NetSDK Programming Manual
(Storage)

V1.0.0

Foreword

Purpose

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

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 SDK interfaces and processes of the general function modules for Network Video Recorder (NVR), Enterprise Video Storage (EVS), and High Definition Composite Video Interface (HDCVI). For more function modules and data structures, refer to *NetSDK Development Manual*.

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

Reader

- SDK software development engineers
- Project managers
- Product managers

Signals

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

Signal Words	Meaning
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

No.	Version	Revision Content	Release Time
1	V1.0.0	First Release.	December 30, 2017

About the Manual

- The Manual is for reference only. If there is inconsistency between the Manual and the actual product, the actual product shall govern.

- 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 between the actual value of some data and the value provided, if there is any doubt or dispute, please refer to our final explanation.
- Please contact the supplier or customer service if there is any problem occurred when using the device.
- We are not liable for any loss caused by the operations that do not comply with the Manual.
- All trademarks, registered trademarks and the company names in the Manual are the properties of their respective owners.
- Please visit our website or contact your local service engineer for more information. If there is any uncertainty or controversy, please refer to our final explanation.

Glossary

This chapter provides the definitions to some of the terms appear in the Manual to help you understand the function of each module.

Term	Definition
NVR	Abbreviation for Network Video Recorder.
EVS	Abbreviation for Enterprise Video Storage.
HDCVI	Abbreviation for High Definition Composite Video Interface.
Main Stream	A type of video stream that usually has better resolution and clarity and provides a better experience if the network resource is not restricted.
Sub Stream	A type of video stream that usually has lower resolution and clarity than the main stream but demands less network resources. The user can choose the stream type according to the particular scenes.
Resolution	Resolution is consisted of display resolution and image resolution. Display resolution refers to the quantity of pixels in unit area, and the image resolution refers to information quantity (the quantity of pixels per inch) stored in the image.
Frame Rate	A measurement, usually in FPS and Hz, which shows the frames of video. The more the frame, more smooth the video. The frames over 24 FPS make the image feels coherent.
Video Channel	An abstract concept of the communication and video stream transmission between SDK and devices. For example, if a number of cameras (SD, IPC) are mounted on a storage device (NVR), the storage device manages the cameras as video channels which are numbered from 0. If SDK connects to the camera directly, the video channel is usually numbered as 0.
Alarm by Dynamic Detection	When detecting a moving object on the image, an alarm by dynamic detection will be uploaded.
Alarm of Hard Disk Failure	When detecting a hard disk failure, an alarm will be uploaded.
Alarm for Video Loss	This alarm is only for analog channel. When the record disappeared from the analog channel, an alarm will be uploaded. For the digital channel, refer to IPC disconnection alarm.
Alarm for Hard Disk Damage	When the hard disk is damaged, an alarm will be uploaded.
IPC Disconnection Alarm	When the IPC is disconnected, an alarm will be uploaded.
External Alarm	NVR local alarm. When the NVR alarm terminal connects with alarm device, an alarm will be uploaded.
IPC External Alarm	When the alarm on IPC device connects with alarm device, an external alarm will be uploaded.

Table of Contents

1 Overview

1.1 General

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

The following are the main functions:

SDK initialization, device login, real-time monitoring, record playback, record download, bidirectional talk, video snapshot, alarm upload, and storage.

The development kit might be different dependent on the environment.

- For the files included in Windows development kit, see Table 1-1.

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
	aacdec.dll	AAC audio decoding library
	amrenc.dll	AMR audio decoding library
	g729dec.dll	G729 audio decoding library
	g7221dec.dll	G7221 audio decoding library
	h264dec.dll	H264 video decoding library
	mjpegdec.dll	MJPEG video decoding library
	mp3dec.dll	MP3 audio decoding library
	hevcdec.dll	H265 video decoding library
	mp2dec.dll	MP2 audio decoding library
	mpeg4dec.dll	MPEG4 video decoding library
	oggdec.dll	OGG audio decoding library
	adpcmdec.dll	ADPCM audio decoding library
	svac_dec.dll	SVAC video decoding library
	fisheye.dll	Fisheye correction library
	MCL_FPTZ.dll	Library used for fisheye and speed dome linkage, matching of fisheye correction library
	postproc.dll	Image processing library
	swscale.dll	Library used for GDI display
Dependent library of	Infra.dll	Infrastructure library

Library type	Library file name	Library file description
"avnetsdk.dll"	json.dll	JSON library
	NetFramework.dll	Network infrastructure library
	Stream.dll	Media transmission structure package library
	StreamSvr.dll	Streaming service
Auxiliary library of "dhnetsdk.dll"	IvsDrawer.dll	Image display library

Table 1-1

- For the files included in Linux development kit, see Table 1-2.

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 "libavnetsdk.so"	libInfra.so	Infrastructure library
	libNetFramework.so	Network infrastructure library
	libStream.so	Media transmission structure package library
	libStreamSvr.so	Streaming service

Table 1-2

NOTE

- 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.
- If the function library includes avnetsdk.dll or libavnetsdk.so, the corresponding dependent library is necessary.

1.2 Applicability

- Recommended memory: No less than 512 M
- System supported by SDK:
  - ◇ Windows
    Windows 10/Windows 8.1/Windows 7 and Windows Server 2008/2003
  - ◇ Linux
    The common Linux systems such as Red Hat/SUSE

- Applicable devices include but not limited to the following: EVS5016/5024/5036/5048 series, EVS7024/7036/7048/7064/7072 series, MCS7024, NVR4832-4KS2, NVR4832, NVR3XX, NVR5XX and NVR724-256 series

1.3 Application Scenario

You can access to the channels of NVR through SDK to get the monitoring data of the IPC and save the videos for future playback and download.

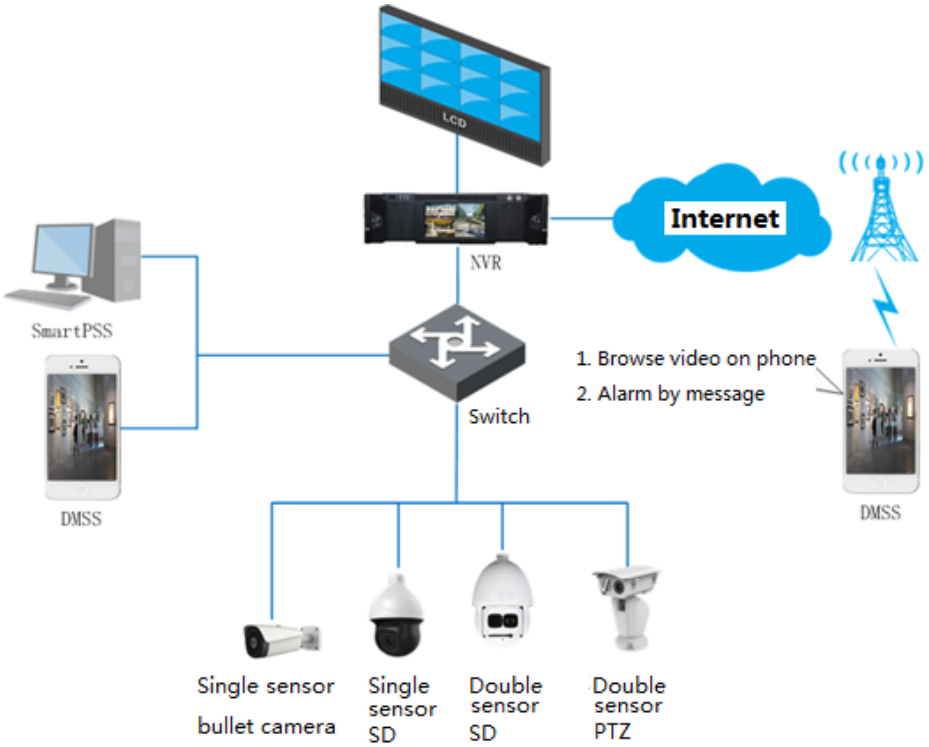

Figure 1-1

2 Function Modules

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 **CLIENT_Cleanup** to release SDK resource.

2.1.2 Interface Overview

Interface	Implication
CLIENT_Init	SDK initialization
CLIENT_Cleanup	SDK cleaning up
CLIENT_SetAutoReconnect	Setting of reconnection after disconnection
CLIENT_SetNetworkParam	Setting of network environment

Table 2-1

2.1.3 Process

For the process of SDK initialization, see Figure 2-1.

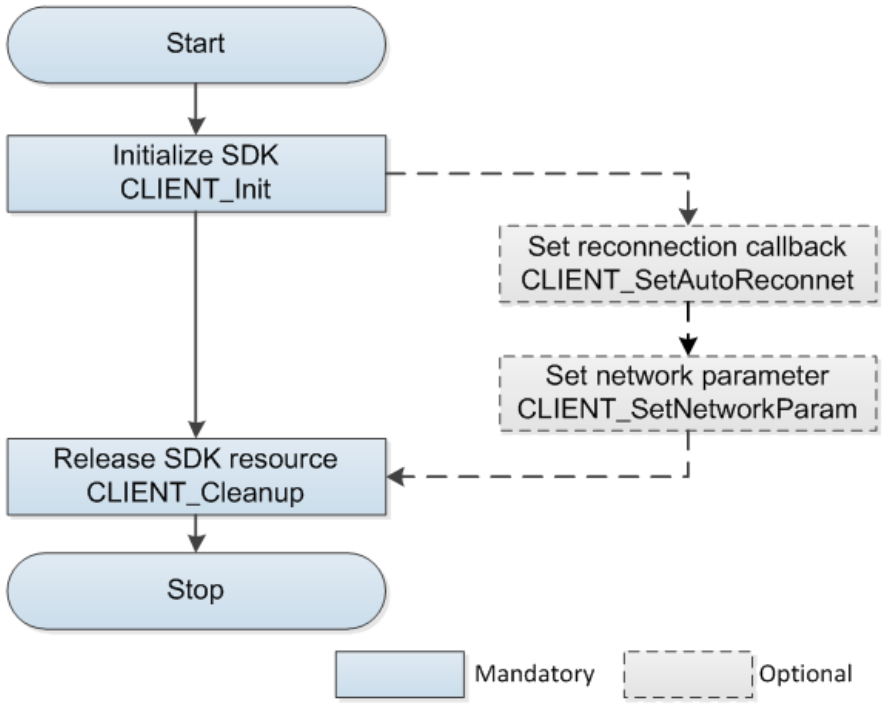

Figure 2-1

Process Description

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

Step 2 (Optional) Call **CLIENT_SetAutoReconnect** to set reconnection callback to allow the auto reconnecting after disconnection.

Step 3 (Optional) Call **CLIENT_SetNetworkParam** to set network login parameter that includes connection timeout and connection attempts.

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

Notes for Process

- Call **CLIENT_Init** and **CLIENT_Cleanup** in pairs. It supports multiple calling but it is suggested to call the pair for only one time overall.
- Initialization: Calling **CLIENT_Init** multiple times is only for internal count without repeating applying resources.
- Cleaning up: The interface **CLIENT_Cleanup** clears all the opened processes, such as login, real-time monitoring, and alarm subscription.
- Reconnection: SDK can set the reconnection function for the situations such as network disconnection and power off. SDK will keep logging until succeeded. Only the real-time monitoring and playback function modules will be resumed after the connection is back.

2.1.4 Example Code

```
// Set this callback through CLIENT_Init. When the device is disconnected, SDK informs the user through this callback.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser)
```

```
{
    printf("Call DisConnectFunc: lLoginID[0x%x]\n", lLoginID);
}
// Initialize SDK
CLIENT_Init(DisConnectFunc, 0);

// .... Call the functional interface to handle the process

// Clean up the SDK resource
CLIENT_Cleanup();
```

2.2 Device Initialization

2.2.1 Introduction

The device is uninitialized by default. Please initialize the device before starting use.

- The uninitialized device cannot be logged.
- A password will be set for the default admin account during initialization.
- You can reset the password if you forgot it.

2.2.2 Interface Overview

Interface	Implication
CLIENT_StartSearchDevices	Search in the LAN to find the uninitialized devices.
CLIENT_InitDevAccount	Initialization interface.
CLIENT_GetDescriptionForResetPwd	Get the password reset information: mobile phone number, email address, and QR code.
CLIENT_CheckAuthCode	Check the validity of security code.
CLIENT_ResetPwd	Reset password.
CLIENT_GetPwdSpecification	Get the password rules.
CLIENT_StopSearchDevices	Stop searching.

Table 2-2

2.2.3 Process

2.2.3.1 Device Initialization

For the process of device initialization, see Figure 2-2.

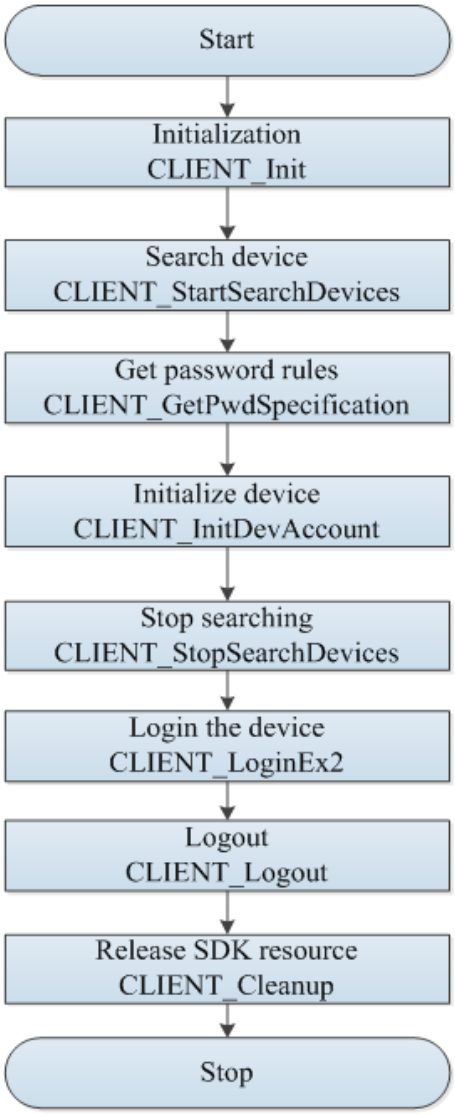

Figure 2-2

Process Description

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

Step 2 Call **CLIENT_StartSearchDevices** to search the devices within the LAN and get the device information.

NOTE

Multi-thread calling is not supported.

Step 3 Call **CLIENT_GetPwdSpecification** to get the password rules.

Step 4 Call **CLIENT_InitDevAccount** to initialize device.

Step 5 Call **CLIENT_StopSearchDevices** to stop searching.

