY, // Modify front-end IP (DHCTRL_IPMODIFY_PARAM) DH_CTRL_WIFI_BY_WPS, // wps connects Wi-Fi (DHCTRL_CONNECT_WIFI_BYWPS) DH_CTRL_FORMAT_PATITION, // Format partition(DH_FORMAT_PATITION) DH_CTRL_EJECT_STORAGE, // Manually eject device (DH_EJECT_STORAGE_DEVICE) DH_CTRL_LOAD_STORAGE, // Manually load device(DH_LOAD_STORAGE_DEVICE) DH_CTRL_CLOSE_BURNER, // Close burner(NET_CTRL_BURNERDOOR) Usually waits for 6 seconds. DH_CTRL_EJECT_BURNER, // Eject burner(NET_CTRL_BURNERDOOR) Usually waits for 4 seconds. DH_CTRL_CLEAR_ALARM, // Clear alarm (NET_CTRL_CLEAR_ALARM) DH_CTRL_MONITORWALL_TVINFO, // TV wall information display(NET_CTRL_MONITORWALL_TVINFO) DH_CTRL_START_VIDEO_ANALYSE, // Start intelligent video analytics(NET_CTRL_START_VIDEO_ANALYSE) DH_CTRL_STOP_VIDEO_ANALYSE, // Stop intelligent video analytics(NET_CTRL_STOP_VIDEO_ANALYSE) DH_CTRL_UPGRADE_DEVICE, // Control and start device upgrade. Device completes upgrade independently. No need to transmit upgrade file. DH_CTRL_MULTIPLAYBACK_CHANNALES, // Switch the playback channel of the multi-channel live view (NET_CTRL_MULTIPLAYBACK_CHANNALES)

Item	Description

	DH_CTRL_SEQPOWER_OPEN, // Power sequencer enables on-off output port (NET_CTRL_SEQPOWER_PARAM)
	DH_CTRL_SEQPOWER_CLOSE, // Power sequencer disables on-off output port(NET_CTRL_SEQPOWER_PARAM)
	DH_CTRL_SEQPOWER_OPEN_ALL, // Power sequencer enables on-off output group port (NET_CTRL_SEQPOWER_PARAM)
	DH_CTRL_SEQPOWER_CLOSE_ALL, // Power sequencer disables on-off output group port (NET_CTRL_SEQPOWER_PARAM)
	DH_CTRL_PROJECTOR_RISE, // Project up(NET_CTRL_PROJECTOR_PARAM)
	DH_CTRL_PROJECTOR_FALL, // Project down (NET_CTRL_PROJECTOR_PARAM)
	DH_CTRL_PROJECTOR_STOP, // Project stop(NET_CTRL_PROJECTOR_PARAM)
	DH_CTRL_INFRARED_KEY, // IR button (NET_CTRL_INFRARED_KEY_PARAM)
	DH_CTRL_START_PLAYAUDIO, // Device starts playing audio file (NET_CTRL_START_PLAYAUDIO)
	DH_CTRL_STOP_PLAYAUDIO, // Device stops playing audio file
	DH_CTRL_START_ALARMBELL, // Enable siren (Corresponding structure NET_CTRL_ALARMBELL)
	DH_CTRL_STOP_ALARMBELL, // Disable siren (Corresponding structure NET_CTRL_ALARMBELL)
	DH_CTRL_ACCESS_OPEN, // A&C control-open door (Corresponding structure NET_CTRL_ACCESS_OPEN)
	DH_CTRL_SET_BYPASS, // Set bypass function(Corresponding structure NET_CTRL_SET_BYPASS)
	DH_CTRL_RECORDSET_INSERT, // Add records, get record set number (Corresponding structureNET_CTRL_RECORDSET_INSERT_PARAM)
	DH_CTRL_RECORDSET_UPDATE, // Update a record of the number set (Corresponding structure NET_CTRL_RECORDSET_PARAM)
	DH_CTRL_RECORDSET_REMOVE, // According to the record set number to delete a record (Corresponding structure NET_CTRL_RECORDSET_PARAM)
	DH_CTRL_RECORDSET_CLEAR, // Remove all record set information(Corresponding structure NET_CTRL_RECORDSET_PARAM)
	DH_CTRL_ACCESS_CLOSE, // A&C control-close door (Corresponding structure NET_CTRL_ACCESS_CLOSE)
	DH_CTRL_ALARM_SUBSYSTEM_ACTIVE_SET, // Alarm sub system activation setup (Corresponding structureNET_CTRL_ALARM_SUBSYSTEM_SETACTIVE)
	DH_CTRL_FORBID_OPEN_STROBE, // Disable device open gateway(Corresponding structure NET_CTRL_FORBID_OPEN_STROBE)
	DH_CTRL_OPEN_STROBE, // Enable gateway (Corresponding structure NET_CTRL_OPEN_STROBE)
	DH_CTRL_TALKING_REFUSE, // The audio talk rejects to answer(Corresponding structure NET_CTRL_TALKING_REFUSE)
	DH_CTRL_ARMED_EX, // Arm/disarm operation(Corresponding structure CTRL_ARM_DISARM_PARAM_EX),upgrade CTRL_ARM_DISARM_PARAM. Recommended.
	DH_CTRL_NET_KEYBOARD = 400, // Net keyboard control(Corresponding structure DHCTRL_NET_KEYBOARD)
	DH_CTRL_AIRCONDITION_OPEN, // Open air conditioner (Corresponding structure NET_CTRL_OPEN_AIRCONDITION)
	DH_CTRL_AIRCONDITION_CLOSE, // Close air-conditioner (Corresponding structureNET_CTRL_CLOSE_AIRCONDITION)
	DH_CTRL_AIRCONDITION_SET_TEMPERATURE, // Set air-conditioner temperature(Corresponding structureNET_CTRL_SET_TEMPERATURE)
	DH_CTRL_AIRCONDITION_ADJUST_TEMPERATURE, // Adjust air-conditioner temperature(Corresponding structure NET_CTRL_ADJUST_TEMPERATURE)
	DH_CTRL_AIRCONDITION_SETMODE, // Set air-conditioner work mode

Item	Description

	(Corresponding structure NET_CTRL_ADJUST_TEMPERATURE)
	DH_CTRL_AIRCONDITION_SETWINDMODE, // Set air-conditioner blow-in mode(Corresponding structure NET_CTRL_AIRCONDITION_SETMODE)
	DH_CTRL_RESTOREDEFAULT_EX , // New protocol to reset device default setup (Corresponding structure NET_CTRL_RESTORE_DEFAULT)
	// If port failed,first use this enumeration to recover setup.
	// CLIENT_GetLastError returns NET_UNSUPPORTED, and then try to use DH_CTRL_RESTOREDEFAULT to reover setup.
	DH_CTRL_NOTIFY_EVENT, // Sends event to device(Corresponding structure NET_NOTIFY_EVENT_DATA)
	DH_CTRL_SILENT_ALARM_SET, // Mute alarm setup
	DH_CTRL_START_PLAYAUDIOEX, // Device starts audio broadcast (Corresponding structure NET_CTRL_START_PLAYAUDIOEX)
	DH_CTRL_STOP_PLAYAUDIOEX, // Device stops audio broadcast
	DH_CTRL_CLOSE_STROBE, // Close gateway (Corresponding structure NET_CTRL_CLOSE_STROBE)
	DH_CTRL_SET_ORDER_STATE, // Set parking reservation status (Corresponding structure NET_CTRL_SET_ORDER_STATE)
	DH_CTRL_RECORDSET_INSERTEX, // Add record,get record set number (Corresponding structure NET_CTRL_RECORDSET_INSERT_PARAM)
	DH_CTRL_RECORDSET_UPDATEEX, // Upgrade the record of one record set number (Corresponding structure NET_CTRL_RECORDSET_PARAM)
	DH_CTRL_CAPTURE_FINGER_PRINT, // Fingerprint collection (Corresponding structure NET_CTRL_CAPTURE_FINGER_PRINT)
	DH_CTRL_ECK_LED_SET, // Parking lot entrance/exit controller LED setup(Corresponding structure NET_CTRL_ECK_LED_SET_PARAM)
	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)
	DH_CTRL_ECK_SYNC_IC_CARD, // Intelligent parking system in/out device IC card info sync command. After received this command, device will delete original IC card info (Corresponding structure NET_CTRL_ECK_SYNC_IC_CARD_PARAM)
	DH_CTRL_LOWRATEWPAN_REMOVE, // Delete specific wireless device(Corresponding structure NET_CTRL_LOWRATEWPAN_REMOVE)
	DH_CTRL_LOWRATEWPAN_MODIFY, // Modify wireless device info (Corresponding structure NET_CTRL_LOWRATEWPAN_MODIFY)
	DH_CTRL_ECK_SET_PARK_INFO, // Set up the vehicle spot information of the machine at the passageway of the intelligent parking system (Corresponding structure NET_CTRL_ECK_SET_PARK_INFO_PARAM)
	DH_CTRL_VTP_DISCONNECT, // Hang up the video phone (Corresponding structure NET_CTRL_VTP_DISCONNECT)
	DH_CTRL_UPDATE_FILES, // Update the multimedia files remotely (Corresponding structure NET_CTRL_UPDATE_FILES)
	DH_CTRL_MATRIX_SAVE_SWITCH, // Saves up the relationship between the hyponymy matrices (Corresponding structure NET_CTRL_MATRIX_SAVE_SWITCH)
	DH_CTRL_MATRIX_RESTORE_SWITCH, // Recover the relationship between the hyponymy matrices (Corresponding structure NET_CTRL_MATRIX_RESTORE_SWITCH)
	DH_CTRL_VTP_DIVERTACK, // Calls and transfers respond (Corresponding structure NET_CTRL_VTP_DIVERTACK)
	DH_CTRL_RAINBRUSH_MOVEONCE, // Wiper moves back and forth for once . It is valid when wiper is in manual mode. (Corresponding structure NET_CTRL_RAINBRUSH_MOVEONCE)
	DH_CTRL_RAINBRUSH_MOVECONTINUOUSLY, // Wiper moves back and forth continuously. It is valid when wiper is in manual mode. (Corresponding structure NET_CTRL_RAINBRUSH_MOVECONTINUOUSLY)
	DH_CTRL_RAINBRUSH_STOPMOVE, // Wiper stops. It is valid when wiper is in manual mode (Corresponding structure NET_CTRL_RAINBRUSH_STOPMOVE)
	DH_CTRL_ALARM_ACK, // Confirm alarm event (Corresponding