Step 6 Call **CLIENT_LoginEx2** and login the admin account with the configured password.

Step 7 After using the function module, call **CLIENT_Logout** to logout the device.

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

Notes for Process

Because the interface is working in multicast, the host PC and device must be in the same multicast group.

2.2.3.2 Password Reset

For the process of device initialization, see Figure 2-3.

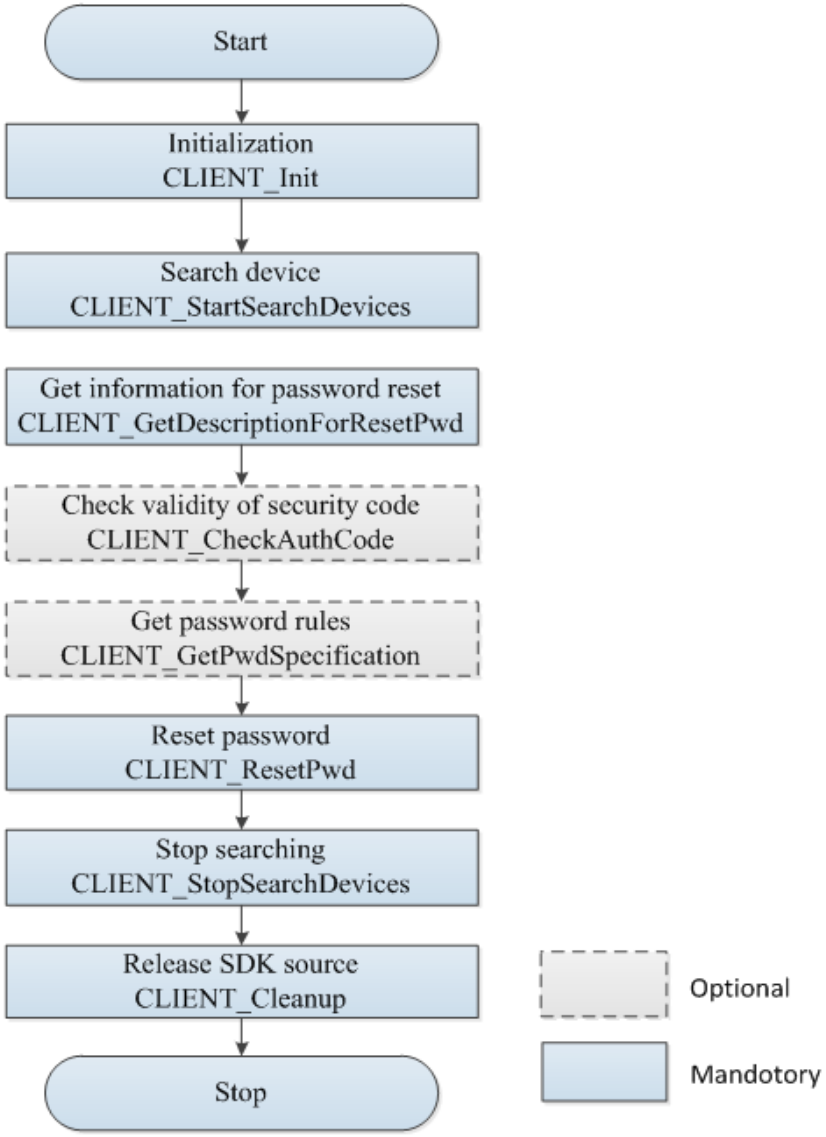

Figure 2-3

Process Description

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

Step 2 Call **CLIENT_StartSearchDevices** to search the devices within the LAN and get the device information.

NOTE

Multi-thread calling is not supported.

Step 3 Call **CLIENT_GetDescriptionForResetPwd** to get the information for password reset.

Step 4 (Optional) Scan the QR code obtained from the previous step to get the security code, and then validate it through **CLIENT_CheckAuthCode**.

Step 5 (Optional) Call **CLIENT_GetPwdSpecification** to get the password rules.

Step 6 Call **CLIENT_ResetPwd** to reset the password.

Step 7 Call **CLIENT_StopSearchDevices** to stop searching.

Step 8 Call **CLIENT_LoginEx2** and login the admin account with the configured password.

Step 9 After using the function module, call **CLIENT_Logout** to logout the device.
Step 10 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

Because the interface is working in multicast, the host PC and device must be in the same multicast group.

2.2.4 Example Code

2.2.4.1 Device Initialization

```
//Firstly, call CLIENT_StartSearchDevices to get the device information.
//Get the password rules
NET_IN_PWD_SPECI stIn = {sizeof(stIn)};
strncpy(stIn.szMac, szMac, sizeof(stIn.szMac) - 1);
NET_OUT_PWD_SPECI stOut = {sizeof(stOut)};
CLIENT_GetPwdSpecification(&stIn, &stOut, 3000, NULL);//In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter. Set the password according to the rules which are used for preventing user from setting the passwords that are not supported by the device.

//Device initialization
NET_IN_INIT_DEVICE_ACCOUNT sInitAccountIn = {sizeof(sInitAccountIn)};
NET_OUT_INIT_DEVICE_ACCOUNT sInitAccountOut = {sizeof(sInitAccountOut)};
sInitAccountIn.byPwdResetWay = 1;//1 stands for password reset by mobile phone number, and 2 stands for password reset by email
strncpy(sInitAccountIn.szMac, szMac, sizeof(sInitAccountIn.szMac) - 1);//Set mac value
strncpy(sInitAccountIn.szUserName, szUserName, sizeof(sInitAccountIn.szUserName) - 1);//Set user name
strncpy(sInitAccountIn.szPwd, szPwd, sizeof(sInitAccountIn.szPwd) - 1);//Set password
strncpy(sInitAccountIn.szCellPhone, szRig, sizeof(sInitAccountIn.szCellPhone) - 1);//If the byPwdResetWay is set as 1, please set szCellPhone field; if the byPwdResetWay is set as 2, please set sInitAccountIn.szMail field.
CLIENT_InitDevAccount(&sInitAccountIn, &sInitAccountOut, 5000, NULL);
```

2.2.4.2 Password Reset

```
//Firstly, call CLIENT_StartSearchDevices to get the device information.
//Get the information for password reset
NET_IN_DESCRIPTION_FOR_RESET_PWD stIn = {sizeof(stIn)};
strncpy(stIn.szMac, szMac, sizeof(stIn.szMac) - 1); //Set mac value
strncpy(stIn.szUserName, szUserName, sizeof(stIn.szUserName) - 1);//Set user name
stIn.byInitStatus = bStstus; //bStstus is the value of return field byInitStatus of device search interface (Callback
```

of CLIENT_SearchDevices and CLIENT_StartSearchDevice, and CLIENT_SearchDevicesByIPs)

NET_OUT_DESCRIPTION_FOR_RESET_PWD stOut = {sizeof(stOut)};

char szTemp[360];

stOut.pQrCode = szTemp;

CLIENT_GetDescriptionForResetPwd(&stIn, &stOut, 3000, NULL);//In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter. After successful connection, stout will output a QR code with address of stOut.pQrCode. Scan this QR code to get the security code for password reset. This security code will be sent to the reserved mobile phone or email box.

//(Optional) Check the security code

NET_IN_CHECK_AUTHCODE stIn1 = {sizeof(stIn1)};

strncpy(stIn1.szMac, szMac, sizeof(stIn1.szMac) - 1); //Set mac value

strncpy(stIn1.szSecurity, szSecu, sizeof(stIn1.szSecurity) - 1); // szSecu is the security code sent to the reserved mobile phone or email box

NET_OUT_CHECK_AUTHCODE stOut1 = {sizeof(stOut1)};

bRet = CLIENT_CheckAuthCode(&stIn1, &stOut1, 3000, NULL); //In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter

//Get password rules

NET_IN_PWD_SPECI stIn2 = {sizeof(stIn2)};

strncpy(stIn2.szMac, szMac, sizeof(stIn2.szMac) - 1); //Set mac value

NET_OUT_PWD_SPECI stOut2 = {sizeof(stOut2)};

CLIENT_GetPwdSpecification(&stIn2, &stOut2, 3000, NULL);// In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter. Set the password according to the rules which are used for preventing user from setting the passwords that are not supported by the device

//Reset password

NET_IN_RESET_PWD stIn3 = {sizeof(stIn3)};

strncpy(stIn3.szMac, szMac, sizeof(stIn3.szMac) - 1); //Set mac value

strncpy(stIn3.szUserName, szUserName, sizeof(stIn3.szUserName) - 1); //Set user name

strncpy(stIn3.szPwd, szPassWd, sizeof(stIn3.szPwd) - 1); //szPassWd is the password reset according to the rules

strncpy(stIn3.szSecurity, szSecu, sizeof(stIn1.szSecurity) - 1); //szSecu is the security code sent to the reserved mobile phone or email box

stIn3.byInitStaus = bStstus; //bStstus is the value of return field byInitStatus of device search interface (Callback of CLIENT_SearchDevices and CLIENT_StartSearchDevice, and CLIENT_SearchDevicesByIPs)

stIn3.byPwdResetWay = bPwdResetWay; // bPwdResetWay is the value of return field byPwdResetWay of device search interface (Callback of CLIENT_SearchDevices and CLIENT_StartSearchDevice, and CLIENT_SearchDevicesByIPs)

NET_OUT_RESET_PWD stOut3 = {sizeof(stOut3)};

CLIENT_ResetPwd(&stIn3, &stOut3, 3000, NULL);//In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter.

2.3 Device Login

2.3.1 Introduction

Device login, also called user authentication, is the precondition of all the other function modules.

You can obtain a unique login ID upon logging in to the device and should call login ID before using other SDK interfaces. The login ID becomes invalid once logged out.

2.3.2 Interface Overview

Interface	Implication
CLIENT_LoginEx2	Login.
CLIENT_Logout	Logout.

Table 2-3

2.3.3 Process

For the process of login, see Figure 2-4.

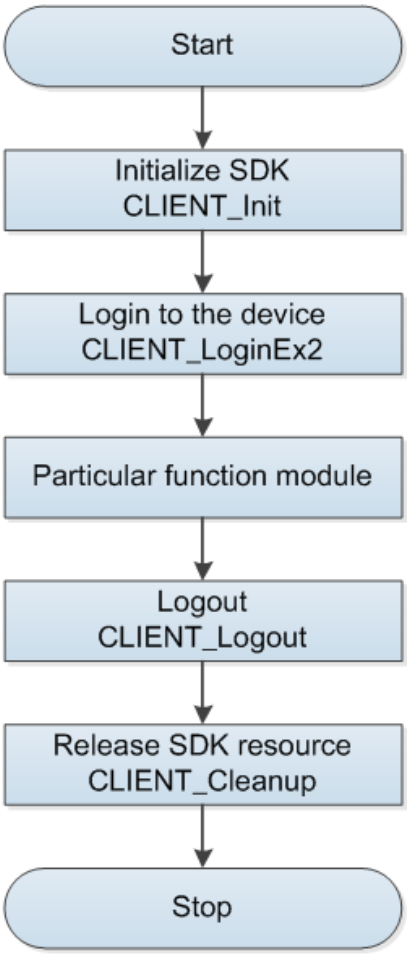

Figure 2-4

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 After successful login, you can realize the required function module.
Step 4 After using the function module, call **CLIENT_Logout** to logout the device.
Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Login handle: When the login is successful, the returned value is not 0 (even the handle is smaller than 0, the login is also successful). One device can login multiple times with different handle at each login. If there is not special function module, it is suggested to login only one time. The login handle can be repeatedly used on other function modules.
- Logout: The interface will release the opened functions internally, but it is not suggested to rely on the cleaning up function. For example, if you opened the monitoring function, you should call the interface that stops the monitoring function when it is no longer required.
- Use login and logout in pairs: The login consumes some memory and socket information and release sources once logout.
- Login failure: It is suggested to check the failure through the error parameter of the login interface. For the common error code, see Table 2-4.

Error code	Meaning
1	Password is wrong.
2	User name does not exist.
3	Login timeout.
4	The account has been logged in.
5	The account has been locked.
6	The account is blacklisted.
7	Out of resources, the system is busy.
8	Sub connection failed.
9	Main connection failed.
10	Exceeded the maximum user connections.
11	Lack of avnetsdk or avnetsdk dependent library.
12	USB flash disk is not inserted into device, or the USB flash disk information error.
13	The client IP is not authorized with login.

Table 2-4

The example code to avoid error code 3 is as follows.

```
NET_PARAM stuNetParam = {0};
stuNetParam.nWaittime = 8000; // unit ms
CLIENT_SetNetworkParam (&stuNetParam);
```

For more information about error codes, see "CLIENT_LoginEx2 interface" in *Network SDK Development Manual.chm*.

2.3.4 Example Code

```
NET_DEVICEINFO_Ex stDevInfo = {0};
int nError = 0;
// Login the device
LLONG lLoginHandle = CLIENT_LoginEx2(szDevIp, nPort, szUserName, szPasswd,
      EM_LOGIN_SPEC_CAP_TCP, NULL, &stDevInfo, &nError);

// Logout the device
if (0 != lLoginHandle)
{
      CLIENT_Logout(lLoginHandle);
}
```

2.4 Real-time Monitoring

2.4.1 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.

- Supports calling the window handle for SDK to directly decode and play the stream (Windows system only).
- Supports calling the real-time stream to you for independent treatment.
- Supports saving the real-time record to the specific file though saving the callback stream or calling the SDK interface.

2.4.2 Interface Overview

Interface	Implication
CLIENT_RealPlayEx	Start real-time monitoring extension interface.
CLIENT_StopRealPlayEx	Stop real-time monitoring extension interface.
CLIENT_SaveRealData	Start saving the real-time monitoring data to the local path.
CLIENT_StopSaveRealData	Stop saving the real-time monitoring data to the local path.
CLIENT_SetRealDataCallBackEx2	Set real-time monitoring data callback function extension interface.

Table 2-5

2.4.3 Process

You can realize the real-time monitoring through SDK decoding library or your play library.

2.4.3.1 SDK Decoding Library

Call PlaySDK library from the SDK auxiliary library to realize real-time play.

For the process of playing by SDK decoding library, see Figure 2-5.

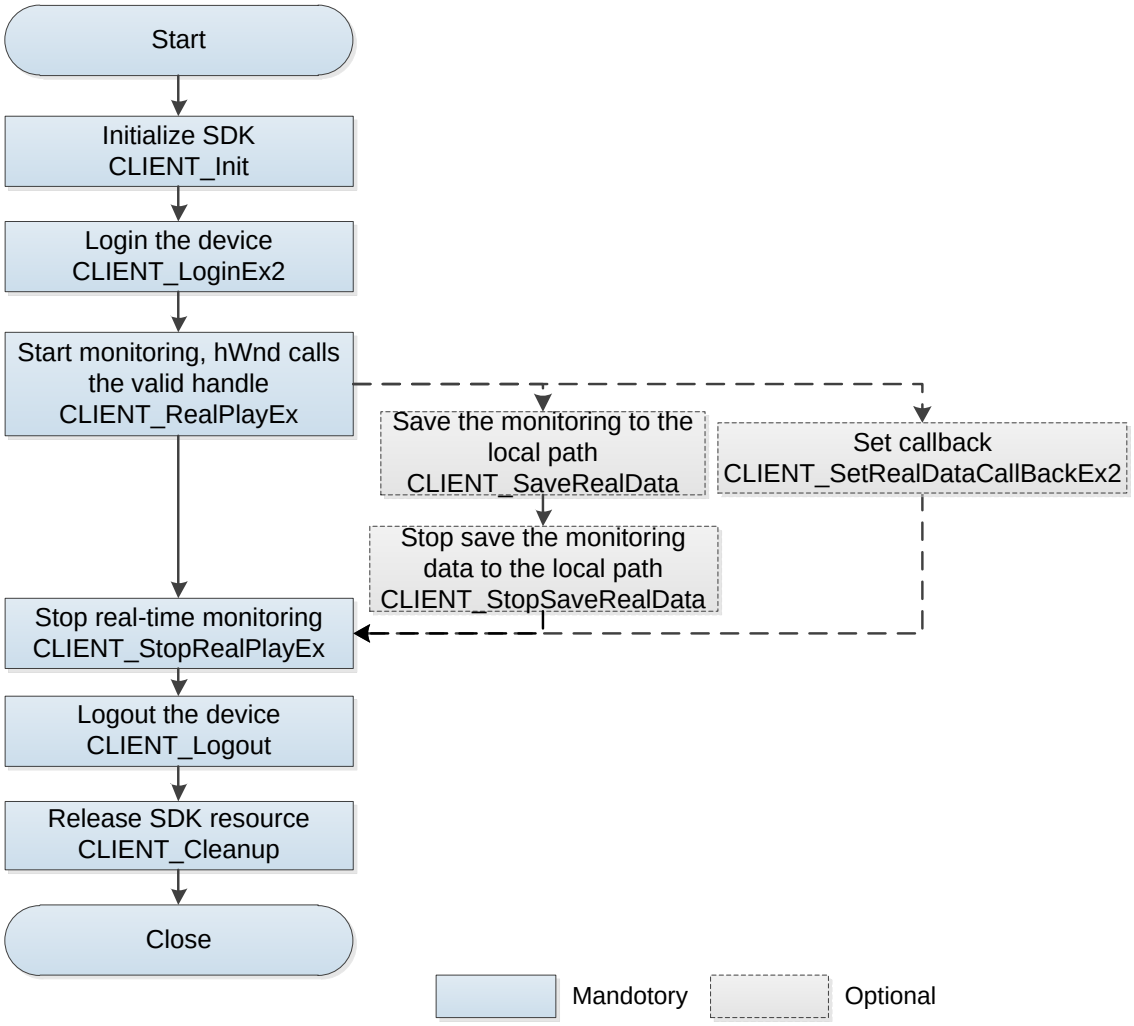

Figure 2-5

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

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

Step 4 (Optional) Call **CLIENT_SaveRealData** to start saving the monitoring data.

Step 5 (Optional) Call **CLIENT_StopSaveRealData** to end the saving process and generate the local video file.

Step 6 (Optional) If you call **CLIENT_SetRealDataCallBackEx2**, you can choose to save or forward the video file. If save the video file, see the step 4 and step 5.

Step 7 After using the real-time function, call **CLIENT_StopRealPlayEx** to stop real-time monitoring**.**

Step 8 After using the function module, call **CLIENT_Logout** to logout the device.

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

Notes for Process

- SDK decoding play only supports Windows system. You need to call the decoding after getting the stream in other systems.
- Multi-thread calling: Multi-thread calling is not supported for the functions within the same login session; however, multi-thread calling can deal with the functions of different login sessions although such calling is not recommended.
- Timeout: The request on applying for monitoring resources should have made some agreement with the device before requiring the monitoring data. There are some timeout settings (see "NET_PARAM structure"), and the field about monitoring is nGetConnInfoTime. If there is timeout due to the reasons such as bad network connection, you can modify the value of nGetConnInfoTime bigger.

  The example code is as follows. Call it for only one time after having called **CLIENT_Init**.

```
NET_PARAM stuNetParam = {0};
stuNetParam. nGetConnInfoTime = 5000; // unit ms
CLIENT_SetNetworkParam (&stuNetParam);
```

- Failed to repeat opening: For some models, the same channel cannot be opened for multiple times during a login. If you are trying to open it repeatedly, you will success in the first try but get failed afterwards. In this case, you can try the following:
  - ◇ Close the opened channel. For example, if you already opened the main stream video on the channel 1 and still want to open the sub stream video on the same channel, you can close the main stream first and then open the sub stream.
  - ◇ Login twice to obtain two login handles to deal with the main stream and sub stream respectively.
- Calling succeeded but no image: SDK decoding needs to use dhplay.dll. It is suggested to check if dhplay.dll and its auxiliary library are missing under the running directory. See Table 1-2.
- If the system resource is insufficient, the device might return error instead of stream. You can receive an event DH_REALPLAY_FAILD_EVENT in the alarm callback that is set in CLIENT_SetDVRMessCallBack. This event includes the detailed error codes. See "DEV_PLAY_RESULT Structure" in *Network SDK Development Manual.chm*.
- 32 channels limit: The decoding consumes resources especially for the high definition videos. Considering the limited resources at the client, currently the maximum channels are set to be 32. If more than 32, it is suggested to use third party play library. See "2.4.3.2 Call Third Party Play Library."

2.4.3.2 Call Third Party Play Library

SDK calls back the real-time monitoring stream to you and you call PlaySDK to decode and play.

For the process of calling the third party play library, see Figure 2-6.

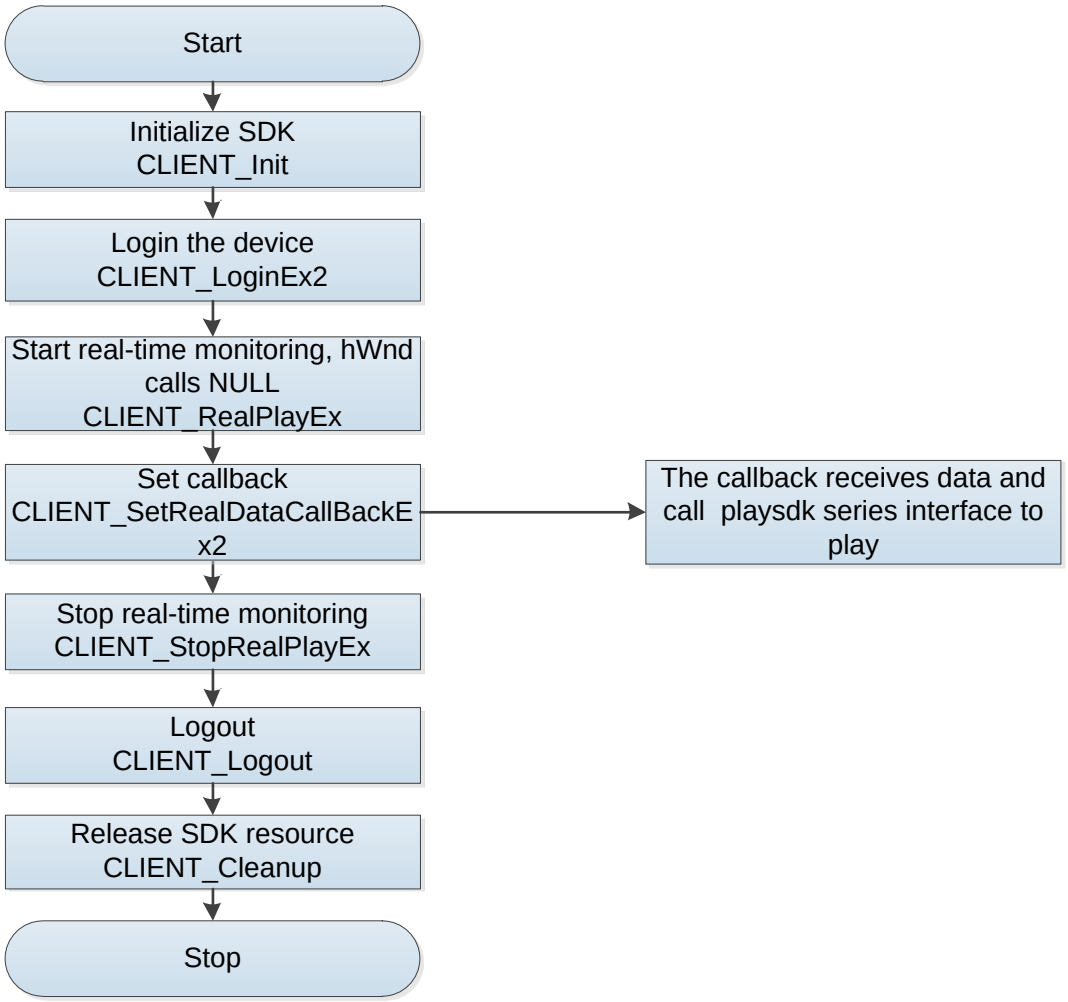

Figure 2-6

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

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

Step 4 Call **CLIENT_SetRealDataCallBackEx2** to set the real-time data callback.

Step 5 In the callback, pass the data to PlaySDK to finish decoding.

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 logout the device.

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

Notes for Process

- Stream format: It is recommended to use PlaySDK for decoding.
- Lag image
  - ◇ When using PlaySDK for decoding, there is a default channel cache size (the PLAY_OpenStream interface in playsdk) for decoding. If the stream resolution value is big, it is recommended to modify the parameter value smaller such as 3 M.

◇ SDK callbacks can only moves into the next process after returning from you. It is not recommended for you to consume time for the unnecessary operations; otherwise the performance could be affected.

2.4.4 Example Code

2.4.4.1 SDK Decoding Play

```
// Take opening the main stream monitoring of channel 1 as an example. The parameter hWnd is a window handle.
LLONG lRealHandle = CLIENT_RealPlayEx(lLoginHandle, 0, hWnd, DH_RType_Realplay);
if (NULL == lRealHandle)
{
    printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
printf("input any key to quit!\n");
getchar();
// Stop preview
if (NULL != lRealHandle)
{
    CLIENT_StopRealPlayEx(lRealHandle);
}
```

2.4.4.2 Call Play Library

```
void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LLONG param, LDWORD dwUser);
// Take opening the main stream monitoring of channel 1 as an example.
LLONG lRealHandle = CLIENT_RealPlayEx(lLoginHandle, 0, NULL, DH_RType_Realplay);
if (NULL == lRealHandle)
{
    printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
else
{
    DWORD dwFlag = REALDATA_FLAG_RAW_DATA; //Initial data labels
    CLIENT_SetRealDataCallBackEx2(lRealHandle, &RealDataCallBackEx, NULL, dwFlag);
}

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

```
if (0 != lRealHandle)
{
    CLIENT_StopRealPlayEx(lRealHandle);
}

void CALLBACK RealDataCallBackEx(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer,
DWORD dwBufSize, LLONG param, LDWORD dwUser)
{
    //Call PlaySDK interface to get the stream data from the device. See SDK monitoring demo source data for
more details.
    printf("receive real data, param: lRealHandle[%p], dwDataType[%d], pBuffer[%p], dwBufSize[%d]\n",
lRealHandle, dwDataType, pBuffer, dwBufSize);
}
```

2.5 Record Playback

2.5.1 Introduction

Record playback function plays the videos of a particular period in some channels to find the target videos for check.

The playback includes the following functions: Start playback, pause Playback, resume playback, and stop playback.

2.5.2 Interface Overview

Interface	Implication
CLIENT_PlayBackByTimeEx2	Playback by time.
CLIENT_SetDeviceMode	Set the work mode such as voice talk, playback, and authority.
CLIENT_StopPlayBack	Stop record playback.
CLIENT_GetPlayBackOsdTime	Get the playback OSD time.
CLIENT_PausePlayBack	Pause or resume playback.

Table 2-6

2.5.3 Process

After SDK initialization, you need to input channel number, start time, stop time, and valid window handle to realize the playback of the required record.

For the process of record playback, see Figure 2-7.

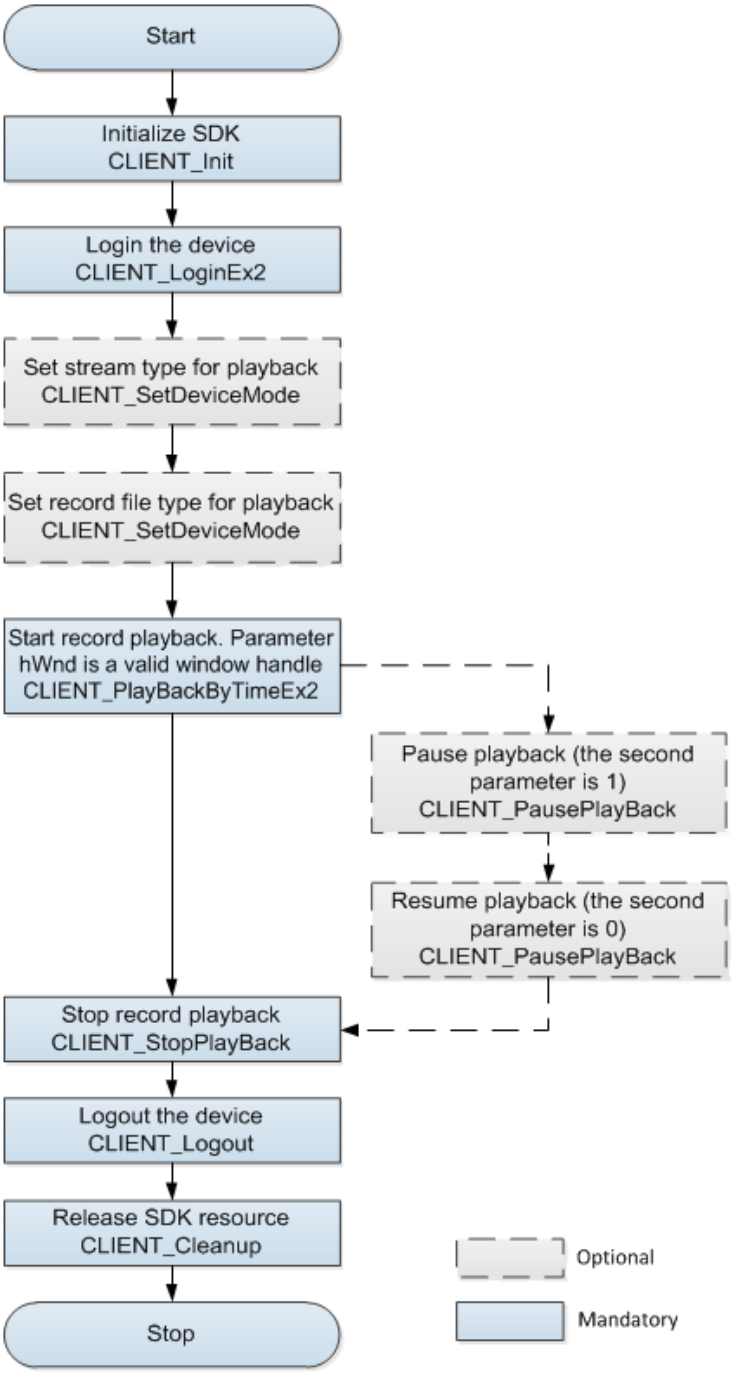

Figure 2-7

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 (Optional) Call **CLIENT_SetDeviceMode** twice and set the stream type parameter emType as DH_RECORD_STREAM_TYPE and the record type parameter emType DH_RECORD_TYPE.

Step 4 Call **CLIENT_PlayBackByTimeEx2** to start playback. The parameter hWnd is a valid window handle value.

Step 5 (Optional) Call **CLIENT_PausePlayBack**. The playback will pause when the second parameter is 1.

Step 6 (Optional) Call **CLIENT_PausePlayBack.** The playback will resume when the second parameter is 0.

Step 7 Call **CLIENT_StopPlayBack** to stop playback**.**

Step 8 After using the function module, call **CLIENT_Logout** to logout the device.

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

2.5.4 Example Code

2.5.4.1 Playback + Stop Playback