Item	Description

	structure NET_CTRL_ALARM_ACK)
	// DH_CTRL_ALARM_ACK DO NOT call this function in alarm callback interface
	DH_CTRL_RECORDSET_IMPORT, // Batch import record set info (Corresponding structure NET_CTRL_RECORDSET_PARAM)
	DH_CTRL_ACCESS_USE_DOOR, // Disable and enable door (Corresponding 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 structure NET_CTRL_ACCESS_SHUT_LOCK)
	DH_CTRL_OPEN_DOOR_CONTINUE, // Continuous unlocking instruction(Corresponding structure NET_CTRL_OPEN_DOOR_CONTINUE)
	// The following commands are only for CLIENT_ControlDeviceEx
	DH_CTRL_THERMO_GRAPHY_ENSHUTTER = 0x10000, // Set to enable or disable thermal shutter,, pInBuf= NET_IN_THERMO_EN_SHUTTER*, pOutBuf= NET_OUT_THERMO_EN_SHUTTER *
	DH_CTRL_RADIOMETRY_SETOSDMARK, // Set the OSD of the detected object as highlighted, pInBuf= 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 *
	DH_CTRL_SNAP_MNG_SNAP_SHOT, // Instant snapshot(Manual snapshot), pInBuf = NET_IN_SNAP_MNG_SHOT *, pOutBuf = NET_OUT_SNAP_MNG_SHOT *
	DH_CTRL_LOG_STOP, // Forcedly sync buffer data to the database and close the database, pInBuf = NET_IN_LOG_MNG_CTRL *, pOutBuf = NET_OUT_LOG_MNG_CTRL *
	DH_CTRL_LOG_RESUME, // Recover database, pInBuf = NET_IN_LOG_MNG_CTRL *, pOutBuf = NET_OUT_LOG_MNG_CTRL *
	DH_CTRL_POS_ADD, // Add a POS device, pInBuf = NET_IN_POS_ADD *, pOutBuf = NET_OUT_POS_ADD *
	DH_CTRL_POS_REMOVE, // Delete a POS device, pInBuf = NET_IN_POS_REMOVE *, pOutBuf = NET_OUT_POS_REMOVE *
	DH_CTRL_POS_REMOVE_MULTI, // Batch deletes POS devices, pInBuf = NET_IN_POS_REMOVE_MULTI *, pOutBuf = NET_OUT_POS_REMOVE_MULTI *
	DH_CTRL_POS_MODIFY, // Modify a POS device, pInBuf = NET_IN_POS_ADD *, pOutBuf = NET_OUT_POS_ADD *
	DH_CTRL_SET_SOUND_ALARM, // /Trigger an alarm with sound, pInBuf = NET_IN_SOUND_ALARM *, pOutBuf = NET_OUT_SOUND_ALARM *
	DH_CTRL_AUDIO_MATRIX_SILENCE, // Audio deposition and one-click mute control (Corresponding pInBuf = NET_IN_AUDIO_MATRIX_SILENCE, pOutBuf = NET_OUT_AUDIO_MATRIX_SILENCE)
	DH_CTRL_MANUAL_UPLOAD_PICTURE, // Set manual upload, 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;

5.12 CFG_VIDEO_COMPRESSION

Table 5-12 CFG_VIDEO_COMPRESSION

Item	Description
Enumeration Description	Video compression format description
Enumeration Definition	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_MJPG, // MJPG VIDEO_FORMAT_FCC_MPEG4, // FCC-MPEG4 VIDEO_FORMAT_H264, // H.264 VIDEO_FORMAT_H265, // H.265 } CFG_VIDEO_COMPRESSION;

5.13 CFG_BITRATE_CONTROL

Table 5-13 CFG_BITRATE_CONTROL

Item	Description
Enumeration Description	Bit rate control mode
Enumeration Definition	typedef enum tagCFG_BITRATE_CONTROL { BITRATE_CBR, // constant bit stream BITRATE_VBR, // Variable bit stream } CFG_BITRATE_CONTROL;

5.14 CFG_IMAGE_QUALITY

Table 5-14 CFG_IMAGE_QUALITY

Item	Description
Enumeration Description	Quality type

Enumeration Definition	typedef enum tagCFG_IMAGE_QUALITY
	{
	IMAGE_QUALITY_Q10 = 1, // Picture quality 10%
	IMAGE_QUALITY_Q30, // Picture quality 30%
	IMAGE_QUALITY_Q50, // Picture quality 50%
	IMAGE_QUALITY_Q60, // Picture quality 60%
	IMAGE_QUALITY_Q80, // Picture quality 80%
	IMAGE_QUALITY_Q100, // Picture quality 100%
	} CFG_IMAGE_QUALITY;

5.15 CFG_H264_PROFILE_RANK

Table 5-15 CFG_H264_PROFILE_RANK

Item	Description
Enumeration Description	H.264 encode level

Enumeration Definition	typedef enum tagCFG_H264_PROFILE_RANK
	{
	PROFILE_BASELINE = 1, // Provides I/P Frame,only support progressive scanning and CAVLC
	PROFILE_MAIN, // Provides I/P/B Frame,support progressive and

Item	Description
	interlaced,provide CAVLC and CABAC PROFILE_EXTENDED, // Provide I/P/B/SP/SI Frame, only support progressive scanning and CAVLC PROFILE_HIGH, // /Based on FRExt,Main_Profile, new add:8x8 intra prediction(8x8 intra-frame prediction), custom quant(customized quantization), lossless video coding(No-loss video encode), more yuv format }CFG_H264_PROFILE_RANK;

5.16 CFG_AUDIO_FORMAT

Table 5-16 CFG_AUDIO_FORMAT

Item	Description
Enumeration Description	Audio encode mode
Enumeration Definition	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;

5.17 EM_SEND_SEARCH_TYPE

Table 5-17 EM_SEND_SEARCH_TYPE

Item	Description
Enumeration Description	Send search type
Enumeration Definition	typedef enum tagEM_SEND_SEARCH_TYPE { EM_SEND_SEARCH_TYPE_MULTICAST_AND_BROADCAST, // Search by multicast and broadcast. EM_SEND_SEARCH_TYPE_MULTICAST, // Multicast search EM_SEND_SEARCH_TYPE_BROADCAST, // Broadcast. search }EM_SEND_SEARCH_TYPE;

5.18 EM_REALPLAY_DISCONNECT_EVENT_TYPE

Table 5-18 EM_REALPLAY_DISCONNECT_EVENT_TYPE

Item	Description
Enumeration Description	Video monitor offline event type
Enumeration Definition	typedef enum _EM_REALPLAY_DISCONNECT_EVENT_TYPE { DISCONNECT_EVENT_REAVE, // The user of the high-level takes the resources of the user of the low-level. DISCONNECT_EVENT_NETFORBID, // Forbid connection DISCONNECT_EVENT_SUBCONNECT, // Dynamic sub-connection offline }EM_REALPLAY_DISCONNECT_EVENT_TYPE;

6 Interface Function Definition

6.1 CLIENT_Init

Table 6-1 CLIENT_Init

Item	Description
Interface description	SDK initialization interface. Call it when initializing program.
Pre-condition	None
Function	BOOL CLIENT_Init(fDisConnect cbDisConnect, LDWORD dwUser);
Parameter	cbDisConnect *[In]* Offline callback function. When the on line device gets disconnected,SDK will notify user by call this function. The 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. dwUser *[in]* User data,when callback function is not 0,SDK will call fDisConnect to return the data to user for following operation.
Return value	Return TRUE for success, and return FALSE for failure.

Use examples	It's not recommended to call SDK interface in callback function, unless call CLIENT_GetLastError to get error code of current process.
	//Device disconnection callback function
	// When the device gets offline,SDK will call this callback function. Go to CLINET_Init to set the callback function.
	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);

Item	Description

	}
	printf("nDVRPort[%d]\n", nDVRPort);
	printf("dwUser[%p]\n", dwUser);
	printf("\n");
	}
	**********Above are callback function definition, the underneath are interface using examples**********
	//Initialize SDK
	g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
	if (FALSE == g_bNetSDKInitFlag)
	{
	printf("Initialize client SDK failed; \n");
	return;
	}
	else
	{
	printf("Initialize client SDK done; \n");
	}

Note	Before call other SDK interface, call this interface first. If call this interface repeatedly,the first time is valid.

6.2 CLIENT_Cleanup

Table 6-2 CLIENT_Cleanup

Item	Description
Interface description	SDK cleaning up interface
Pre-condition	Already called initialization interface CLIENT_Init
Function	void CLIENT_Cleanup( );
Parameter	None
Return value	None

Use examples	// Clean initialization resources
	printf("CLIENT_Cleanup!\n");
	CLIENT_Cleanup();

Note	When application program is closed, call this interface to release resources at last.

6.3 CLIENT_GetSDKVersion

Table 6-3 CLIENT_GetSDKVersion

Item	Description
Interface description	The interface to get the version information of SDK
Pre-condition	Already called initialization interface CLIENT_Init
Function	DWORD CLIENT_GetSDKVersion( );
Parameter	None
Return value	Return value is version,for example 34219000 corresponding to version 3.42 19000.
Use examples	/ /Get SDK version info DWORD dwNetSdkVersion = CLIENT_GetSDKVersion(); printf("NetSDK version is [%d]\n", dwNetSdkVersion);
Note	None

6.4 CLIENT_GetLastError

Table 6-4 CLIENT_GetLastError

Item	Description
Interface description	Interface to get error code,get current thread error code.
Pre-condition	Already called initialization interface CLIENT_Init
Function	DWORD CLIENT_GetLastError(void);
Parameter	None
Return value	Current thread error code

Use examples	Solution 1:
	Print the error code in hexadecimal format. Search for the hexadecimal value in dhnetsdk.h to find the corresponding explanation.
	// For example,
	printf("Last Error[%x]\n" , CLIENT_GetLastError());
	The error code is 0x80000017. Search for corresponding NET_NOT_SUPPORTED in the dhnetsdk.h header file.
	Solution 2:
	// According to error code, user can find corresponding explanation in dhnetsdk.h.It is to print hexadecimal here, not decimal shows in header file, be careful with conversion.
	For 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("Last Error[%x]\n" , CLIENT_GetLastError());

Note	Call this interface after failed to call thread SDK interface. There is too much error code, so it is impossible to illustrate one by one here. User can search the following fields in dhnetsdk.h: // Error type code, corresponds with return value of CLIENT_GetLastError interface. #define _EC(x) (0x80000000|x) To find instruction of corresponding error code.