```
// Set the stream type as the main stream for playback
int nStreamType = 0; // 0-Main stream,1- Main stream,2-Sub stream
CLIENT_SetDeviceMode(lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

// Set the record file type as all records for playback
NET_RECORD_TYPE emFileType = NET_RECORD_TYPE_ALL; // All records
CLIENT_SetDeviceMode(lLoginHandle, DH_RECORD_TYPE, &emFileType);

// Start record playback
int nChannelID = 0; // Channel number
NET_TIME stuStartTime = {0}; // Start time of recording
stuStartTime.dwYear = 2015;
stuStartTime.dwMonth = 11;
stuStartTime.dwDay = 20;

NET_TIME stuStopTime = {0}; // Stop time of recording
stuStopTime.dwYear = 2015;
stuStopTime.dwMonth = 11;
stuStopTime.dwDay = 21;

NET_IN_PLAY_BACK_BY_TIME_INFO stIn = {0};
NET_OUT_PLAY_BACK_BY_TIME_INFO stOut = {0};
memcpy(&stIn.stStartTime, &stuStartTime, sizeof(stuStartTime));
memcpy(&stIn.stStopTime, &stuStopTime, sizeof(stuStopTime));
stIn.hWnd = hWnd;
stIn.fDownLoadDataCallBack = DataCallBack;
stIn.dwDataUser = NULL;
stIn.cbDownLoadPos = NULL;
stIn.dwPosUser = NULL;
stIn.nPlayDirection = emDirection;
```

```
stIn.nWaittime = 10000;

LLONG lPlayHandle = CLIENT_PlayBackByTimeEx2(lLoginHandle, nChannelID, &stIn, &stOut);
if (0 == lPlayHandle)
{
    printf("CLIENT_PlayBackByTimeEx2: failed! Error code: %x.\n", CLIENT_GetLastError());
}
printf("input any key to quit!\n");
getchar();

// Stop playback
if (0 != lPlayHandle)
{
    if (FALSE == CLIENT_StopPlayBack(lPlayHandle))
    {
        printf("CLIENT_StopPlayBack Failed, lRealHandle[%x]!Last Error[%x]\n" , lPlayHandle,
            CLIENT_GetLastError());
    }
    else
    {
        lPlayHandle = 0;
    }
```

2.5.4.2 Payback Pause + Resume Playback (Optional)

```
// Pause playback
BOOL bSuccess = CLIENT_PausePlayBack(m_hPlayBack,TRUE);
if (!bSuccess)
{
    printf("CLIENT_PausePlayBack Failed, lPlayHandle[%x]!Last Error[%x]\n" , lPlayHandle,
CLIENT_GetLastError());
}
// Resume playback
    bSuccess = CLIENT_PausePlayBack(m_hPlayBack, FALSE);
if (!bSuccess)
{
    printf("CLIENT_PausePlayBack Failed, lPlayHandle[%x]!Last Error[%x]\n" , lPlayHandle,
CLIENT_GetLastError());
}
```

2.6 Record Download

2.6.1 Introduction

Video surveillance system widely applies to safe 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.

2.6.2 Interface Overview

Interface	Implication
CLIENT_SetDeviceMode	Set the work modes such as voice talk, playback, and authority.
CLIENT_QueryRecordFile	Query all the record files within a period.
CLIENT_FindFile	Open the record query handle.
CLIENT_FindNextFile	Find the record file.
CLIENT_FindClose	Close the record query handle.
CLIENT_DownloadByRecordFileEx	Download the record by file.
CLIENT_DownloadByTimeEx	Download the record by time.
CLIENT_GetDownloadPos	Get the record download progress.
CLIENT_StopDownload	Stop the record download.

Table 2-7

2.6.3 Process

The record download is consisted of download by file and download by time.

2.6.3.1 Download by File

You need to import the record file information to be downloaded. SDK can download the specified record file and save to the required place.

You can also provide a callback pointer to SDK that calls back the specified record file to you for treatment.

For the process of download by file, see Figure 2-8.

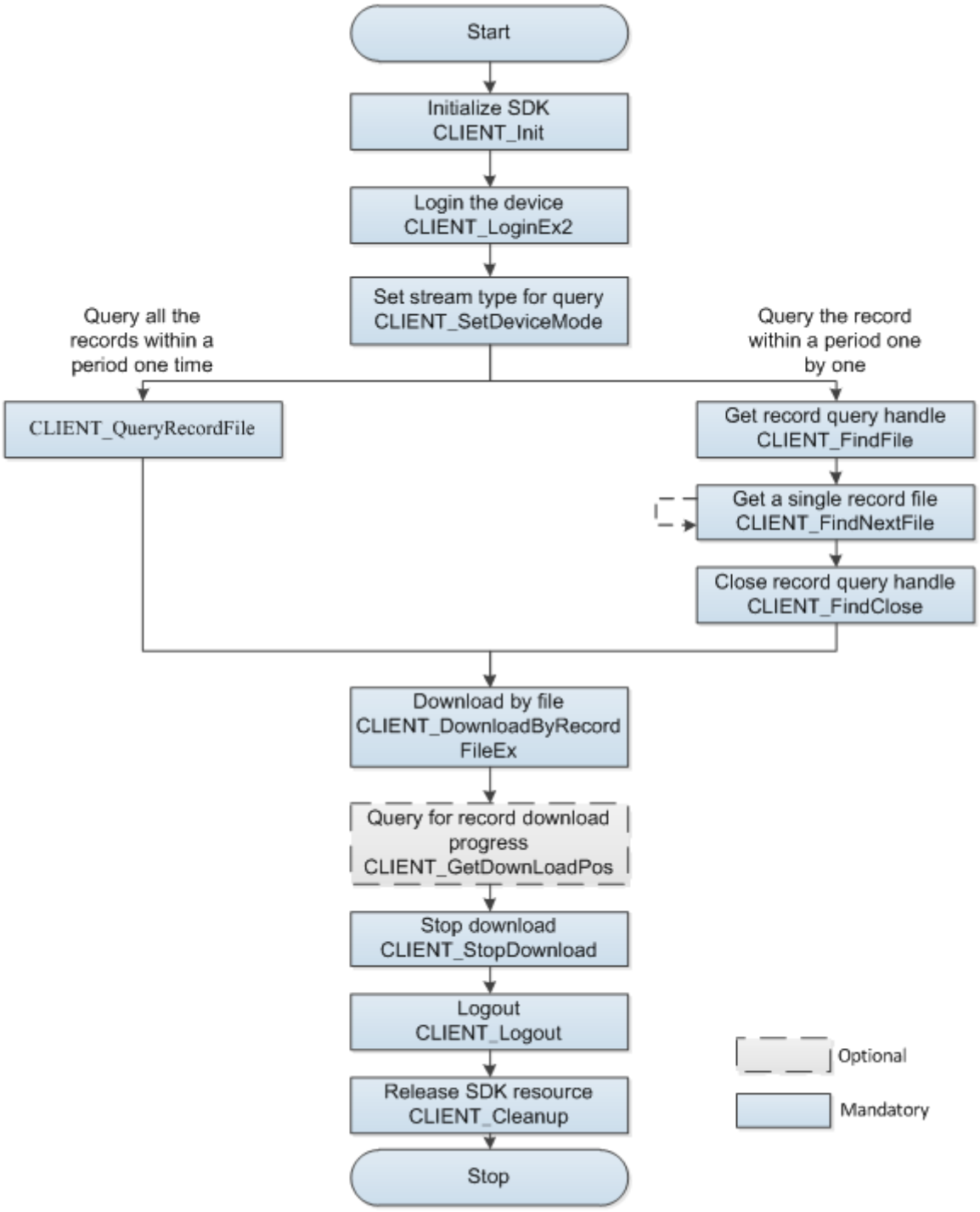

Figure 2-8

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Call **CLIENT_SetDeviceMode** to set the parameter emType as DH_RECORD_STREAM_TYPE. It is suggested to set 0 as main stream; otherwise the result cannot be obtained from some devices. If you only need main stream record, you can filter the sub stream from the result.

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 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. Either sSavedFileName or fDownLoadDataCallBack is valid at least. You can decide whether to use cbDownLoadPos, if not, set it as NULL.

Step 6 (Optional) During downloading, call **CLIENT_GetDownloadPos** to query the record downloading progress or use cbDownLoadPos mentioned in step 5 to obtain the real-time download progress.

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

Step 8 After using the function module, call **CLIENT_Logout** to logout the device.

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

2.6.3.2 Download by Time

You can import the start time and end time of download. SDK can download the specified record file and save to the required place.

You can also provide a callback pointer to SDK which calls back the specified record file to you for treatment.

For the process of download by time, see Figure 2-9.

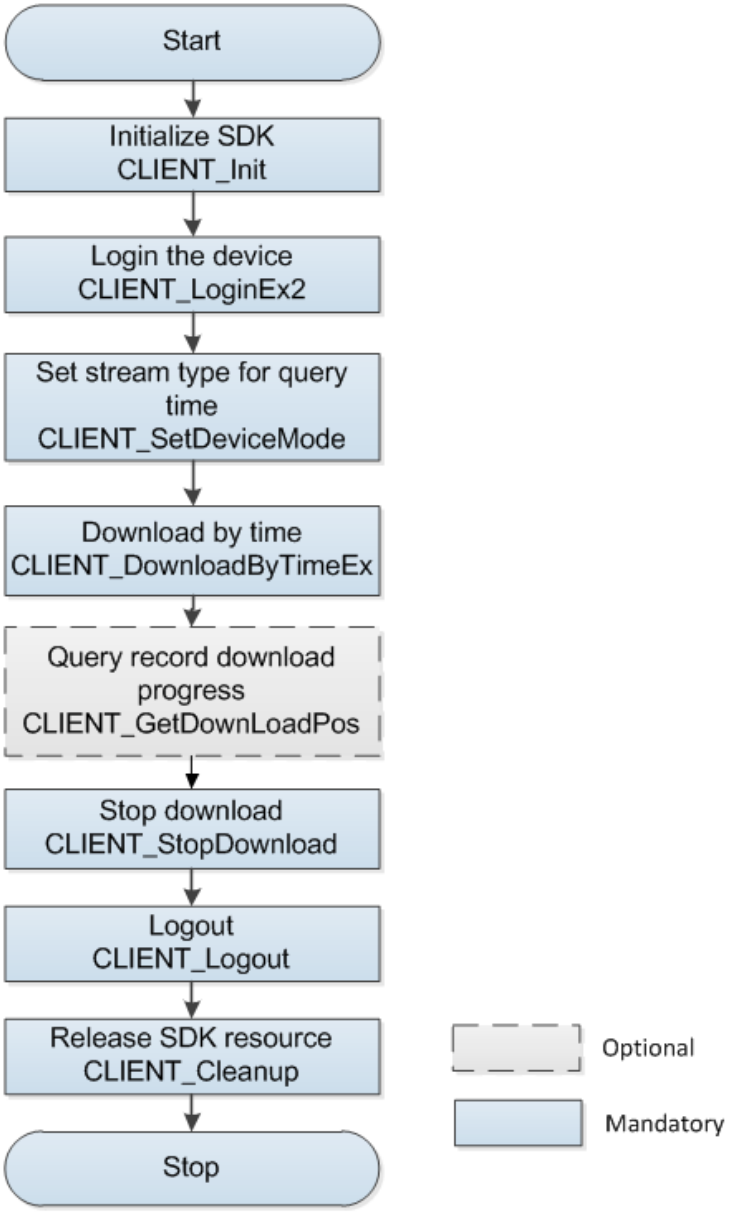

Figure 2-9

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Call **CLIENT_SetDeviceMode** to set the parameter emType as DH_RECORD_STREAM_TYPE.

Step 4 Call **CLIENT_DownloadByTimeEx** to start downloading by time. Either sSavedFileName or fDownLoadDataCallBack is valid. You can decide whether to use cbDownLoadPos, if not, set it as NULL**.**

Step 5 (Optional) During downloading, you can choose to call **CLIENT_GetDownloadPos** to query the record downloading progress or use cbDownLoadPos mentioned in step 5 to obtain the real-time download progress.

Step 6 Call **CLIENT_StopDownload** to stop download**.** You can close the download process after it is completed or it is just partially completed.

Step 7 After using the function module, call **CLIENT_Logout** to logout the device.

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

2.6.4 Example Code

2.6.4.1 Download by File

```
// Callback declaration
// Playback/download progress callback
// It is not recommended to call SDK interface in this callback
// dwDownLoadSize: "-1" represents current playback/download has completed. "-2" represents writing file failed. Other values represent valid data.
// Set this callback through CLIENT_DownloadByRecordFileEx. When SDK receives playback/download data, it will call this function.
void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);

// Playback/download data callback
// It is not recommended to call SDK interface in this callback
// During playback, the parameters return. "0" represents the failure of this calling, and the next calling will return the same data. "1" represents the success of this calling, and the next calling will returns the following data.
// During downloading, no matter which value is returned, the calling is successful, and the next calling will return the following data.
// Set the callback through CLIENT_DownloadByRecordFileEx. When SDK receives the playback/download data, it will call this function.
int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);

// Record file query
    // Set the record stream type for query as main stream.
    int nStreamType = 0; // 0-main stream,1-main stream,2-sub stream
    CLIENT_SetDeviceMode(lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);

    // There are two ways of record query:1. Get all the record files of a certain period one time; 2. Get all the record files of a certain period by several times.
    // Way 1: Get all the record files of a certain period one time
    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};
    NET_RECORDFILE_INFO stuNetFileInfo;
    int nFileCount = 0;
    //Record file query
    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());
    }
    else
    {
        // Set the first file queried as the download file
        memcpy(&stuNetFileInfo, (void *)&netFileInfo[0], sizeof(stuNetFileInfo));
    }

    // Way 2: Get all the record files of a certain period by several times
    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(lLoginHandle, nChannelID, 0, NULL, &stuStartTime, &stuStopTime, FALSE, 5000);
    if (0 == lFindHandle)
    {
        printf("CLIENT_FindFile Failed!Last Error[%x]\n",CLIENT_GetLastError());
        return;
    }
    // The demo code takes the record file nMaxRecordFileCount as an example.
    std::vector<NET_RECORDFILE_INFO> bufFileInfo(nMaxRecordFileCount);
```

```
    for (int nFileIndex = 0; nFileIndex < nMaxRecordFileCount; ++nFileIndex)
    {
        int result = CLIENT_FindNextFile(lFindHandle, &bufFileInfo[nFileIndex]);
        if (0 == result)// Getting the record file information finished
        {
            break;
        }
        else if (1 != result)// Parameter error
        {
            printf("CLIENT_FindNextFile Failed!Last Error[%x]\n",CLIENT_GetLastError());
            break;
        }
    }

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

// Record file download
    // Enable record download
    // Either sSavedFileName or fDownLoadDataCallBack is valid at least
    // In the application, save to sSavedFileName or callback to handle the data.
lDownloadHandle = CLIENT_DownloadByRecordFileEx(lLoginHandle, &stuNetFileInfo, "test.dav", DownLoadPosCallBack, NULL, DataCallBack, NULL);
    if (0 == lDownloadHandle)
    {
```

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

// Close downloading can be called after download is finished or in process.
        if (0 != lDownloadHandle)
        {
                if (FALSE == CLIENT_StopDownload(lDownloadHandle))
                {
                    printf("CLIENT_StopDownload Failed, lDownloadHandle[%x]!Last Error[%x]\n" ,
lDownloadHandle, CLIENT_GetLastError());
                }
                else
                {
                     lDownloadHandle = 0;
                }
        }

// Callback definition
void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, LDWORD dwUser)
{
        // If several playback/download use the same progress callback, the user can make one-one correspondence
  through lPlayHandle.
        if (lPlayHandle == 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 several playback/download use the same data callback, you can make one-one correspondence through
```

```
lRealHandle.
    if(lRealHandle == 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
            // You can save the stream data here and decode or forward after leaving the callback.
            nRet = 1;

            break;
        case 1:
            //Standard video data

            break;
        case 2:
            //yuv data

            break;
        case 3:
            //pcm audio data

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

2.6.4.2 Download by Time

```
// Callback declaration
// Playback progress callback by time
```

```
// It is not recommended to call SDK interface in this callback
// dwDownLoadSize: "-1" represents current playback/download has completed. "-2" represents writing file failed.
   Other values represent valid data.

// Set this callback through CLIENT_DownloadByTimeEx. When SDK receives playback/download data, it 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
// It is not recommended to call SDK interface in this callback
// During playback, the parameters return. "0" represents the failure of this calling, and the next calling will return
the same data. "1" represents the success of this calling, and the next calling will returns the following data.
// During downloading, no matter which value is returned, the calling is successful, and the next calling will return
the following data.
// Set the callback through CLIENT_DownloadByTimeEx. When SDK receives the playback/download data, it
will call this function.
int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LDWORD dwUser);

// Set the record stream type for query as main stream.
      int nStreamType = 0; // 0-main and sub stream,1-main stream,2-sub stream
      CLIENT_SetDeviceMode(lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);
      // Set download start time and end time
      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;

// Start record download
      // The formal parameter either sSavedFileName or fDownLoadDataCallBack should be valid, otherwise there
   will be parameter input error
lDownloadHandle = CLIENT_DownloadByTimeEx(lLoginHandle, nChannelID, EM_RECORD_TYPE_ALL,
&stuStartTime, &stuStopTime, "test.dav", TimeDownLoadPosCallBack, NULL, DataCallBack, NULL);
      if (lDownloadHandle == 0)
```

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

// Close downloading can be called after download is finished or in process.
    if (0 != lDownloadHandle)
    {
        if (FALSE == CLIENT_StopDownload(lDownloadHandle))
        {
printf("CLIENT_StopDownload Failed, lDownloadHandle[%x]!Last Error[%x]\n" , lDownloadHandle,
CLIENT_GetLastError());
        }
        else
        {
            lDownloadHandle = 0;
        }
    }

// Callback definition
void CALLBACK TimeDownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser)
{
    // If several playback/download use the same progress callback, you can make one-one correspondence
through lPlayHandle.
    if (lPlayHandle == 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 several playback/download use the same progress callback, you can make one-one correspondence through lPlayHandle.
    if(lRealHandle == 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
            // You can save the stream data here and decode or forward after leaving the callback.
            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;
```

```
}
```

2.7 PTZ Control

2.7.1 Introduction

PTZ is a mechanical platform that carries the device and the protective enclosure and performs remote control in all directions.

PTZ is consisted of two motors that can perform horizontal and vertical movement to provide the all-around vision.

This section provides guidance to you about how to control directions (there are eight directions: upper, lower, left, right, upper left, upper right, bottom left, and bottom right), focus, zoom, iris, fast positioning, and 3-dimensional positioning through SDK.

2.7.2 Interface Overview

Interface	Implication
CLIENT_DHPTZControlEx2	PTZ control extension interface.

Table 2-8

2.7.3 Process

Direction control, focus, zoom, and aperture are the continuous operations. SDK provides start and stop interfaces to you for timing control.

For the process of PTZ control, see Figure 2-10.

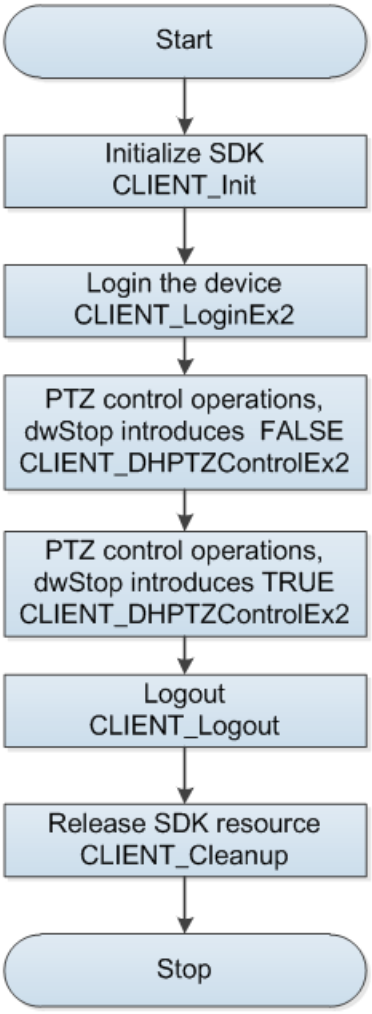

Figure 2-10

Both fast positioning and 3-dimensional positioning belong to one-time action, which needs to call the PTZ control interface just one time.

For the process of one-time operation of PTZ control, see Figure 2-11.

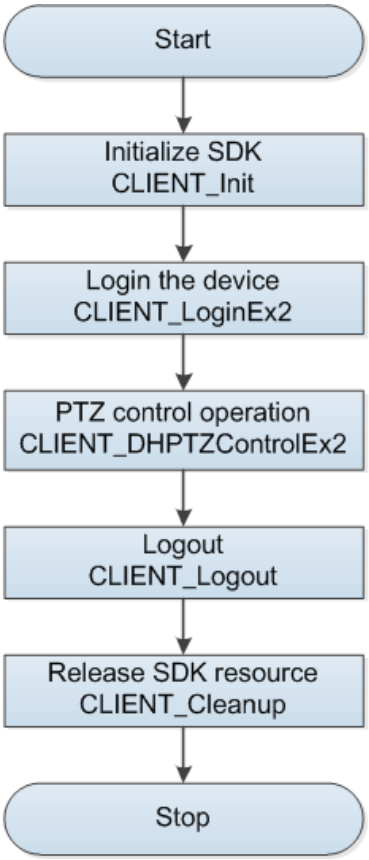

Figure 2-11

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Call **CLIENT_DHPTZControlEx2** to operate the PTZ according to the situation. Different PTZ command might need different parameters, and part of commands need to call the corresponding stop command, such as moving left and moving right. For details, see "2.7.4 Example Code."

Step 4 After using the function module, call **CLIENT_Logout** to logout the device.

Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Fast positioning: For the SD, take the current monitoring image center as origin, and the valid range of horizontal and vertical coordinates is [-8191, 8191]. For example, if the horizontal coordinate is 2000 and the vertical is 2000, the SD moves toward upper right and gets a new origin, which means the coordinate specified every time is only relative to the current location.
- 3-dimensional positioning: For the SD, there is an initial position first. The horizontal coordinate is [0, 3600] and the vertical is [-1800, 1800]. The coordinate specified each time is the absolute coordinate and is irrelevant to the location of the SD image last time.
- For more example code see the SDK package on the website.

2.7.4 Example Code

```
LONG lParam1 = 0; // Rotating speed in horizontal direction.
LONG lParam2 = 4; // Rotating speed in vertical direction.
LONG lParam3 = 0;
// Continuous operation: take moving upward as example.
// Start moving upward.
BOOL bRet = CLIENT_DHPTZControlEx2(lLoginHandle, nChannelId, DH_PTZ_UP_CONTROL, lParam1,
                                   lParam2, lParam3, FALSE, NULL);
// Stop moving forward.
bRet = CLIENT_DHPTZControlEx2(lLoginHandle, nChannelId, DH_PTZ_UP_CONTROL, lParam1, lParam2,
                                   lParam3, TRUE, NULL);

// One-time operation movement: take fast positioning as example.
lParam1 = 2000;    // Horizontal coordinate, valid range[-8191,8191]
lParam2 = 2000;    // Vertical coordinate, valid range [-8191,8191]
lParam3 = 1;       // Zoom, valid range (-16 ~ 16),1 indicates rotating without zooming
bRet = CLIENT_DHPTZControlEx2(lLoginHandle, nChannelId, DH_EXTPTZ_FASTGOTO, lParam1, lParam2,
                                   lParam3, FALSE, NULL);
```

2.8 Voice Talk

2.8.1 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.

2.8.2 Interface Overview

Interface	Implication
CLIENT_StartTalkEx	Start voice talk.
CLIENT_StopTalkEx	Stop voice talk.
CLIENT_RecordStartEx	Start client record (valid only in Windows system).
CLIENT_RecordStopEx	Stop client record (valid only in Windows system).
CLIENT_TalkSendData	Send voice data to the device.
CLIENT_AudioDecEx	Decode audio data (valid only in Windows system).

Table 2-9

2.8.3 Process

When SDK has collected the audio data from the local audio card, or SDK has received the audio data from the front-end devices, SDK will call the callback of audio data.

You can call the SDK interface in the callback parameters to send the local audio data to the front-end devices, or call SDK interface to decode and playback the audio data received from the front-end devices.

This process is valid only in Windows system. For the process of voice talk, see Figure 2-12.

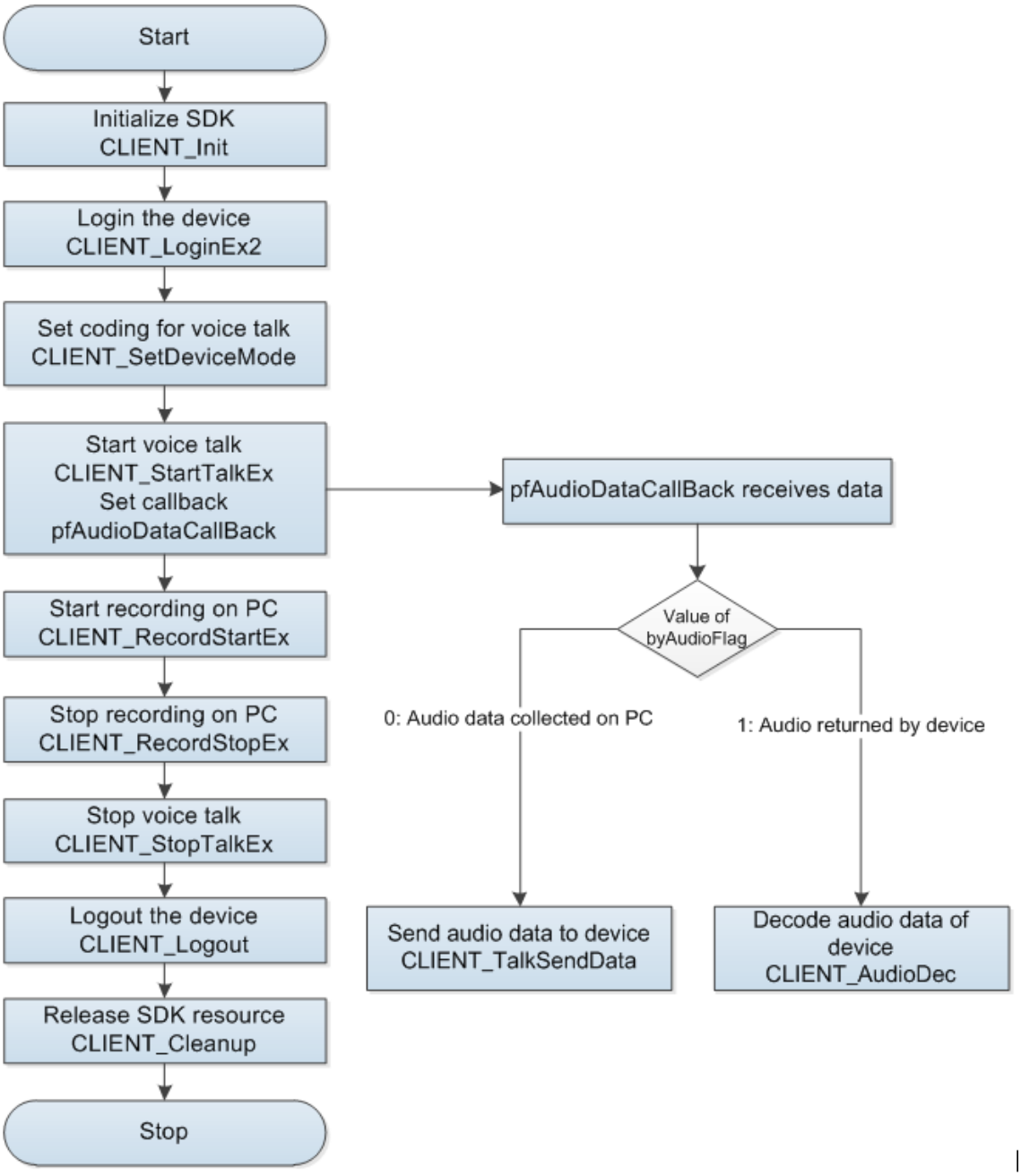

Figure 2-12

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Call **CLIENT_SetDeviceMode** to set decoding information of voice talk. Set parameter emType as DH_TALK_ENCODE_TYPE.