6.5 CLIENT_SetAutoReconnect

Table 6-5 CLIENT_SetAutoReconnect

Item
Interface description
Pre-condition
Function
Parameter
Return value
Use examples

Item
Note

6.6 CLIENT_SetConnectTime

Table 6-6 CLIENT_SetConnectTime

Item	Description
Interface description	Sets device connection timeout value and trial times.
Pre-condition	Already called initialization interface CLIENT_Init
Function	void CLIENT_SetConnectTime(int nWaitTime, int nTryTimes);
Parameter	nWaitTime *[in]*The timeout time means waiting time for device's answer in every login. nTryTimes *[in]*The trial time means the times of trying to connect device in every login.
Return value	None

Use examples	// Set device connection timeout time and trial times.
	// This operation 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);

Note	If do not call CLIENT_SetConnectTime interface, the device response timeout is 5 seconds. The try to login device attempt is 1 by default.

6.7 CLIENT_SetNetworkParam

Table 6-7 CLIENT_SetNetworkParam

Item
Interface description
Pre-condition
Function
Parameter
Return value

Item
Use examples
Note

6.8 CLIENT_SetOptimizeMode

Table 6-8 CLIENT_SetOptimizeMode

Item	Description
Interface description	Optimize obtaining hard disk information
Pre-condition	CLIENT_Init initialization interface is already called.
Function	BOOL CLIENT_SetOptimizeMode(EM_OPTIMIZE_TYPE emType, void *pParam);
Parameter	● pstInParam [in]Input parameter, plan type emType. Refer to EM_OPTIMIZE_TYPE for the enumerated definition. [In]Input parameter, memory for pParam should be freed by the user. Refer to the size of the structure corresponding to emType.
Return value	If succeeded, return True. If failed, return False.
Use examples	int opt = OPTTYPE_MOBILE_DISK_INFO; CLIENT_SetOptimizeMode(EM_OPT_TYPE_MOBILE_OPTION, &opt);
Note	Not optimized by default.

6.9 CLIENT_LoginWithHighLevelSecurity

Table 6-9 CLIENT_LoginWithHighLevelSecurity

Item	Description
Interface description	High level login interface. To register user to device. It defines the device capabilities the user supported.
Pre-condition	Already called initialization interface CLIENT_Init
Function	LLONG CLIENT_LoginWithHighLevelSecurity (NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY* pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY* pstOutParam);
Parameter	pstInParam [in]Input parameter Refer to the structure definition of NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY pstOutParam [out]Output parameter

Item	Description
	Refer to the structure definition of NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY
Return value	Return the device ID for success, and return 0 for failure Uses this value (device ID) to operate the device after successfully logged in by working with interface of SDK.

Use examples	// Log in to the device
	NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
	memset(&stInparam, 0, sizeof(stInparam));
	stInparam.dwSize = sizeof(stInparam);
	strncpy(stInparam.szIP, "192.168.1.108", sizeof(stInparam.szIP) - 1);
	strncpy(stInparam.szPassword, "123456", sizeof(stInparam.szPassword) - 1);
	strncpy(stInparam.szUserName, "admin", sizeof(stInparam.szUserName) - 1);
	stInparam.nPort = 37777;
	stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
	
	NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
	memset(&stOutparam, 0, sizeof(stOutparam));
	stOutparam.dwSize = sizeof(stOutparam);
	LLONG lLoginID = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);

Note	Call this interface to register to the specified device after initialization. Return device ID for other functions to callback if successful. Recommended to login by TCP mode of emSpecCap = EM_LOGIN_SPEC_CAP_TCP

6.10 CLIENT_Logout

Table 6-10 CLIENT_Logout

Item	Description
Interface description	Logout interface.
Function	BOOL CLIENT_Logout(LLONG lLoginID);
Parameter	lLoginID [in] Device login handle Return value of CLIENT_LoginWithHighLevelSecurity
Return value	Return TRUE for success, and return FALSE for failure.

Use examples	printf("CLIENT_Logout!\n");
	if(!CLIENT_Logout(g_lLoginHandle))
	{
	printf("CLIENT_Logout Failed!Last Error[%x]\n" , CLIENT_GetLastError());
	}
	Refer to the synchronization login code of the device registration

Note	When logout device, the related businesses will stop ,such as real-time live

Item	Description
	view and so on.

6.11 CLIENT_RealPlayEx

Table 6-11 CLIENT_RealPlayEx

Item	Description
Interface description	Begin live view extension interface. It is to get real-time monitoring data stream from logged in device.
Pre-condition	Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	LLONG CLIENT_RealPlayEx(LLONG lLoginID, int nChannelID, HWND hWnd, DH_RealPlayType rType = DH_RType_Realplay);
Parameter	lLoginID *[In]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity nChannelID *[in]* Real-time monitoring channel number which starts from 0. hWnd *[in]* Window handle,when value is 0, data is not decoded and image is not displayed. rType *[in]* Real-time monitoring type. Default type is DH_RType_Realplay, Refer to enumeration definition of DH_RealPlayType
Return value	Return 0 when failed,otherwise return real-time monitoring ID(real-time monitoring handle) and used as parameters of related function.

Use examples	typedef HWND (WINAPI *PROCGETCONSOLEWINDOW)();
	PROCGETCONSOLEWINDOW GetConsoleWindow;
	// Gets the console window handle.
	HMODULE hKernel32 = GetModuleHandle("kernel32");
	GetConsoleWindow = (PROCGETCONSOLEWINDOW)GetProcAddress(hKernel32,"GetConsoleWindow");
	HWND hWnd = GetConsoleWindow();
	//Starts real-time monitoring.
	int nChannelID = 0; // Live view channel
	DH_RealPlayType emRealPlayType = DH_RType_Realplay;
	g_lRealHandle = CLIENT_RealPlayEx(g_lLoginHandle, nChannelID, hWnd, emRealPlayType);
	if (g_lRealHandle == 0)

Item	Description

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

Note	For NVR device, fills nChannelID as video output channel number in multi-play livew preview mode. According to information when device logged in, user can open a valid real-time monitoring channel and display it in any designated window by calling this interface. If succeeded, real-time monitoring ID is returned for more operation and control.

6.12 CLIENT_StopRealPlayEx

Table 6-12 CLIENT_StopRealPlayEx

Item	Description
Interface description	Stop real-time monitor extension interface, stop pulling real-time monitor bit stream from the logged in device.
Pre-condition	Already called CLIENT_RealPlayEx to pluu the real-time monitor bit stream
Function	BOOL CLIENT_StopRealPlayEx (LLONG lRealHandle);
Parameter	lRealHandle *[In]* Real-time monitor handle The return value of pulling real-time monitor bit stream interface such as CLIENT_RealPlayEx
Return value	Return TRUE for success, and return FALSE for failure

Use examples	if (!CLIENT_StopRealPlayEx(g_lRealHandle))
	{
	printf("CLIENT_StopRealPlayEx Failed, g_lRealHandle[%x]!Last Error[%x]\n" , g_lRealHandle, CLIENT_GetLastError());
	}

Note	None

6.13 CLIENT_SetRealDataCallBackEx

Table 6-13 CLIENT_SetRealDataCallBackEx

Item	Description
Interface description	Extension interface of setting the real-time monitoring data callback function.
Pre-condition	Already called initialization interface

Item	Description
	CLIENT_Init Already called CLIENT_LoginWithHighLevelSecurity to log in to the device. Already called CLIENT_RealPlayEx to pull the real-time monitor bit stream
Function	BOOL CLIENT_SetRealDataCallBackEx(LLONG lRealHandle, fRealDataCallBackEx cbRealData, LDWORD dwUser, DWORD dwFlag);
Parameter	lRealHandle *[In]* Real-time monitor handle The return value of pulling real-time monitor bit stream interface such as CLIENT_RealPlayEx cbRealData *[in]* Callback function of real-time monitoring data If cbRealData value is 0,do not callback real-time monitoring data. If cbRealData value is not 0,callback real-time monitoring data by callback function cbRealData. Refer to callback function (fRealDataCallBackEx) for details. dwUser *[in]* User data. SDK sends the data to user for further use by callback function fRealDataCallBackEx. dwFlag [in] Callback data selection flag Callback specified data only, do not callback data that has no callback data type. Different values have different data types. dwFlag — Data type 0x00000001 — Same as the original data 0x00000002 — MPEG4/H264 standard data 0x00000004 — YUV data 0x00000008 — PCM data 0x00000010 — Original audio data 0x0000001f — The above 5 data types
Return value	Return TRUE for success, and return FALSE for failure

Use examples	// It's not recommended to call SDK interface in callback function, unless call CLIENT_GetLastError to get error code of current thread.
	// Original shape of the real-time monitor call function-- Extension
	// When receiving the real-time monitoring data, SDK will call this function. Go to CLIENT_SetRealDataCallBackEx to set call function.
	// It is recommended to save data when using this callback function. It is to copy corresponding data to user's storage space and then encode/decode data after leaving callback function.
	// DO NOT encode/decode data directly on the callback function.
	void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD

Item	Description

	dwDataType, BYTE *pBuffer, DWORD dwBufSize, LONG param, LDWORD dwUser)
	{
	if (lRealHandle == g_lRealHandle)
	{
	switch(dwDataType)
	{
	case 0:
	//Original audio/video mixed data
	printf("receive real data, param: lRealHandle[%p], dwDataType[%d], pBuffer[%p], dwBufSize[%d], param[%p], dwUser[%p]\n",
	lRealHandle, 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;
	}
	}
	}
	**********Above are callback function definition, the underneath are interface examples**********
	DWORD dwFlag = 0x00000001;
	if (!CLIENT_SetRealDataCallBackEx(g_lRealHandle, &RealDataCallBackEx, NULL, dwFlag))
	{
	printf("CLIENT_SetRealDataCallBackEx: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

Note	Adds one callback data type flag dwFlag to callback specified data. Do not callback data that has no callback data type.

6.14 CLIENT_FindFile

Table 6-14 CLIENT_FindFile

Item	Description
Interface description	Open the record search handle
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	LLONG CLIENT_FindFile(LLONG lLoginID, int nChannelId, int nRecordFileType, char* cardid, LPNET_TIME time_start, LPNET_TIME time_end, BOOL bTime, int waittime);
Parameter	lLoginID *[In]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity nChannelId *[in]* Channel ID, starting from 0 nRecordFileType *[in]* Record file type The different record types have different values. Refer to the enumeration note of EM_QUERY_RECORD_TYPE. cardid *[in]* Extension parameter,working with nRecordFileType. nRecordFileType — cardid EM_RECORD_TYPE_CARD — Card No. EM_RECORD_TYPE_CONDITION — Card number &&transaction type&&transaction amount (Set as null if want to skip a specified field) EM_RECORD_TYPE_CARD_PICTURE — Card No. EM_RECORD_TYPE_FIELD — FELD1&&FELD2&&FELD3&&(Set as null if want to skip a specified field) The cardid value is NULL except the above conditions. tmStart *[in]* Start time of searching record Refer to the structure description of NET_TIME tmEnd *[in]* Stop time of searching record

Item	Description
	Refer to the structure description of NET_TIME bTime *[in]* Search by time or not This parameter is invalid now. Transmit FALSE. waittime *[in]* Waiting time
Return value	Return record search handle for success, and return 0 for failure

Use examples	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 search handle
	if(!CLIENT_FindFile (lLoginHandle, nChannelID,
	(int)EM_RECORD_TYPE_ALL, NULL, &StartTime, &StopTime, FALSE, 5000))
	{
	printf("CLIENT_FindFile: failed! Error code: %x.\n",
	CLIENT_GetLastError());
	}

Note	Call this interface to search video record before playback,then call CLIENT_FindNextFile function to return a detailed video record for playing. After search is finished,call CLIENT_FindClose close query handle.

6.15 CLIENT_FindNextFile

Table 6-15 CLIENT_FindNextFile

Item	Description
Interface description	Search
Pre-condition	Already called CLIENT_FindFile to get search record handle
Function	int CLIENT_FindNextFile(LLONG lFindHandle, LPNET_RECORDFILE_INFO lpFindData);
Parameter	lFindHandle *[in]* Record search handle

Item	Description
	Corresponding return value of device login interface of CLIENT_FindFile lpFindData *[out]* Record file butter To output searched record file information. Refer to NET_RECORDFILE_INFO
Return value	1: Successfully got one record, 0: Got all records, -1: Parameter error.

Use examples	NET_RECORDFILE_INFO struFileData = {0};
	int result = CLIENT_FindNextFile(lFindHandle, & struFileData);
	if(result == 1)//Get a video record file
	{
	// Storage record file
	}
	elseif(result == 0)//Got all record file info data
	{
	;
	}
	else//Parameter error
	{
	printf("CLIENT_FindNextFile: failed! Error code:0x%x.\n", CLIENT_GetLastError());
	}

Note	Before calling this interface, call CLIENT_FindFile first to open the search handle One call returns one video record.

6.16 CLIENT_FindClose

Table 6-16 CLIENT_FindClose

Item	Description
Interface description	Close the record search handle
Pre-condition	Already called CLIENT_FindFile to get search record handle
Function	BOOL CLIENT_FindClose(LLONG lFindHandle);
Parameter	lFindHandle *[in]* Record search handle Corresponding return value of CLIENT_FindFile
Return value	Return TRUE for success, and return FALSE for failure

Use examples	if(!CLIENT_FindClose (lFindHandle))
	{
	printf("CLIENT_FindNextFile: failed! Error code:0x%x.\n",

Item	Description
	CLIENT_GetLastError()); }
Note	Call CLIENT_FindFile to open the search handle; after the search is completed, call this function to close the search handle

6.17 CLIENT_PlayBackByTimeEx

Table 6-17 CLIENT_PlayBackByTimeEx

Item	Description
Interface description	To playback by time.-- Extension interface
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	LLONG CLIENT_PlayBackByTimeEx(LLONG lLoginID, int nChannelID, LPNET_TIME lpStartTime, LPNET_TIME lpStopTime, HWND hWnd, fDownLoadPosCallBack cbDownLoadPos, LDWORD dwPosUser, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity nChannelID *[in]* Channel ID, starting from 0 lpStartTime *[in]* Playback start time Refer to the structure description of NET_TIME lpStopTime *[in]* Playback end time Refer to the structure description of NET_TIME hWnd *[in]* Playback window cbDownLoadPos *[In]* Progress callback user parameters If cbDownLoadPos value is 0,do not callback playback data process; If cbDownLoadPos value is not 0,callback playback data process by cbDownLoadPos to user. Refer to callback function note of fDownLoadPosCallBack

Item	Description
	dwPosUser [in] User data SDK returns the data to user by playback data process callback function fDownLoadPosCallBack so that the user can continue the following operations fDownLoadDataCallBack [in] Record data callback function If fDownLoadDataCallBack value is 0, do not callback playback data process; If fDownLoadDataCallBack value is not 0, callback playback data process by cbDownLoadPos to user. Refer to callback function note of fDataCallBack dwDataUser [in] User data SDK returns the data to user by playback data process callback function fDownLoadPosCallBack so that the user can continue the following operations.
Return value	Return record playback handle for success, and return 0 for failure

Use examples	// The following sample codes are based on playsdk library decode when playback by time.
	typedef HWND (WINAPI *PROCGETCONSOLEWINDOW)();
	PROCGETCONSOLEWINDOW GetConsoleWindow;
	// Get the console window handle
	HMODULE hKernel32 = GetModuleHandle("kernel32");
	GetConsoleWindow = (PROCGETCONSOLEWINDOW)GetProcAddress(hKernel32,"GetConsoleWindow");
	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_lPlayHandle = CLIENT_PlayBackByTimeEx(g_lLoginHandle, nChannelID, &stuStartTime, &stuStopTime, hWnd, NULL, NULL, NULL, NULL);
	if (g_lPlayHandle == 0)
	{
	printf("CLIENT_PlayBackByTimeEx: failed! Error code: 0x%x.\n", CLIENT_GetLastError());

Item	Description
	}
Note	hWnd and fDownLoadDataCallBack can not be NULL at the same time,otherwise the interface callback may fail.

6.18 CLIENT_StopPlayBack

Table 6-18 CLIENT_StopPlayBack

Item	Description
Interface description	Stop record playback interface
Pre-condition	Already called interfaces such as CLIENT_PlayBackByTimeEx to get record playback handle
Function	BOOL CLIENT_StopPlayBack(LLONG lPlayHandle);
Parameter	lPlayHandle *[in]* Record Playback handle Corresponding return value of CLIENT_PlayBackByTimeEx
Return value	Return TRUE for success, and return FALSE for failure

Use examples	if (!CLIENT_StopPlayBack(g_lPlayHandle))
	{
	printf("CLIENT_StopPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

Note	Call interface such as CLIENT_PlayBackByTimeEx to get record playback handle,Call CLIENT_StopPlayBack to close record playback handle.

6.19 CLIENT_GetPlayBackOsdTime

Table 6-19 CLIENT_GetPlayBackOsdTime

Item	Description
Interface description	Get playback OSD time interface The parameters of this interface is valid only when parameter hWnd of opening file playback interface is valid. Otherwise it is invalid.
Pre-condition	Already called interfaces such as CLIENT_PlayBackByTimeEx to get record playback handle
Function	BOOL CLIENT_GetPlayBackOsdTime(LLONG lPlayHandle, LPNET_TIME lpOsdTime, LPNET_TIME lpStartTime, LPNET_TIME lpEndTime

Item	Description
	);
Parameter	lPlayHandle *[in]* Record playback handle Corresponding return value of CLIENT_PlayBackByTimeEx lpOsdTime *[out]* OSD time Refer to the structure note of NET_TIME lpStartTime *[in]* Playback start time Refer to the structure note of NET_TIME lpEndTime *[in]* Playback end time Refer to the structure note of NET_TIME
Return value	Return TRUE for success, and return FALSE for failure

Use examples	NET_TIME stuOsdTime = {0};
	NET_TIME stuStartTime = {0};
	NET_TIME stuEndTime = {0};
	
	if (!CLIENT_GetPlayBackOsdTime (g_lPlayHandle, &stuOsdTime, &stuStartTime, &stuEndTime))
	{
	printf("CLIENT_ GetPlayBackOsdTime Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

Note	The parameters of this interface is valid only when parameter hWnd of opening file playback interface is valid. Otherwise it is invalid.

6.20 CLIENT_QueryRecordFile

Table 6-20 CLIENT_QueryRecordFile

Item	Description
Interface description	Search the interfaces of all record files in this period.
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_QueryRecordFile(LLONG lLoginID, int nChannelId, int nRecordFileType,

<table>
<tr><th>Item</th><th>Description</th></tr>
<tr><td></td><td>LPNET_TIME tmStart,
LPNET_TIME tmEnd,
char* pchCardid,
LPNET_RECORDFILE_INFO nriFileinfo,
int maxlen,
int *filecount,
int waittime=1000,
BOOL bTime = FALSE
);</td></tr>
<tr><td>Parameter</td><td>lLoginID
[in] Device login ID
Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity
nChannelId
[in] Channel ID, starting from 0
nRecordFileType
[in] Record file type
The different record types have different values. Refer to the enumeration note of EM_QUERY_RECORD_TYPE .
tmStart
[in] Start time of searching record
Refer to the structure description of NET_TIME
tmEnd
[in] End time of searching record
Refer to the structure description of NET_TIME
pchCardid
[in] Extension parameter,working with nRecordFileType.
<table>
<tr><td>nRecordFileType</td><td>pchCardid</td></tr>
<tr><td>EM_RECORD_TYPE_CARD</td><td>Card No.</td></tr>
<tr><td>EM_RECORD_TYPE_CONDITION</td><td>Card number &&transaction type&&transaction amount (Set as null if want to skip a specifed field)</td></tr>
<tr><td>EM_RECORD_TYPE_CARD_PICTURE</td><td>Card No.</td></tr>
<tr><td>EM_RECORD_TYPE_FIELD</td><td>FELD1&&FELD2&&FELD3&&(Set as null if want to skip a specified field)</td></tr>
</table></td></tr>
</table>

Item	Description
	pchCardid value is NULL except the above conditions. nriFileinfo *[out]* Info of the returned record file The pointer of the structure array NET_RECORDFILE_INFO. Refer to structure note of NET_RECORDFILE_INFO maxlen *[in]* nriFileinfo butter max. length Unit:byte. Recommended length:(100~200)*sizeof(NET_RECORDFILE_INFO) filecount *[out]* Returned file amount Get the max. output parameter when buffer is full. waittime *[In]* Waiting time bTime *[in]* Search by time or not This parameter is invalid now. Transmit FALSE.
Return value	Return TRUE for success, and return FALSE for failure

Use examples	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;
	//Search record file
	if(!CLIENT_QueryRecordFile(lLoginHandle, nChannelID, (int)EM_RECORD_TYPE_ALL, &StartTime, &StopTime, NULL, &netFileInfo[0], sizeof(netFileInfo), &nFileCount,5000, FALSE))
	{
	printf("CLIENT_QueryRecordFile: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

Note	Before playback by file,call this interface to

Item	Description
	search video record.If searched video record info of input period is larger than defined buffer size, SDK returns video record that buffer can storage,and can continue search as needed.