Step 4 Call **CLIENT_StartTalkEx** to set callback and start voice talk. In the callback, call **CLIENT_AudioDec** to decode the audio data sent from the decoding device, and call **CLIENT_TalkSendData** to send the audio data of the PC end to the device.

Step 5 Call **CLIENT_RecordStartEx** to start recording at PC. After this interface is called, the voice talk callback in CLIENT_StartTalkEx will receive the local audio data.

Step 6 After using the voice talk function, call **CLIENT_RecordStopEx** to stop recording.

Step 7 Call **CLIENT_StopTalkEx** to stop voice talk.

Step 8 After using the function module, call **CLIENT_Logout** to logout the device.

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

Notes from Process

- Voice encoding format: The example uses the common PCM format. SDK supports accessing the voice encoding format supported by the device. For more details of the example code, see the SDK package on the website. If the default PCM can satisfy the requirement, it is not recommended to obtain the voice encoding format from the device.
- No sound at the device: The audio data needs to be collected by the device such as microphone. It is recommended to check if the microphone or other equivalent device is plugged in and if the CLIENT_RecordStartEx succeeded in returning.

2.8.4 Example Code

```
// Set the voice talk encoding data, take PCM as an example.
DHDEV_TALKDECODE_INFO curTalkMode;
curTalkMode.encodeType = DH_TALK_PCM;
curTalkMode.nAudioBit = 16;
curTalkMode.dwSampleRate = 8000;
curTalkMode.nPacketPeriod = 25;
CLIENT_SetDeviceMode(lLoginHandle, DH_TALK_ENCODE_TYPE, &curTalkMode);
// Start voice talk
lTalkHandle = CLIENT_StartTalkEx(lLoginHandle, AudioDataCallBack, (LDWORD)NULL);
if(0 != lTalkHandle)
{
    BOOL bSuccess = CLIENT_RecordStartEx(lLoginHandle);
}

// Stop local recording
if (!CLIENT_RecordStopEx(lLoginHandle))
{
    printf("CLIENT_RecordStop Failed!Last Error[%x]\n", CLIENT_GetLastError());
}
```

```
// Stop voice talk
if (0 != lTalkHandle)
{
    CLIENT_StopTalkEx(lTalkHandle);
}
void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag, LDWORD dwUser)
{
    if(0 == byAudioFlag)
    {
        // Send the sound data checked by the PC to the device
        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 the voice data of the device to SDK encode and play.
        CLIENT_AudioDec(pDataBuf, dwBufSize);
    }
}
```

2.9 Video Snapshot

2.9.1 Introduction

Video snapshot obtains the picture data of the current video. This section introduces the following two snapshot ways:

- Network snapshot: Call the SDK interface to send the capturing command to the device that captures the current image and send to SDK through network, and then SDK returns the image data to you.
- Local snapshot: When the monitoring is opened, you can save the monitoring data to the picture format that is the frame information that does not have an interaction with the device.

2.9.2 Interface Overview

Interface	Implication

Interface	Implication
CLIENT_SnapPictureToFile	Snapshot and directly returns the picture data to the user.
CLIENT_CapturePictureEx	Local snapshot with the parameters that could be monitoring handle or playback handle.

Table 2-10

2.9.3 Process

Video snapshot is consisted of network snapshot and local snapshot.

2.9.3.1 Network Snapshot

For the process of network snapshot, see Figure 2-13.

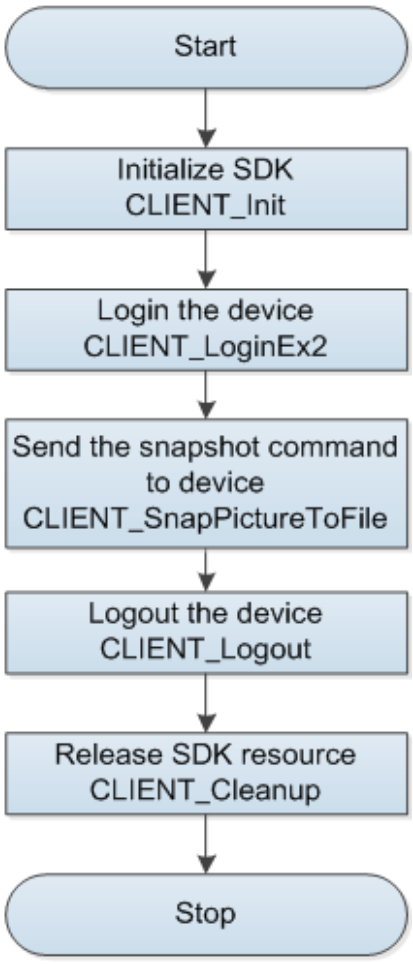

Figure 2-13

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Call **CLIENT_SnapPictureToFile** to obtain the picture data.

Step 4 After using the function module, call **CLIENT_Logout** to logout the device.

Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

Notes for Process

- Picture size limit: SDK allocates the fixed memory (2 M) to receive the picture data returned from the device. If the picture is too big to put into the fixed memory, SDK will return the truncated data.
- SDK provides the interface to modify the default memory. If the picture (for example, the high definition picture) is truncated, you can modify nPicBufSize bigger. The example code is as follows. After calling CLIENT_Init, call the example code just once is sufficient.

```
NET_PARAM stuNetParam = {0};
stuNetParam. nPicBufSize = 4*1024*1024; // unit byte
CLIENT_SetNetworkParam (&stuNetParam);
```

- Multi-thread calling: Multi-thread calling is not supported for the functions within the same login session.
- Snapshot configuration: You can configure the items such as quality and definition for the snapshot. However, it is not recommended to modify if the default configurations are satisfactory. For more details of the example code, see the SDK package on the website.
- Picture save: the picture data is returned as memory and the interface supports saving the picture data as file (the precondition is that you have set the szFilePath field of NET_IN_SNAP_PIC_TO_FILE_PARAM).

2.9.3.2 Local Snapshot

For the process of local snapshot, see Figure 2-14.

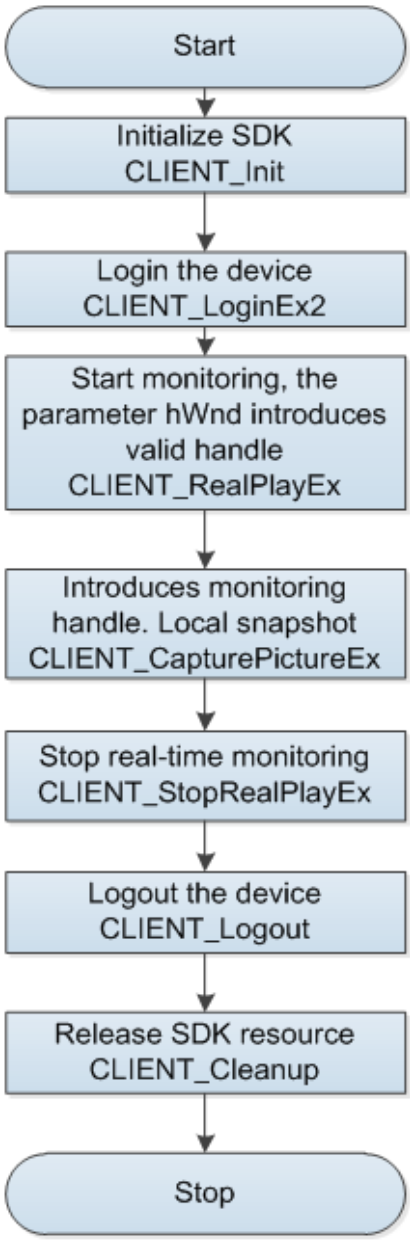

Figure 2-14

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 Call **CLIENT_RealPlayEx** to open monitoring and obtain the monitoring handle.
Step 4 Call **CLIENT_CapturePictureEx** to introduce the monitoring handle.
Step 5 Call **CLIENT_StopRealPlayEx** to close the real-time monitoring.
Step 6 After using the function module, call **CLIENT_Logout** to logout the device.
Step 7 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.9.4 Example Code

```
// Network capturing example
NET_IN_SNAP_PIC_TO_FILE_PARAM stuInParam = {sizeof(stuInParam)};
```

```
NET_OUT_SNAP_PIC_TO_FILE_PARAM stuOutParam = {sizeof(stuOutParam)};
SNAP_PARAMS stuSnapParams = {0};
stuSnapParams.Channel = 0;      // Take the first channel as an example
int nBufferLen = 2*1024*1024;
char* pBuffer = new char[nBufferLen]; // Picture cache
memset(pBuffer, 0, nBufferLen);
stuOutParam.szPicBuf = pBuffer;
stuOutParam.dwPicBufLen = nBufferLen;
if (FALSE == CLIENT_SnapPictureToFile(lLoginHandle, &stuSnapParams))
{
      printf("CLIENT_SnapPictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
}
delete[] pBuffer;

// Example of local capturing. hPlayHandle is the handle for opening monitoring.
if (FALSE == CLIENT_CapturePictureEx(hPlayHandle, “test.jpg”, NET_CAPTURE_JPEG))
{
      printf("CLIENT_CapturePictureEx Failed!Last Error[%x]\n", CLIENT_GetLastError());
}
```

2.10 Alarm Upload

2.10.1 Introduction

Alarm upload can be realized through SDK login the device and subscription of the alarm function to the device which will send the detected alarm event to SDK. The alarm information can be obtained through callback.

2.10.2 Interface Overview

Interface	Implication
CLIENT_SetDVRMessCallBack	Set the alarm callback.
CLIENT_StartListenEx	Subscribe alarm.
CLIENT_StopListen	Stop subscribing alarm.

Table 2-11

2.10.3 Process

For the process of alarm upload, see Figure 2-15.

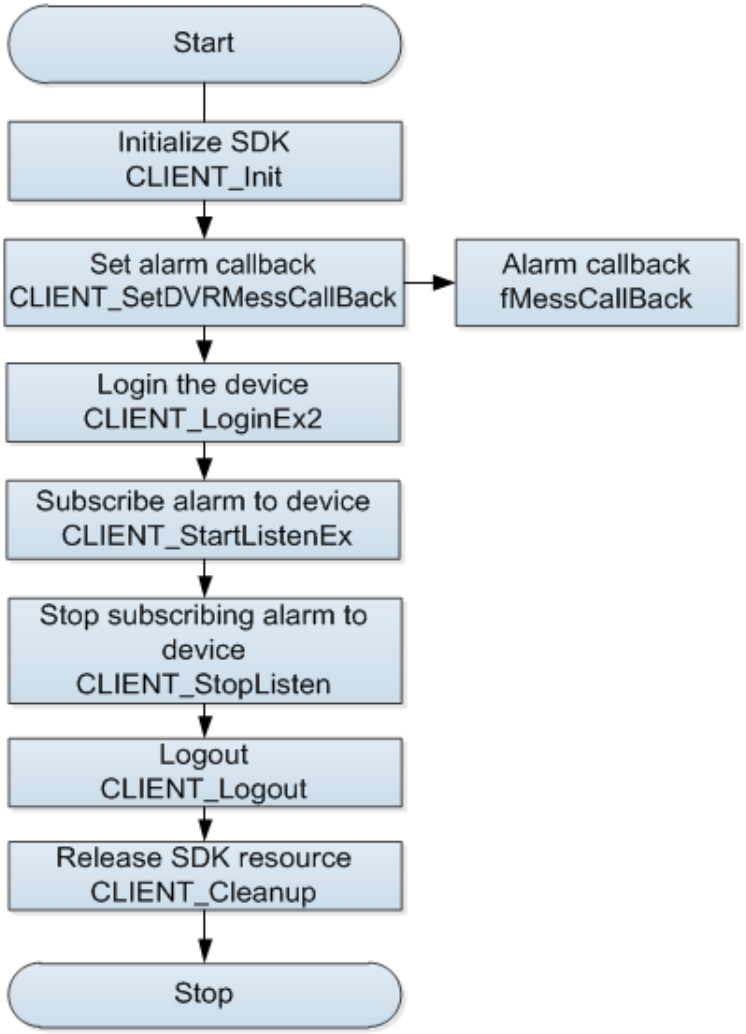

Figure 2-15

Process Description

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

Step 2 Call **CLIENT_SetDVRMessCallBack** to set alarm callback which should be called before subscribing alarm.

Step 3 Call **CLIENT_LoginEx2** to login the device.

Step 4 Call **CLIENT_StartListenEx** to subscribe alarm to the device. If succeeded, the alarm event uploaded by the device will be informed to you through the callback set by CLIENT_SetDVRMessCallBack.

Step 5 After using the alarm upload function, call **CLIENT_StopListen** to stop subscribing alarm to the device.

Step 6 After using the function module, call **CLIENT_Logout** to logout the device.

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

Notes for Process

If the previously uploaded alarm event is no longer uploaded, check if the device is disconnected. Because the device will not upload the alarm after auto reconnection, you need to cancel subscription and subscribe again.