6.21 CLIENT_DownloadByTimeEx

Table 6-21 CLIENT_DownloadByTimeEx

Item	Description
Interface description	Extension interface of download the recorded video by time.
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	LLONG CLIENT_DownloadByTimeEx(LLONG lLoginID, int nChannelId, int nRecordFileType, LPNET_TIME tmStart, LPNET_TIME tmEnd, char *sSavedFileName, fTimeDownLoadPosCallBack cbTimeDownLoadPos, LDWORD dwUserData, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser, void* pReserved = NULL);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity nChannelId *[in]* Channel number, starting from 0 nRecordFileType *[in]* Record file type Refer to enumeration note of EM_QUERY_RECORD_TYPE tmStart *[in]* Start time of downloading record Refer to the structure note of NET_TIME tmEnd *[in]* End time of downloading record

Item	Description
	Refer to the structure note of NET_TIME sSavedFileName *[In]* Video file name user wants to save Full path is recommended cbTimeDownLoadPos [in]Download process callback function Refer to callback function note of fTimeDownLoadPosCallBack dwUserData *[In]* User data of download progress callback functions SDK returns the data to user by download progress function fTimeDownLoadPosCallBack so that the user can continue the following operations fDownLoadDataCallBack [in] Download data callback function Refer to callback function note of fDataCallBack dwDataUser *[in]* User data of download callback functions SDK returns the data to user by playback data process callback function fDataCallBack so that the user can continue the following operations pReserved [In] Reserved parameter For future development. It is invalid now. Default value is NULL.
Return value	Return the download ID for success, and return 0 for failure

Use examples	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 download records
	// At least one value of formal parameter sSavedFileName or fDownLoadDataCallBack is valid.
	g_lDownloadHandle =

Item	Description

	CLIENT_DownloadByTimeEx(g_lLoginHandle, nChannelID, EM_RECORD_TYPE_ALL, &stuStartTime, &stuStopTime, "test.dav", TimeDownLoadPosCallBack, NULL, DataCallBack, NULL);
	if (g_lDownloadHandle == 0)
	{
	printf("CLIENT_DownloadByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

Note	sSavedFileName is not null, write the record data to the file of the corresponding path; fDownLoadDataCallBack is not null, return record data by callback function After download is complete, call CLIENT_StopDownload to close download handle.

6.22 CLIENT_StopDownload

Table 6-22 CLIENT_StopDownload

Item	Description
Interface description	Stop downloading record interface
Pre-condition	Already called record download interface such as CLIENT_DownloadByTimeEx
Function	BOOL CLIENT_StopDownload(LLONG lFileHandle);
Parameter	lFileHandle [in] Download handle Corresponding return value of record download interface such as CLIENT_DownloadByTimeEx
Return value	Return TRUE for success, and return FALSE for failure

Use examples	// Close download. Call after download is complete or call during the download.
	if (g_lDownloadHandle)
	{
	if (!CLIENT_StopDownload(g_lDownloadHandle))
	{
	printf("CLIENT_StopDownload Failed, g_lDownloadHandle[%x]!Last Error[%x]\n" , g_lDownloadHandle, CLIENT_GetLastError());

Item	Description
	} }
Note	Close download when all files are downloaded or stop download during the downloading process.

6.23 CLIENT_PlayBackByRecordFileEx

Table 6-23 CLIENT_PlayBackByRecordFileEx

Item	Description
Interface description	Playback by file extension interface
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	LLONG CLIENT_PlayBackByRecordFileEx(LLONG lLoginID, LPNET_RECORDFILE_INFO lpRecordFile, HWND hWnd, fDownLoadPosCallBack cbDownLoadPos, LDWORD dwPosUser, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity lpRecordFile *[In]* Record file information Get by record information search interface such as CLIENT_FindNextFile. Refer to structure note of NET_RECORDFILE_INFO Refer to the structure note of NET_TIME hWnd *[In]* Playback window cbDownLoadPos *[in]* Record process callback function If cbDownLoadPos value is 0,do not callback playback data process; If cbDownLoadPos value is not 0,callback playback data process by cbDownLoadPos to user. Refer to callback function note of

Item	Description
	fDownLoadPosCallBack dwPosUser [in] User data SDK returns the data to user by playback data process callback function fDownLoadPosCallBack so that the user can continue the following operations fDownLoadDataCallBack [in] Record data callback function If fDownLoadDataCallBack value is 0, do not callback playback data process; If fDownLoadDataCallBack value is not 0, callback playback data process by cbDownLoadPos to user. Refer to callback function note of fDataCallBack dwDataUser [in] User data SDK returns the data to user by playback data process callback function fDownLoadPosCallBack so that the user can continue the following operations
Return value	Return record playback handle for success, and return 0 for failure

Use examples	// Function formal parameter pa hWnd need to be valid.
	// stuNetFileInfo is the record file info of three interfaces: CLIENT_FindFile,CLIENT_FindNextFile,CLIENT_FindClose
	g_lPlayHandle = CLIENT_PlayBackByRecordFileEx(g_lLoginHandle, &stuNetFileInfo, hWnd, NULL, NULL, NULL, NULL);
	if (g_lPlayHandle == 0)
	{
	printf("CLIENT_PlayBackByRecordFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
	}

Note	The hWnd and fDownLoadDataCallBack can not be NULL at the same time,otherwise the function callback may fail.

6.24 CLIENT_PausePlayBack

Table 6-24 CLIENT_PausePlayBack

Item	Description
Interface description	Pause or resume record playback The parameters of this interface is valid only when parameter hWnd of opening file playback interface is valid. Otherwise it is invalid.
Pre-condition	Already called interfaces such as CLIENT_PlayBackByTimeEx to get record playback handle
Function	BOOL CLIENT_PausePlayBack(LLONG lPlayHandle, BOOL bPause);
Parameter	lPlayHandle *[in]* Record playback handle Corresponding return value of CLIENT_PlayBackByTimeEx bPause *[in]* The tag of the playback pause and resume playback control TRUE: Pause, FALSE: Resume
Return value	Return TRUE for success, and return FALSE for failure

Use examples	if (!CLIENT_ PausePlayBack (g_lPlayHandle))
	{
	printf("CLIENT_ PausePlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

Note	The parameters of this interface is valid only when parameter hWnd of opening file playback interface is valid. Otherwise it is invalid.

6.25 CLIENT_SeekPlayBack

Table 6-25 CLIENT_SeekPlayBack

Item	Description
Interface description	Locate the start position of record playback
Pre-condition	Already called interfaces such as CLIENT_PlayBackByTimeEx to get record playback handle
Function	BOOL CLIENT_SeekPlayBack(LLONG lPlayHandle, unsigned int offsettime, unsigned int offsetbyte);
Parameter	lPlayHandle *[in]* Record playback handle Corresponding return value of CLIENT_PlayBackByTimeEx offsettime *[in]* Relative offset of start time(unit : s)

Item	Description
	offsetbyte [in] This parameter is deleted Set value as 0xffffffff.
Return value	Return TRUE for success, and return FALSE for failure

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

Note	None

6.26 CLIENT_FastPlayBack

Table 6-26 CLIENT_FastPlayBack

Item	Description
Interface description	Fast play interface.Increasing frame rate by 1x The parameters of this interface is valid only when parameter hWnd of opening file playback interface is valid. Otherwise it is invalid.
Pre-condition	Already called interfaces such as CLIENT_PlayBackByTimeEx to get record playback handle
Function	BOOL CLIENT_FastPlayBack(LLONG lPlayHandle);
Parameter	lPlayHandle *[in]* Record playback handle Corresponding return value of CLIENT_PlayBackByTimeEx
Return value	Return TRUE for success, and return FALSE for failure

Use examples	if (!CLIENT_ FastPlayBack (g_lPlayHandle))
	{
	printf("CLIENT_ FastPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

Note	Can not fast forward without limit, currently the max frame is 200. Return FALSE if the value is bigger than 200 frames.Fast forward is null if there is audio. The parameters of this interface is valid only when parameter hWnd of opening file playback interface is valid. Otherwise it is invalid.

6.27 CLIENT_SlowPlayBack

Table 6-27 CLIENT_SlowPlayBack

Item	Description
Interface description	Slow play interface.Decreasing frame rate by 1/2
Pre-condition	Already called interfaces such as CLIENT_PlayBackByTimeEx to get record playback handle
Function	BOOL CLIENT_SlowPlayBack (LLONG lPlayHandle);
Parameter	lPlayHandle *[in]* Record playback handle Corresponding return value of CLIENT_PlayBackByTimeEx
Return value	Return TRUE for success, and return FALSE for failure

Use examples	if (!CLIENT_SlowPlayBack (g_lPlayHandle))
	{
	printf("CLIENT_SlowPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

Note	The min frame is 1. Return FALSE if the value is less than 1. When the parameter hWnd of opening record playback interface is 0 and device supports playback speed control, SDK can send speed control command to device. When the parameter hWnd of opening record playback interface is a valid value and device supports playback speed control, SDK can send speed control command to device and call the speed control command of playsdk library displayed on the window.

6.28 CLIENT_NormalPlayBack

Table 6-28 CLIENT_NormalPlayBack

Item	Description
Interface description	Resume normal playback speed interface
Pre-condition	Already called interfaces such as CLIENT_PlayBackByTimeEx to get record playback handle
Function	BOOL CLIENT_NormalPlayBack(

Item	Description
	LLONG lPlayHandle);
Parameter	lPlayHandle *[in]* Record playback handle Corresponding return value of CLIENT_PlayBackByTimeEx
Return value	Return TRUE for success, and return FALSE for failure

Use examples	if (!CLIENT_NormalPlayBack (g_lPlayHandle))
	{
	printf("CLIENT_NormalPlayBack Failed, g_lPlayHandle[%x]!Last Error[%x]\n" , g_lPlayHandle, CLIENT_GetLastError());
	}

Note	When the parameter hWnd of opening record playback interface is 0 and device supports playback speed control, SDK can send speed control command to device. When the parameter hWnd of opening record playback interface is a valid value and device supports playback speed control, SDK can send speed control command to device and call the speed control command of playsdk library displayed on the window.