2.10.4 Example Code

```
// Alarm callback
int CALLBACK    afMessCallBack(LONG lCommand, LLONG lLinID, char *pBuf, DWORD dwBufLen,
char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
      if(lCommand == DH_ALARM_STORAGE_FAILURE_EX)//Alarm for hard disk drive failure
      {
            printf("Alarm for hard disk failure ");
            ALARM_STORAGE_FAILURE_EX*                    pstStorageFailureInfo                    =
(ALARM_STORAGE_FAILURE_EX*)pBuf;
            //Next the user can use pstStorageFailureInfo to obtain the corresponding alarm information
      }
      if(lCommand == DH_DISKERROR_ALARM_EX) // Alarm for damaged hard disk drive
            {
                  printf("Alarm for damaged hard disk drive \n");
            // Alarm of video loss is not accompanied with any alarm information
            }
}

// Set alarm callback
CLIENT_SetDVRMessCallBack(&afMessCallBack,0);
// Subscribe alarm
BOOL bRet = CLIENT_StartListenEx(lLoginHandle);
if(!bRet)
{
      printf("CLIENT_StartListenEx Failed, Last Error[%x]\n" ,
      CLIENT_GetLastError());
}
else
{
        printf("listen succeed.\n");
}

// Stop alarm subscription
BOOL bRet = CLIENT_StopListen(lLoginHandle);
if(!bRet)
{
      printf("CLIENT_StopListen Failed, Last Error[%x]\n" ,
      CLIENT_GetLastError());
```

```
}
else
{
    printf("stop listen succeed.\n");
}
```

2.11 Storage

2.11.1 Introduction

The storage interface mainly includes access to remote device information, query subscription on connection state of remote device, and modification of remote channel name.

2.11.2 Interface Overview

Interface	Implication
CLIENT_QueryDevState	Directly get the connection state of remote device for each channel individually. The parameter type is DH_DEVSTATE_VIRTUALCAMERA.
CLIENT_QueryDevInfo	Directly get the connection state of remote device for all the channels at the same time. The parameter nQueryType is NET_QUERY_GET_CAMERA_STATE.
CLIENT_AttachCameraState	Subscribe the remote device state. When the state changes, the corresponding information will be reported.
CLIENT_DetachCameraState	Stop subscribing the remote device state. Used with CLIENT_AttachCameraState in match.
CLIENT_MatrixGetCameras	Get the information of remote device, such as device type and IP.
CLIENT_QueryChannelName	Get the channel name.
CLIENT_GetNewDevConfig	Get the channel name. The parameter szCommand is CFG_CMD_VIDEOIN or CFG_CMD_CHANNELTITLE.
CLIENT_ParseData	Analyze the data. Used with CLIENT_GetNewDevConfig in match.

Table 2-12

2.11.3 Process

The storage module has the following processes:

- Direct access to connection state of remote device

- Subscription to connection state of remote device
- Access to the information of remote device
- Access to channel name of remote device

2.11.3.1 Direct access to connection state of remote device

For the process of direct access to connection state of remote device, see Figure 2-16.

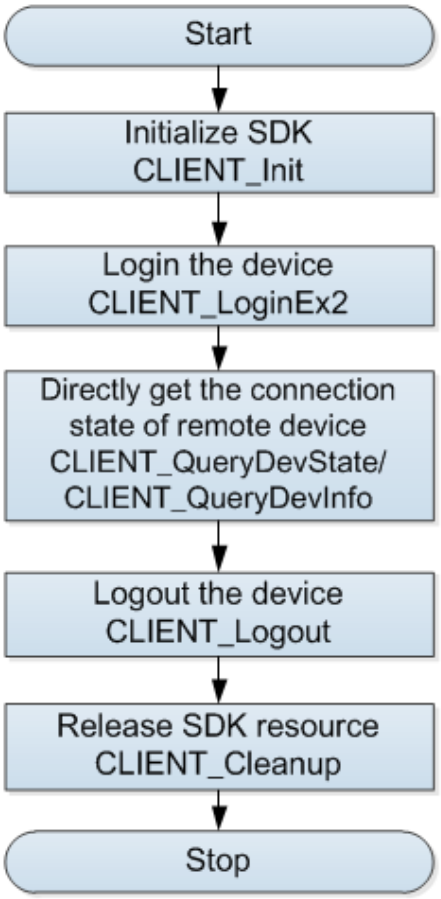

Figure 2-16

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Directly access to connection state of remote device.

There are two interfaces that are CLIENT_QueryDevState (type is DH_DEVSTATE_VIRTUALCAMERA) and CLIENT_QueryDevInfo (QueryType is NET_QUERY_GET_CAMERA_STATE) respectively.

NOTE

The two interfaces are different depending on the device. It is suggested to have a test prior to use, and then select the proper interface.

Step 4 After using the function module, call **CLIENT_Logout** to logout the device.

Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.11.3.2 Subscription to Connection State of Remote Device

For the process of subscription to connection state of remote device, see Figure 2-17.

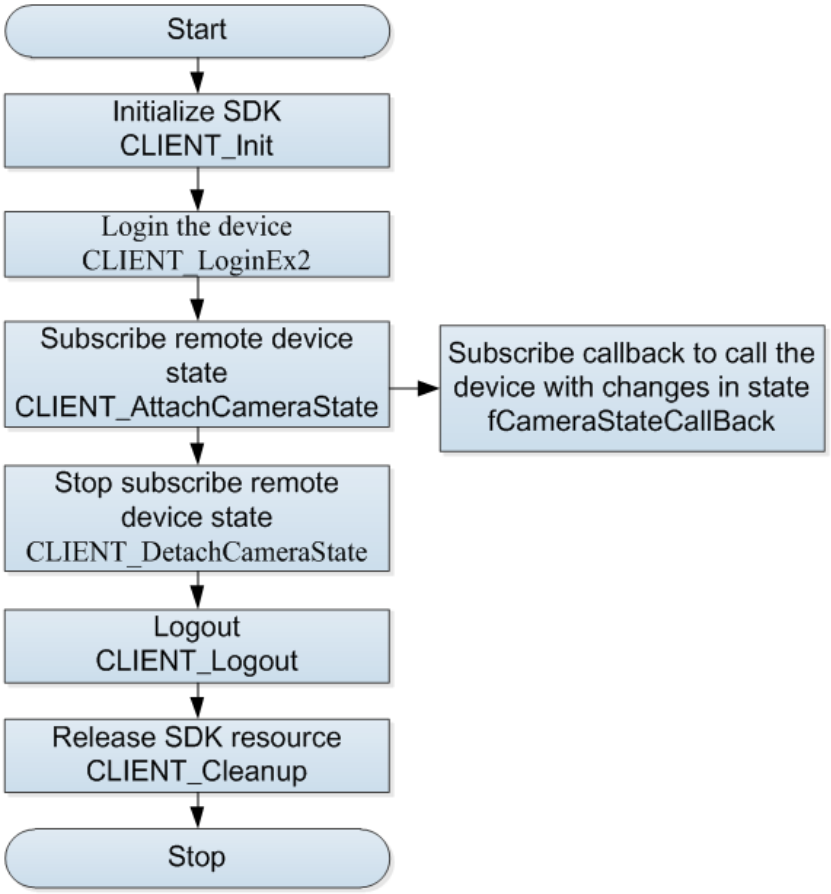

Figure 2-17

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Call **CLIENT_AttachCameraState** to subscribe the connection state of remote device. If the state changes, the report will be sent through fCameraStateCallBack.

Step 4 Call **CLIENT_DetachCameraState** to cancel the subscription. There will be no upload even the remote device state is changed.

Step 5 After using the function module, call **CLIENT_Logout** to logout the device.

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

2.11.3.3 Access to The Information of Remote Device

For the process of access to the information of remote device, see Figure 2-18.

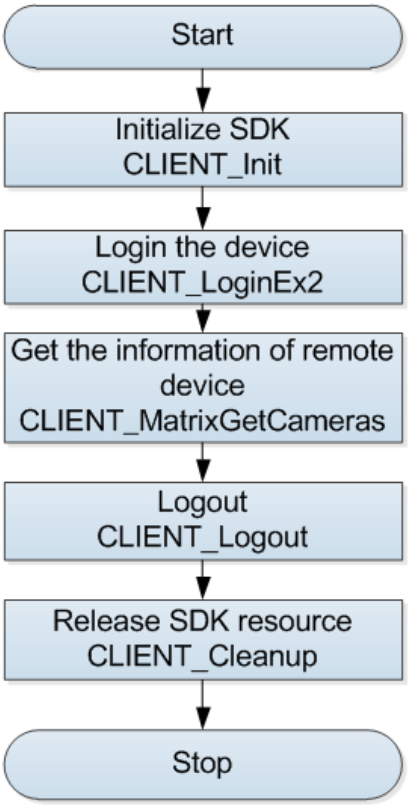

Figure 2-18

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Call **CLIENT_MatrixGetCameras** to get the remote device information such as device type, IP.

Step 4 After using the function module, call **CLIENT_Logout** to logout the device.

Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.11.3.4 Access to channel name of remote device

For the process of access to channel name of remote device, see Figure 2-19

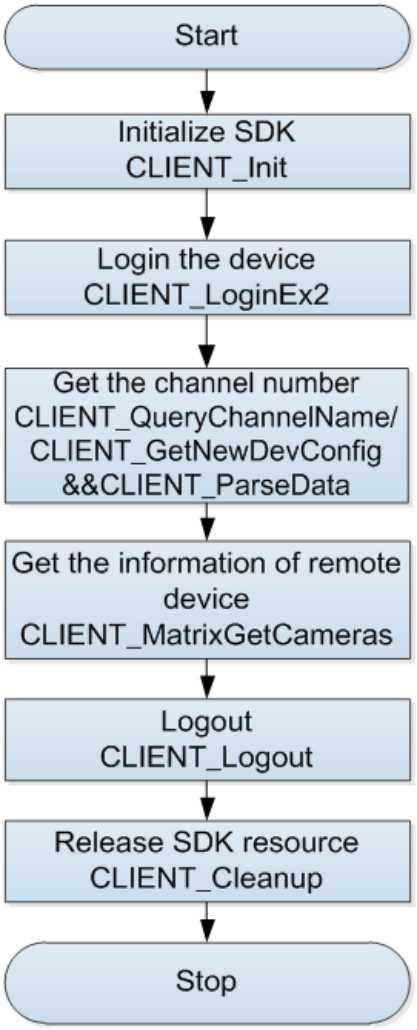

Figure 2-19

Process Description

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

Step 2 Call **CLIENT_LoginEx2** to login the device.

Step 3 Get the channel name through one of the following two interfaces:

- CLIENT_QueryChannelName
- CLIENT_GetNewDevConfig&&CLIENT_ParseData

The second interface has two commands to get the channel name CFG_CMD_VIDEOIN and CFG_CMD_CHANNELTITLE.

NOTE

The function of the two interfaces might be different dependent on the device. It is recommended to test prior to use to decide which one to use.

Step 4 After using the function module, call **CLIENT_Logout** to logout the device.

Step 5 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resource.

2.11.4 Example Code

2.11.4.1 Direct Access to Connection State of Remote Device

```
// Interface 1 for getting the connection state of remote device
    DHDEV_VIRTUALCAMERA_STATE_INFO stInOut =
    {sizeof(stInOut)};
    int nRetLen = 0;
    for(int i=0;i< nChnCount;i++)
    {
        stInOut.nChannelID = i;//Get one by one per channel
        BOOL bRet = CLIENT_QueryDevState(lLoginHandle,DH_DEVSTATE_VIRTUALCAMERA,
        (char *)&stInOut,sizeof(stInOut),&nRetLen,3000);
        if(bRet)
        {
            if (stInOut.emConnectState == 0)
            {
                printf("Channel%d state is disconnected\n",i);
            }
            else if (stInOut.emConnectState == 1)
            {
                printf("Channel%d state is connecting\n",i);
            }
            else if(stInOut.emConnectState == 2)
            {
                printf("Channel%d state is connected\n",i);
            }
            memset((void *)&stInOut,0,sizeof(stInOut));
            stInOut.nStructSize = sizeof(stInOut);
        }
        else
        {
            printf("get virtual camera state failed.");
        }
    }
    else
    {
        printf("CLIENT_QueryDevState Failed, Last Error[%x]\n",
            CLIENT_GetLastError());
```

```
    }

// Interface 2 for getting the connection state of remote device
    NET_IN_GET_CAMERA_STATEINFO stIn = {sizeof(stIn)};
    stIn.bGetAllFlag = TRUE;
    NET_OUT_GET_CAMERA_STATEINFO stOut = {sizeof(stOut)};
    stOut.pCameraStateInfo = new NET_CAMERA_STATE_INFO[64];
    memset(stOut.pCameraStateInfo,0,sizeof(*stOut.pCameraStateInfo)*64);
    stOut.nMaxNum = 64;
    BOOL bRet = CLIENT_QueryDevInfo(lLoginHandle,NET_QUERY_GET_CAMERA_STATE,
    &stIn,&stOut,NULL,3000);
    if(bRet)
    {
        for(int i = 0;i<stOut.nValidNum;i++)
        {
            if(stOut.pCameraStateInfo[i].emConnectionState == 0)
            {
                cout << "Current channel"<<stOut.pCameraStateInfo[i].nChannel<<
                  "State:"<<"Unknown"<<endl;
            }
            else if (stOut.pCameraStateInfo[i].emConnectionState == 1)
            {
                cout << "Current channel"<<stOut.pCameraStateInfo[i].nChannel<<
                  " State:"<<"Connecting"<<endl;
            }
            else if(stOut.pCameraStateInfo[i].emConnectionState == 2)
            {
                cout << "Current channel"<<stOut.pCameraStateInfo[i].nChannel<<
                  " State:"<<"Connected"<<endl;
            }
            else if(stOut.pCameraStateInfo[i].emConnectionState == 3)
            {
                cout << "Current channel"<<stOut.pCameraStateInfo[i].nChannel<<
                  " State:"<<"Disconnected"<<endl;
            }
            else if (stOut.pCameraStateInfo[i].emConnectionState == 4)
            {
                cout << "Current channel"<<stOut.pCameraStateInfo[i].nChannel<<
                  "State:"<<"Channel not configured, and no information"<<endl;
            }
```

```
            else if (stOut.pCameraStateInfo[i].emConnectionState == 5)
            {
                cout << "Current channel"<<stOut.pCameraStateInfo[i].nChannel<<"State:"<<
                  "Channel configured but prohibited"<<endl;
            }
        }
            cout<<"succeed"<<endl;
    }
    else
    {
        printf("CLIENT_QueryDevInfo Failed, Last Error[%x]\n",
            CLIENT_GetLastError());

    }
```