6.29 CLIENT_DownloadByRecordFileEx

Table 6-29 CLIENT_DownloadByRecordFileEx

Item	Description
Interface description	Download by time extension interface
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	LLONG CLIENT_DownloadByRecordFileEx(LLONG lLoginID, LPNET_RECORDFILE_INFO lpRecordFile, char *sSavedFileName, fDownLoadPosCallBack cbDownLoadPos, LDWORD dwUserData, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser, void* pReserved = NULL);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity lpRecordFile *[in]* Record file information pointer

Item	Description
	Obtained by record search interface. Refer to NET_RECORDFILE_INFO sSavedFileName *[In]* Video file name user wants to save Full path is recommended cbDownLoadPos [in] Download process callback function Refer to callback function fDownLoadPosCallBack dwUserData [in] User data of download process callback function SDK returns the data to user by download progress function fTimeDownLoadPosCallBack so that the user can continue the following operations fDownLoadDataCallBack [in] Download process callback function Refer to callback function fDataCallBack dwUserData [in] User data of download callback function SDK returns the data to user by playback data process callback function fDataCallBack so that the user can continue the following operations pReserved [in] Reserved parameter For future development. It is invalid now. Default value is NULL.
Return value	Return the download ID for success, and return 0 for failure

Use examples	// At least one value of formal parameter sSavedFileName or fDownLoadDataCallBack is valid.
	g_lDownloadHandle = CLIENT_DownloadByRecordFileEx(g_lLoginHandle, &stuNetFileInfo, "test.dav", DownLoadPosCallBack, NULL, DataCallBack, NULL);
	if (g_lDownloadHandle == 0)
	{
	printf("CLIENT_DownloadByRecordFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
	
	}

Note	sSavedFileName is not null, write the record

Item	Description
	data to the file of the corresponding path; fDownLoadDataCallBack is not null, return record data by callback function. After download is complete, call CLIENT_StopDownload to close download handle..

6.30 CLIENT_ParseData

Table 6-30 CLIENT_ParseData

Item	Description
Interface description	Parse the searched configuration information
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_ParseData(char* szCommand, char* szInBuffer, LPVOID lpOutBuffer, DWORD dwOutBufferSize, void* pReserved);
Parameter	szCommand [in] Command parameter Refer to the following notes for details. szInBuffer [in] Input buffer Input the json string contents for the buffer internal storage to parse lpOutBuffer [out] Output buffer Different commands are corresponding to different structure types. Refer to the following notes for detail. dwOutBufferSize [in] Output buffer size pReserved [in] Reserved parameter
Return value	Return TRUE for success, and return FALSE for failure
Use examples	CFG_PTZ_PROTOCOL_CAPS_INFO stuPtzCapsInfo = {sizeof(stuPtzCapsInfo)}; if (FALSE == CLIENT_ParseData(CFG_CAP_CMD_PTZ, pBuffer,

Item	Description

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

Note	Command Parameters: #define CFG_CAP_CMD_PTZ "ptz.getCurrentProtocolCaps" // Get PTZ capability set(CFG_PTZ_PROTOCOL_CAPS_INFO) #define CFG_CMD_ENCODE "Encode" // Video channel properties setup (CFG_ENCODE_INFO) Refer to dhconfigsdk.h for more command parameters

6.31 CLIENT_DHPTZControlEx2

Table 6-31 CLIENT_DHPTZControlEx2

Item	Description
Interface description	Private PTZ control extension port Support 3D fast positioning, fisheye
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_DHPTZControlEx2(LLONG lLoginID, int nChannelID, DWORD dwPTZCommand, LONG lParam1, LONG lParam2, LONG lParam3, BOOL dwStop , void* param4 = NULL);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity nChannelID *[in]* Operation channel No.

Item	Description
	Channel number starting from 0 dwPTZCommand *[in]* Speed dome control commands Refer to enumeration note of DH_PTZ_ControlType and DH_EXTPTZ_ControlType lParam1 *[in]* Aux parameter 1 Working with other parameters. Different control commands have different parameter combination:groups. lParam2 *[in]* Aux parameter 2 Working with other parameters. Different control commands have different parameter combination:groups. lParam3 *[in]* Aux parameter 3 Working with other parameters. Different control commands have different parameter combination:groups. dwStop [in] Stop or not It is valid when operating PTZ eight directions and lens, otherwise fill in FALSE when operating others functions. lParam4 *[in]* Aux parameter 4. Default value is NULL. Working with other parameters. Different control commands have different parameter combination:groups.
Return value	Return TRUE for success, and return FALSE for failure

Use examples	if (!CLIENT_DHPTZControlEx2(g_lLoginHandle, 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());
	}

Note	Refer to CLIENT_DHPTZControlEx2 on Network SDK development manual for lParam1-4 information.

6.32 CLIENT_QueryNewSystemInfo

Table 6-32 CLIENT_QueryNewSystemInfo

Item	Description
Interface description	New system capability search interface. Search system capability information(Json format. Refer to configuration SDK)
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_QueryNewSystemInfo(LLONG lLoginID, char* szCommand, int nChannelID, char* szOutBuffer, DWORD dwOutBufferSize, int *error, int waittime=1000);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity szCommand *[in]* Corresponding search command Refer to notes. nChannelID *[in]* Corresponding search channel. Channel begins with 0. When it is -1, search all channels. Some commands do not support channel number as -1. szOutBuffer [in]Storage data buffer To save the searched json data dwOutBufferSize *[in]* Buffer size error [out] Return error code Netsdk fills in the corresponding error code on the pointer address if failed to get. waittime [in]Timeout period Wait for the returned command timeout . 1000ms by defaults

Item	Description
Return value	Return TRUE for success, and return FALSE for failure

Use examples	char* pBuffer = new char[2048];
	if (NULL == pBuffer)
	{
	return;
	}
	
	int nError = 0;
	if (FALSE == CLIENT_QueryNewSystemInfo(g_lLoginHandle, CFG_CAP_CMD_PTZ, 0, pBuffer, 2048, &nError))
	{
	printf("CLIENT_QueryNewSystemInfo Failed, cmd[CFG_CAP_CMD_PTZ], Last Error[%x]\n" , CLIENT_GetLastError());
	if (pBuffer)
	{
	delete [] pBuffer;
	pBuffer = NULL;
	}
	return;
	}

Note	Uses CLIENT_ParseData to analyze json got by this interface, otherwise, it can not be used. The capability set command of CLIENT_QueryNewSystemInfo is: #define CFG_CAP_CMD_PTZ "ptz.getCurrentProtocolCaps" // Get PTZ capability set (CFG_PTZ_PROTOCOL_CAPS_INFO) Refer to dhconfigsdk.h for more commands.

6.33 CLIENT_SetDeviceMode

Table 6-33 CLIENT_SetDeviceMode

Item	Description
Interface description	Set working mode interface of device audio talk, playback and rights
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_SetDeviceMode(LLONG lLoginID,

Item	Description
	EM_USEDEV_MODE emType, void* pValue);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity emType *[in]* Work mode type Refer to enumeration note of EM_USEDEV_MODE pValue [in] Extension parameter The different emType values have different extension parameters. Refer to the enumeration note of EM_QUERY_RECORD_TYPE.
Return value	Return TRUE for success, and return FALSE for failure

Use examples	// Set bit stream type when playback
	int nStreamType = 0; // 0-main and sub stream, 1-main stream, 2-sub stream
	if(!CLIENT_SetDeviceMode(g_lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType))
	{
	printf("CLIENT_ SetDeviceMode: failed! Error code: 0x%x.\n", CLIENT_GetLastError());
	}

Note	None

6.34 CLIENT_StartSearchDevicesEx

Table 6-34 CLIENT_StartSearchDevicesEx

Item	Description
Interface description	Asynchronously search the IPC, NVS device in the same IP segment
Pre-condition	Already called initialization interface CLIENT_Init
Function	LLONG CLIENT_StartSearchDevicesEx (NET_IN_STARTSERACH_DEVICE* pInBuf, NET_OUT_STARTSERACH_DEVICE* pOutBuf);
Parameter	pInBuf

Item	Description
	[in] Input parameter for searching device asynchronously. Refer to the definition of NET_IN_STARTSERACH_DEVICE pOutBuf *[out]* Output parameter for searching device asynchronously. Refer to the definition of NET_OUT_STARTSERACH_DEVICE
Return value	Return handle for success, and return 0 for failure

Use examples	// Start Asynchronously search the IPC, NVS device in the same IP segment
	NET_IN_STARTSERACH_DEVICE stuInParam = {sizeof(stuInParam)};
	stuInParam.emSendType = EM_SEND_SEARCH_TYPE_BROADCAST;
	stuInParam.cbSearchDevices = SearchDevicesCBEx;
	NET_OUT_STARTSERACH_DEVICE stuOutParam = {sizeof(stuOutParam)};
	LLONG g_lSearchHandle = CLIENT_StartSearchDevicesEx(SearchDevicesCB, &g_lDeviceList);
	if (NULL == g_lSearchHandle)
	{
	printf("CLIENT_StartSearchDevicesEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	return;
	}

Note	The interface searches the device in the same IP segment. Call CLIENT_SearchDevicesByIPs to search in different IP segments at the same time.

6.35 CLIENT_QueryDevState

Table 6-35 CLIENT_QueryDevState

Item	Description
Interface description	Search device status
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_QueryDevState(LLONG lLoginID, int nType, char *pBuf,

Item	Description
	int nBufLen, int *pRetLen, int waittime=1000);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity. nType [In] Search type Refer to the following notes for details. pBuf [out] Output buffer To save the searched result information, working with search matching type. Refer to the following notes. nBufLen [in] Buffer zone size pRetLen *[out]* Actually searched data length. The unit is byte waittime *[In]* Search waiting time,1000ms by default
Return value	Return TRUE for success, and return FALSE for failure

Use examples	// To get the encode type of audio talk supported by the 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 (!(bSuccess && retlen == sizeof(stulstTalkEncode)))
	{
	printf("CLIENT_QueryDevState cmd[%d] Failed!Last Error[%x]\n" , DH_DEVSTATE_TALK_ECTYPE, CLIENT_GetLastError());
	}

Note	Supported search types #define DH_DEVSTATE_TALK_ECTYPE

Item	Description
	0x0009 // Search the audio talk format list device supported. Refer to DHDEV_TALKFORMAT_LIST Refer to dhnetsdk.h for more commands.

6.36 CLIENT_StartTalkEx

Table 6-36 CLIENT_StartTalkEx

Item	Description
Interface description	Extension interface of starting the audio talk
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	LLONG CLIENT_StartTalkEx(LLONG lLoginID, pfAudioDataCallBack pfcb, LDWORD dwUser);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity pfcb *[In]* Audio data callback function of audio talk Refer to callback function of pfAudioDataCallBack dwUser *In]* User data of audio data callback function of audio talk SDK returns the data to user by download progress function pfAudioDataCallBack so that the user can continue the following operations
Return value	Return handle of audio talk for success, and return 0 for failure

Use examples	g_lTalkHandle =
	CLIENT_StartTalkEx(g_lLoginHandle, AudioDataCallBack, (DWORD)NULL);
	if(0 == g_lTalkHandle)
	{
	printf("CLIENT_StartTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	}

Note	None

6.37 CLIENT_StopTalkEx

Table 6-37 CLIENT_StopTalkEx

Item	Description
Interface description	Stop audio talk extension interface
Pre-condition	Already called audio talk interface such as CLIENT_StartTalkEx
Function	BOOL CLIENT_StopTalkEx(LLONG lTalkHandle);
Parameter	lTalkHandle *[in]* Handle ID of audio talk Corresponding return value of opening audio talk interface such as CLIENT_StartTalkEx
Return value	Return TRUE for success, and return FALSE for failure

Use examples	if(!CLIENT_StopTalkEx(g_lTalkHandle))
	{
	printf("CLIENT_StopTalkEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	}
	else
	{
	g_lTalkHandle = 0;
	}

Note	None

6.38 CLIENT_RecordStartEx

Table 6-38 CLIENT_RecordStartEx

Item	Description
Interface description	Start audio extension interface on PC (Extension of CLIENT_RecordStart())
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_RecordStartEx(LLONG lLoginID);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity
Return value	Return TRUE for success, and return FALSE for failure
Use examples	BOOL bSuccess =

Item	Description

	CLIENT_RecordStartEx(g_lLoginHandle);
	if(!bSuccess)
	{
	printf("CLIENT_RecordStartEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	}

Note	None

6.39 CLIENT_RecordStopEx

Table 6-39 CLIENT_RecordStopEx

Item	Description
Interface description	Stop audio extension interface on PC (Extension of CLIENT_RecordStart())
Pre-condition	Already called CLIENT_RecordStartEx to enable local audio collection interface
Function	BOOL CLIENT_RecordStopEx(LLONG lLoginID);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity
Return value	Return TRUE for success, and return FALSE for failure

Use examples	//Stop local audio record
	if (g_RecordFlag)
	{
	if (!CLIENT_RecordStopEx(g_lLoginHandle))
	{
	printf("CLIENT_RecordStop Failed!Last Error[%x]\n", CLIENT_GetLastError());
	}
	else
	{
	g_RecordFlag = FALSE;
	}
	}

Note	CLIENT_RecordStopEx needs to work with CLIENT_RecordStartEx

6.40 CLIENT_TalkSendData

Table 6-40 CLIENT_TalkSendData

Item	Description
Interface description	Send audio data to the device
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	LONG CLIENT_TalkSendData(LLONG lTalkHandle, char *pSendBuf, DWORD dwBufSize);
Parameter	lTalkHandle *[in]* Audio talk handle ID Corresponding return value of opening audio talk such as CLIENT_StartTalkEx pSendBuf *[In]* Send buffer zone Save audio data to be sent dwBufSize [in] Buffer size, Length of audio data to be sent. Unit is byte
Return value	Return the transmits locations device length of data for success, and return -1 for failure

Use examples	LONG lSendLen =
	CLIENT_TalkSendData(lTalkHandle, pDataBuf, dwBufSize);
	if(lSendLen != (long)dwBufSize)
	{
	printf("CLIENT_TalkSendData Failed!Last Error[%x]\n" , CLIENT_GetLastError());
	}

Note	After receiving the audio data from CLIENT_StartTalkEx, use this interface to send to device.

6.41 CLIENT_AudioDecEx

Table 6-41 CLIENT_AudioDecEx

Item	Description
Interface description	Decode audio data extension interface
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.

Item	Description
Function	BOOL CLIENT_AudioDecEx(LLONG lTalkHandle, char *pAudioDataBuf, DWORD dwBufSize);
Parameter	lTalkHandle *[in]* Audio talk handle ID Corresponding return value of opening audio talk interface such as CLIENT_PlayBackByTimeEx pAudioDataBuf *[In]* Audio buffer zone Audio data to be decoded dwBufSize [in] Buffer size Length of audio data to be decoded. Unit is byte
Return value	Return TRUE for success, and return FALSE for failure

Use examples	//Pass the audio data sent from the device to SDK for decoding play
	if (!CLIENT_AudioDecEx(lTalkHandle, pDataBuf, dwBufSize))
	{
	printf("CLIENT_AudioDecEx Failed!Last Error[%x]\n" ,
	CLIENT_GetLastError());
	}

Note	Decode the data from the audio talk device

6.42 CLIENT_SetDVRMessCallBack

Table 6-42 CLIENT_SetDVRMessCallBack

Item	Description
Interface description	Set alarm callback function interface
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	void CLIENT_SetDVRMessCallBack(fMessCallBack cbMessage, LDWORD dwUser);
Parameter	cbMessage [in] Alarm callback function Refer to callback function fMessCallBack dwUser *[in]* User data. SDK sends the data to user for further use by callback function fMessCallBack
Return value	None
Use examples	/ / Set alarm event callback function

Item	Description
	CLIENT_SetDVRMessCallBack(MessCallBack , NULL);
Note	Call CLIENT_SetDVRMessCallBack before alarm subscription. The event of the configured callback function do not contain the event picture.

6.43 CLIENT_StartListenEx

Table 6-43 CLIENT_StartListenEx

Item	Description
Interface description	Alarm subscription extension interface
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_StartListenEx(LLONG lLoginID);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity
Return value	Return TRUE for success, and return FALSE for failure

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

Note	Alarm events of all devices returned to the user are by callback function of CLIENT_SetDVRMessCallBack

6.44 CLIENT_StopListen

Table 6-44 CLIENT_StopListen

Item	Description
Interface description	Stop subscribing alarm

Item	Description
Pre-condition	Already called alarm reporting interface such as CLIENT_StartListenEx
Function	BOOL CLIENT_StopListen(LLONG lLoginID);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity
Return value	Return TRUE for success, and return FALSE for failure

Use examples	/ / Stop subscribing alarm from the device
	if (g_bStartListenFlag)
	{
	if (!CLIENT_StopListen(g_lLoginHandle))
	{
	printf("CLIENT_StopListen Failed!Last Error[%x]\n", CLIENT_GetLastError());
	}
	else
	{
	g_bStartListenFlag = FALSE;
	}
	}

Note	None

6.45 CLIENT_StopSearchDevices

Table 6-45 CLIENT_StopSearchDevices

Item	Description
Interface description	Stop asynchronously search the IPC, NVS device in the same IP segment
Pre-condition	Already called asynchronously search device interface such as CLIENT_StartSearchDevicesEx
Function	BOOL CLIENT_StopSearchDevices(LLONG lSearchHandle);
Parameter	lSearchHandle [in] Asynchronously search device ID Corresponding return value of asynchronously search device interface such as CLIENT_StartSearchDevicesEx
Return value	Return TRUE for success, and return FALSE for failure

Use examples	// Stop asynchronously search device in the same IP segment
	if (NULL != g_lSearchHandle)
	{
	if (FALSE == CLIENT_StopSearchDevices(g_lSearchHandle))

Item	Description

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

Note	The interface needs to work with CLIENT_StartSearchDevicesEx

6.46 CLIENT_SearchDevicesByIPs

Table 6-46 CLIENT_SearchDevicesByIPs

Item	Description
Interface description	Synchronously search device cross different IP segments at the same time
Pre-condition	Already called initialization interface CLIENT_Init
Function	BOOL CLIENT_SearchDevicesByIPs(DEVICE_IP_SEARCH_INFO* pIpSearchInfo, fSearchDevicesCB cbSearchDevices, LDWORD dwUserData, char* szLocalIp, DWORD dwWaitTime);
Parameter	pIpSearchInfo *[in]* Search device information Save the device IP to be searched. DEVICE_IP_SEARCH_INFO refer to dhnetsdk.h cbSearchDevices *[In]* Callback function for searching device When there is device response packet, SDK parses the response packet to valid information and then notify the user by callback function. Refer to callback function note of fSearchDevicesCB for details. Callback address cannot be null. dwUserData [in] User data NetSDK searches device callback function fSearchDevicesCB to return the data to user so that the user can continue the following operations. szLocalIp [in] Local IP Do not need to input. The default value is NULL dwWaitTime

Item	Description
	[in] User expected search time User sets the parameter according to actual requirements. Since it is the synchronization interface, it returns the value when the search time is finish.
Return value	Return TRUE for success, and return FALSE for failure

Use examples	DWORD dwWaitTime = 5000;
	// Please note the interface only returns when time is out. Set the timeout period according to network environment.
	if (FALSE ==
	CLIENT_SearchDevicesByIPs(&stuTmp, SearchDevicesCB, (LDWORD)&g_lDeviceList, szLocalIp, dwWaitTime))
	{
	printf("CLIENT_SearchDevicesByIPs Failed!Last Error[%x]\n", CLIENT_GetLastError());
	sreturn;
	}

Note	It is the synchronization interface. The interface only returns when search time starts . Set the search period according to network environment.

6.47 CLIENT_RealLoadPictureEx

Table 6-47 CLIENT_RealLoadPictureEx

Item	Description
Interface description	Intelligent picture alarm subscription interface
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	LLONG CLIENT_RealLoadPictureEx(LLONG lLoginID, int nChannelID, DWORD dwAlarmType, BOOL bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);
Parameter	lLoginID *[in]* Device login ID

Item	Description
	Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity nChannelID *[in]* Intelligent picture alarm subscription interface channel No. Channel No. begins with 0 dwAlarmType *[in]* The alarm type to be subscripted Such as:EVENT_IVS_ALL //Upload all alarm info Refer to dhnetsdk.h for more types. bNeedPicFile *[in]* Subscribe picture file or not TRUE: subscribe picture file. The callback function returns the intelligent picture information. FALSE:Do not subscribe picture file. The callback function does not return the intelligent picture info (It reduces the network flows when there is no picture information.) cbAnalyzerData [in] Intelligent picture alarm callback function SDK calls the return value of the function to user when there is uploaded intelligent picture alarm from the device. dwUser *[in]* User data. SDK sends the data to user for further use by callback fAnalyzerDataCallBack Reserved [in] Reserved parameter Fill in NULL in the field.
Return value	Return 0 for failure, return intelligent pictures alarm subscription ID as the parameter of CLIENT_StopLoadPic

Use examples	// Intelligent picture alarm subscription
	LDWORD dwUser = 0;
	g_lRealLoadHandle = CLIENT_RealLoadPictureEx(g_lLoginHandle, 0, EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, dwUser, NULL);
	if (0 == g_lRealLoadHandle)
	{
	printf("CLIENT_RealLoadPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	return;

Item	Description
	}
Note	Each interface is corresponding to one channel and one event type at each time. Set dwAlarmType as EVENT_IVS_ALL if user wants to subscribe all event types of current channel. If user wants to subscribe one channel to upload two event types, call CLIENT_RealLoadPictureEx twice and then input different types. Call CLIENT_StopLoadPic to cancel subscription

6.48 CLIENT_ControlDeviceEx

Table 6-48 CLIENT_ControlDeviceEx

Item	Description
Interface description	Device control extension interface
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_ControlDeviceEx(LLONG lLoginID, CtrlType emType, void* pInBuf, void* pOutBuf = NULL, int nWaitTime = 1000);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity emType [in] Control type Working with pInBuf and pOutBuf,different emType,pInBuf and pOutBuf point to different structures. Refer to enumeration note of CtrlType. pInBuf *[in]* Input parameter of device control Working with emType. Different emType and pInBuf point to different structures. Refer to enumeration note of enum CtrlType for details.Fill in NULL if the value of emType did not indicate what structure pInBuf is.

Item	Description
	pOutBuf *[out]* Output parameter of device control. It is NULL by default. Working with emType,different emType and pInBuf point to different structures. Refer to enumeration note of CtrlType for details.Fill in NULL if the value of emType not indicate what struct pOutBuf is. Do not need to fill in pOutBuf if emType is less than 0x10000 nWaitTime *[in]* Timeout when waiting for device to return. Unit is ms It is 1000 by default.
Return value	Return TRUE for success, and return FALSE for failure

Use examples	MANUAL_SNAP_PARAMETER stuSanpParam = {0};
	stuSanpParam.nChannel = 0;
	memcpy(stuSanpParam.bySequence, "just for test", sizeof(stuSanpParam.bySequence) - 1);
	// Manual snapshot triggers alarm function. For ITC only.
	if (FALSE == CLIENT_ControlDeviceEx(g_lLoginHandle, DH_MANUAL_SNAP, &stuSanpParam))
	{
	printf("CLIENT_ControlDeviceEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	break;
	}

Note	None

6.49 CLIENT_StopLoadPic

Table 6-49 CLIENT_StopLoadPic

Item	Description
Interface description	Cancel intelligent picture alarm subscription interface
Pre-condition	Already called intelligent picture alarm subscription interface such as CLIENT_RealLoadPictureEx
Function	BOOL CLIENT_StopLoadPic(LLONG lAnalyzerHandle

Item	Description
	);
Parameter	lAnalyzerHandle *[in]* Intelligent picture alarm subscription ID Corresponding intelligent picture alarm subscription interface such as Return value of CLIENT_RealLoadPictureEx
Return value	Return TRUE for success, and return FALSE for failure

Use examples	/ / Cancel intelligent picture alarm subscription
	if (0 != g_lRealLoadHandle)
	{
	if (FALSE ==
	CLIENT_StopLoadPic(g_lRealLoadHandle))
	{
	printf("CLIENT_StopLoadPic Failed!Last Error[%x]\n", CLIENT_GetLastError());
	}
	else
	{
	g_lRealLoadHandle = 0;
	}
	}

Note	None

6.50 CLIENT_GetDownloadPos

Table 6-50 CLIENT_GetDownloadPos

Item	Description
Interface description	Search download process. The unit is KB.
Pre-condition	Already called record download interface such as CLIENT_DownloadByTimeEx
Function	BOOL CLIENT_GetDownloadPos(LLONG lFileHandle, int *nTotalSize, int *nDownLoadSize);
Parameter	lFileHandle [in] Download handle Corresponding return value of record download interface such as CLIENT_DownloadByTimeEx nTotalSize *[out]* Downloaded total size. The unit is KB nDownLoadSize *[out]* Downloaded total length. The unit is KB

Item	Description
Return value	Return TRUE for success, and return FALSE for failure

Use examples	int nTotal = 0;
	int nDownLoad = 0;
	if (FALSE ==
	CLIENT_GetDownloadPos(g_lDownloadHandle, &nTotal, &nDownLoad))
	{
	printf("CLIENT_GetDownloadPos Failed!Last Error[%x]\n", CLIENT_GetLastError());
	}

Note	None

6.51 CLIENT_SetSnapRevCallBack

Table 6-51 CLIENT_SetSnapRevCallBack

Item	Description
Interface description	Set front-end video snapshot callback function interface
Pre-condition	Already called CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	void CLIENT_SetSnapRevCallBack(fSnapRev OnSnapRevMessage, LDWORD dwUser);
Parameter	OnSnapRevMessage *[In]* Front-end video snapshot callback function Refer to callback function note of fSnapRev dwUser *[in]* User data. SDK sends the data to user for further use by callback function fSnapRev.
Return value	None
Use examples	*In]* Set front-end video snapshot callback function CLIENT_SetSnapRevCallBack(SnapRev, NULL);
Note	Call CLIENT_SetSnapRevCallBack before calling front-end video snapshot interface

6.52 CLIENT_SnapPictureEx

Table 6-52 CLIENT_SnapPictureEx

Item	Description
Interface description	Snapshot request extension interface
Pre-condition	Already called

Item	Description
	CLIENT_LoginWithHighLevelSecurity to log in to the device.
Function	BOOL CLIENT_SnapPictureEx(LLONG lLoginID, SNAP_PARAMS *par, int *reserved = 0);
Parameter	lLoginID *[in]* Device login ID Corresponding return value of device login interface of CLIENT_LoginWithHighLevelSecurity par *[in]* Snapshot parameters Refer to the structure note of SNAP_PARAMS reserved *[in]* Reserved field
Return value	Return TRUE for success, and return FALSE for failure

Use examples	// Send out snapshot command to front-end device
	SNAP_PARAMS stuSnapParams;
	stuSnapParams.Channel = nChannelId;
	stuSnapParams.mode = nSnapType;
	stuSnapParams.CmdSerial = ++g_nCmdSerial; // Snapshot request SN. The value ranges from 0 to 65535. Once the value is out of range, it is unsigned short.
	if (FALSE == CLIENT_SnapPictureEx(g_lLoginHandle, &stuSnapParams))
	{
	printf("CLIENT_SnapPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
	return;
	}
	else
	{
	printf("CLIENT_SnapPictureEx succ\n");
	}

Note	None

Appendix 1 Cybersecurity Recommendations

Cybersecurity is more than just a buzzword: it's something that pertains to every device that is connected to the internet. IP video surveillance is not immune to cyber risks, but taking basic steps toward protecting and strengthening networks and networked appliances will make them less susceptible to attacks. Below are some tips and recommendations on how to create a more secured security system.

Mandatory actions to be taken for basic equipment network security:

1. **Use Strong Passwords**

   Please refer to the following suggestions to set passwords:

   - The length should not be less than 8 characters;
   - Include at least two types of characters; character types include upper and lower case letters, numbers and symbols;
   - Do not contain the account name or the account name in reverse order;
   - Do not use continuous characters, such as 123, abc, etc.;
   - Do not use overlapped characters, such as 111, aaa, etc.;

2. **Update Firmware and Client Software in Time**

   - According to the standard procedure in Tech-industry, we recommend to keep your equipment (such as NVR, DVR, IP camera, etc.) firmware up-to-date to ensure the system is equipped with the latest security patches and fixes. When the equipment is connected to the public network, it is recommended to enable the "auto-check for updates" function to obtain timely information of firmware updates released by the manufacturer.
   - We suggest that you download and use the latest version of client software.

"Nice to have" recommendations to improve your equipment network security:

1. **Physical Protection**

   We suggest that you perform physical protection to equipment, especially storage devices. For example, place the equipment in a special computer room and cabinet, and implement well-done access control permission and key management to prevent unauthorized personnel from carrying out physical contacts such as damaging hardware, unauthorized connection of removable equipment (such as USB flash disk, serial port), etc.

2. **Change Passwords Regularly**

   We suggest that you change passwords regularly to reduce the risk of being guessed or cracked.

3. **Set and Update Passwords Reset Information Timely**

   The equipment supports password reset function. Please set up related information for password reset in time, including the end user's mailbox and password protection questions. If the information changes, please modify it in time. When setting password protection questions, it is suggested not to use those that can be easily guessed.

4. **Enable Account Lock**

   The account lock feature is enabled by default, and we recommend you to keep it on to guarantee the account security. If an attacker attempts to log in with the wrong password several times, the corresponding account and the source IP address will be locked.

5. **Change Default HTTP and Other Service Ports**

   We suggest you to change default HTTP and other service ports into any set of numbers between 1024~65535, reducing the risk of outsiders being able to guess which ports you are using.

6. **Enable HTTPS**

   We suggest you to enable HTTPS, so that you visit Web service through a secure communication channel.

7. **Enable Allowlist**

   We suggest you to enable allowlist function to prevent everyone, except those with specified IP addresses, from accessing the system. Therefore, please be sure to add your computer's IP address and the accompanying equipment's IP address to the allowlist.

8. **MAC Address Binding**

   We recommend you to bind the IP and MAC address of the gateway to the equipment, thus reducing the risk of ARP spoofing.

9. **Assign Accounts and Privileges Reasonably**

   According to business and management requirements, reasonably add users and assign a minimum set of permissions to them.

10. **Disable Unnecessary Services and Choose Secure Modes**

    If not needed, it is recommended to turn off some services such as SNMP, SMTP, UPnP, etc., to reduce risks.

    If necessary, it is highly recommended that you use safe modes, including but not limited to the following services:

    - SNMP: Choose SNMP v3, and set up strong encryption passwords and authentication passwords.
    - SMTP: Choose TLS to access mailbox server.
    - FTP: Choose SFTP, and set up strong passwords.
    - AP hotspot: Choose WPA2-PSK encryption mode, and set up strong passwords.

11. **Audio and Video Encrypted Transmission**

    If your audio and video data contents are very important or sensitive, we recommend that you use encrypted transmission function, to reduce the risk of audio and video data being stolen during transmission.

    Reminder: encrypted transmission will cause some loss in transmission efficiency.

12. **Secure Auditing**

    - Check online users: we suggest that you check online users regularly to see if the device is logged in without authorization.
    - Check equipment log: By viewing the logs, you can know the IP addresses that were used to log in to your devices and their key operations.

13. **Network Log**

    Due to the limited storage capacity of the equipment, the stored log is limited. If you need to save the log for a long time, it is recommended that you enable the network log function to ensure that the critical logs are synchronized to the network log server for tracing.

14. **Construct a Safe Network Environment**

    In order to better ensure the safety of equipment and reduce potential cyber risks, we recommend:

    - Disable the port mapping function of the router to avoid direct access to the intranet devices from external network.
    - The network should be partitioned and isolated according to the actual network needs. If there are no communication requirements between two sub networks, it is suggested to use VLAN, network GAP and other technologies to partition the network, so as to achieve the network isolation effect.

- Establish the 802.1x access authentication system to reduce the risk of unauthorized access to private networks.
- It is recommended that you enable your device's firewall or blocklsit and allowlist feature to reduce the risk that your device might be attacked.