2.11.4.2 Subscription to Connection State of Remote Device

```
// Subscription to the state of remote device
// Callback declaration of remote device state. Any change in state will return through this callback.
    void CALLBACK AfCameraStateCallBack(LLONG lLoginID, LLONG lAttachHandle, const
NET_CB_CAMERASTATE *pBuf, int nBufLen, LDWORD dwUser);

    // Subscription to the state of remote device
    NET_IN_CAMERASTATE stIn1 = {sizeof(stIn1)};
    int nChannel = -1;
    stIn1.pChannels = &nChannel;
    stIn1.nChannels = 4;
    stIn1.cbCamera = AfCameraStateCallBack;
    NET_OUT_CAMERASTATE stOut1 = {sizeof(stOut1)};
    LLONG llRet = CLIENT_AttachCameraState(lLoginHandle,&stIn1,&stOut1,3000);
    if(llRet)
    {
        printf("Subscription succeeded. Any following changes in state of front-end devices will be
uploaded.....\n");
    }
    else
    {
        printf("CLIENT_AttachCameraState Failed, Last Error[%x]\n",
        CLIENT_GetLastError());
    }
```

```
    // Stops subscription of remote device state
    If(llRet)
    {
        BOOL bRet = CLIENT_DetachCameraState(llRet);
        If(bRet)
        {
            printf(“Subscription cancel succeeded.\n”);
        }
        else
        {
            printf("CLIENT_DetachCameraState Failed, Last Error[%x]\n",
                CLIENT_GetLastError());
         }
    }

    // Definition of state callback of remote device
    void CALLBACK AfCameraStateCallBack(LLONG lLoginID, LLONG lAttachHandle, const
NET_CB_CAMERASTATE *pBuf, int nBufLen, LDWORD dwUser)
    {
        if(pBuf->emConnectState == 0)
        {
            printf("Current channel %d state is<<Disconnected>>\n", pBuf->nChannel);
        }
        else if(pBuf->emConnectState == 1)
        {
            printf("Current channel %d state is<<Connecting>>\n", pBuf->nChannel);
        }
        else if(pBuf->emConnectState == 2)
        {
            printf("Current channel %d state is<<Connected>>\n", pBuf->nChannel);
        }
}
```

2.11.4.3 Access to Information of Remote Device

```
// Get the remote device information interface 1
// nChanNum is the channel numbers returned by the login interface
NET_IN_GET_CAMERA_STATEINFO stuInfo = { sizeof(NET_IN_GET_CAMERA_STATEINFO),
TRUE };
```

```
NET_OUT_GET_CAMERA_STATEINFO stuOutInfo =
{ sizeof(NET_OUT_GET_CAMERA_STATEINFO) };
NET_CAMERA_STATE_INFO* pstuArrayStatInfo = new NET_CAMERA_STATE_INFO[nChanNum];
memset(pstuArrayStatInfo,0,sizeof(NET_CAMERA_STATE_INFO)*nChanNum);
stuOutInfo.nMaxNum = nChanNum;
stuOutInfo.pCameraStateInfo = pstuArrayStatInfo;
printf("State(0:UNKNOWN,1:CONNECTING,2:CONNECTED,3:UNCONNECT,4:EMPTY,5:DISABLE).\n
");
BOOL bRet = CLIENT_QueryDevInfo(lLoginHandle, NET_QUERY_GET_CAMERA_STATE, &stuInfo,
&stuOutInfo, NULL, 2000);
if (bRet)
{
    printf("CLIENT_QueryDevInfo NET_QUERY_GET_CAMERA_STATE success.\n");
    for (int i = 0; i < stuOutInfo.nValidNum; i++)
    {
    printf("channel:%d,Status:%d.\n",
stuOutInfo.pCameraStateInfo[i].nChannel,stuOutInfo.pCameraStateInfo[i].emConnectionState);
    }
}
else
{
    printf("CLIENT_QueryDevInfo Failed, Last Error[%x]\n",
    CLIENT_GetLastError());
}

// Get the remote device information interface 1
DH_IN_MATRIX_GET_CAMERAS stuInParm = {sizeof(DH_IN_MATRIX_GET_CAMERAS)};
DH_OUT_MATRIX_GET_CAMERAS stuOutParam = {sizeof(DH_OUT_MATRIX_GET_CAMERAS)};
DH_MATRIX_CAMERA_INFO   stuAllmatrixcamerinfo[128] = {0};
stuOutParam.nMaxCameraCount = nChanNum; // The maximum obtained number.
stuOutParam.pstuCameras = stuAllmatrixcamerinfo;
for (int i=0;i< __min(stuOutParam.nMaxCameraCount,stuOutParam.nRetCameraCount);++i)
{
    stuOutParam.pstuCameras[i].dwSize = sizeof(DH_MATRIX_CAMERA_INFO);
    stuOutParam.pstuCameras[i].stuRemoteDevice.dwSize = sizeof(DH_REMOTE_DEVICE);
}
int iNumbers = 1;
// Get all the valid display sources
BOOL bRet = CLIENT_MatrixGetCameras(lLoginHandle, &stuInParm, &stuOutParam);
printf("ALL the Device list Info Begin:\n");
```

```
if(bRet)
{
    int iChannelNumbers =0;
    char szUserInput[32] = "";
    memset(szUserInput, 0, sizeof(szUserInput));
    printf("too many channels info:Input    your show numbers: ==>\n");
    gets(szUserInput);
    iChannelNumbers = atoi(szUserInput);
    for ( int j=0;j<__min(stuOutParam.nRetCameraCount,iChannelNumbers);++j)
    {
        DH_MATRIX_CAMERA_INFO stuinfo = stuOutParam.pstuCameras[j];
        if( TRUE)// Whether remotely connects to the device
        {
        switch (stuinfo.emChannelType)
        {
        case LOGIC_CHN_REMOTE:
                {
                    printf("This is LOGIC_CHN_REMOTE(Remote channel):\n");
                    break;
                }
        case LOGIC_CHN_LOCAL:
                {
                    printf("This is LOGIC_CHN_LOCAL(Local channel):\n");
                    break;
                 }
        case LOGIC_CHN_COMPOSE:
                 {
                    printf("This is LOGIC_CHN_COMPOSE(Composed channel):\n");
                    break;
                 }
                    case LOGIC_CHN_MATRIX:
                 {
                    printf("This is LOGIC_CHN_MATRIX(Analog matrix channel):\n");
                    break;
                 }
                    case LOGIC_CHN_CASCADE:
                 {
                    printf("This is LOGIC_CHN_CASCADE(Cascade channel):\n");
                    break;
                 }
```

```
            default:
                {
                        printf("This is LOGIC_CHN_UNKNOWN(Unknown channel):\n");
                }
            }
                printf("..................................................\n");
                printf("This is the %d remote camera:\n",iNumbers++);
                printf("Dev Remote ChannelID = %d,the Local
                nUniqueChannel = %d.\n",stuinfo.nChannelID,stuinfo.nUniqueChannel);
                printf("Dev Local szDevID = %s,
                the Local szName = %s.\n",stuinfo.szDevID,stuinfo.szName);
                DH_REMOTE_DEVICE stuRemoteDevice = stuinfo.stuRemoteDevice;

                printf("RemoteDev IP = %s,
                RemoteDev Port = %d.\n",stuRemoteDevice.szIp,stuRemoteDevice.nPort);
              }
        }
}
else
{
printf("CLIENT_MatrixGetCameras Failed, Last Error[%x]\n",
CLIENT_GetLastError());
}
```

2.11.4.4 Access to Channel Name of Remote Device

```
// Get the remote device information interface 1
// Get the relevant configurations and then modify them.
// The example code only demonstrates the command CFG_CMD_CHANNELTITLE. The command CFG_CMD_VIDEOIN can refer to this example.
    #define MAX_SPACE_NUM 50
    char * szOut = new char[1024* MAX_SPACE_NUM];
    AV_CFG_ChannelName *stOut = new AV_CFG_ChannelName[MAX_SPACE_NUM];
    memset(szOut,0,1024*MAX_SPACE_NUM);//Initialize
    memset(stOut,0,sizeof(*stOut)*MAX_SPACE_NUM);
    for(int i=0; i<MAX_SPACE_NUM;i++)
    {
        stOut[i].nStructSize = sizeof(stOut[i]);//Valuation of structure size
    }
```

```
    int nError = 0;
    BOOL bRet = 0;
    // Get the configuration
    bRet = CLIENT_GetNewDevConfig(lLoginHandle,CFG_CMD_CHANNELTITLE,-1,
        szOut,MAX_SPACE_NUM*1024,&nError,3000);
    if(bRet)
    {
        int nRetLen = 0;
        bRet= CLIENT_ParseData(CFG_CMD_CHANNELTITLE,szOut,stOut,
            MAX_SPACE_NUM*sizeof(*stOut),&nRetLen);
        if(bRet)
        {
            int nRetNum = nRetLen/sizeof(*stOut);
            for(int n=0; n<nRetNum;n++)
            {
                AV_CFG_ChannelName stTitle = stOut[n];
                printf("channel %d title is %s\n",n,stTitle.szName);
            }
            printf("get succeed\n");
        }
        else
        {
            printf("CLIENT_ParseData Failed, Last Error[%x]\n",
                CLIENT_GetLastError());
        }
    }
    else
    {
        printf("CLIENT_GetNewDevConfig Failed, Last Error[%x]\n",
            CLIENT_GetLastError());
    }

    // Modify configuration
    printf("Modify the first channel name to: AAA\n");
    char szName1[CFG_MAX_CHANNEL_NAME_LEN] = {"AAA"};
    memcpy(stOut[0].szName, szName1, CFG_MAX_CHANNEL_NAME_LEN);

    bRet = CLIENT_PacketData(CFG_CMD_CHANNELTITLE,stOut,
sizeof(AV_CFG_ChannelName),szOut,1024*MAX_SPACE_NUM);
```

```
    if(bRet)
    {
        bRet = CLIENT_SetNewDevConfig(lLoginHandle,CFG_CMD_CHANNELTITLE,0,
            szOut,1024*MAX_SPACE_NUM,NULL,NULL,3000);
        //Modify which channel name, which number to enter, and the channel number starts from 0. Modify the first channel name and enter 0 for this channel.
        if(bRet)
        {
            printf("set succeed\n");
        }
        else
        {
            printf("CLIENT_SetNewDevConfig Failed, Last Error[%x]\n",
                CLIENT_GetLastError());
        }
    }
    else
    {
        printf("CLIENT_PacketData Failed, Last Error[%x]\n",
            CLIENT_GetLastError());
            }
// Get the remote device information interface 2
// This interface suits for DVR analog device. Because the DVR device is disappearing from the market, please pay attention to the usage.
char * szOut = new char[32*18];
int nChannelCount = 0;
BOOL bRet = CLIENT_QueryChannelName(lLoginHandle,szOut,32*16,&nChannelCount,3000);
if(bRet)
{
    for(int i = 0; i<nChannelCount; i++)
    {
        cout << "channel:" << i <<","<<"Channel name:"<<szOut<<endl;
        szOut = szOut +32;
    }
}
else
{
    printf("CLIENT_QueryChannelName Failed, Last Error[%x]\n",
            CLIENT_GetLastError());
}
```

3 Interface Definition

3.1 SDK Initialization

3.1.1 SDK CLIENT_Init

Item	Description	
Name	Initialize SDK.	
Function	BOOL CLIENT_Init(fDisConnect cbDisConnect, LDWORD dwUser);	
Parameter	[in]cbDisConnect	Disconnection callback.
	[in]dwUser	User parameter of disconnection callback.
Return value	● Success: TRUE ● Failure: FALSE	
Note	● The precondition for calling other function modules of SDK. ● The callback will not send to the user after the device is disconnected if the callback is set as NULL.	

3.1.2 CLIENT_Cleanup

Item	Description
Name	Clean up SDK.
Function	void CLIENT_Cleanup();
Parameter	None.
Return value	None.
Note	Call SDK cleanup interface before the process stops.

3.1.3 CLIENT_SetAutoReconnect

Item	Description	
Name	Set auto reconnection for callback.	
Function	void CLIENT_SetAutoReconnect(fHaveReConnect cbAutoConnect, LDWORD dwUser);	
Parameter	[in]cbAutoConnect	Reconnection callback.
	[in]dwUser	User parameter of disconnection callback.

Item	Description
Return value	None.
Note	Set the reconnection callback interface. If the callback is set as NULL, it will not connect automatically.

3.1.4 CLIENT_SetNetworkParam

Item	Description	
Name	Set the related parameters for network environment.	
Function	void CLIENT_SetNetworkParam(NET_PARAM *pNetParam);	
Parameter	[in]pNetParam	Parameters such as network delay, reconnection times, and cache size.
Return value	None.	
Note	Adjust the parameters according to the actual network environment.	

3.2 Device Initialization

3.2.1 CLIENT_StartSearchDevices

Item	Description	
Name	Search the device.	
Function	LLONG CLIENT_StartSearchDevices (fSearchDevicesCB cbSearchDevices, void* pUserData, char* szLocalIp=NULL);	
Parameter	[in]cbSearchDevices	Device information callback.
	[out]pUserData	User data.
	[in]szLocalIp	● In case of single network card, enter NULL, which means using the host PC IP. ● In case of multiple network card, enter the IP of the specified network card.
Return value	Searching handle.	
Note	Multi-thread calling is not supported.	

3.2.2 CLIENT_InitDevAccount

Item	Description
Name	Initialize the device.

Item	Description	
Function	BOOL CLIENT_InitDevAccount(const NET_IN_INIT_DEVICE_ACCOUNT *pInitAccountIn, NET_OUT_INIT_DEVICE_ACCOUNT *pInitAccountOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pInitAccountIn	Corresponds to structure of NET_IN_INIT_DEVICE_ACCOUNT.
	[out]pInitAccountOut	Corresponds to structure of NET_OUT_INIT_DEVICE_ACCOUNT.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.3 CLIENT_GetDescriptionForResetPwd

Name	Description	
Name	Get information for password reset.	
Function	BOOL CLIENT_GetDescriptionForResetPwd(const NET_IN_DESCRIPTION_FOR_RESET_PWD *pDescriptionIn, NET_OUT_DESCRIPTION_FOR_RESET_PWD *pDescriptionOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pDescriptionIn	Corresponds to structure of NET_IN_DESCRIPTION_FOR_RESET_PWD.
	[out]pDescriptionOut	Corresponds to structure of NET_OUT_DESCRIPTION_FOR_RESET_PWD.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.4 CLIENT_CheckAuthCode

Item	Description	
Name	Check the validity of security code.	
Function	BOOL CLIENT_CheckAuthCode(const NET_IN_CHECK_AUTHCODE *pCheckAuthCodeIn, NET_OUT_CHECK_AUTHCODE *pCheckAuthCodeOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pCheckAuthCodeIn	Corresponds to structure of NET_IN_CHECK_AUTHCODE.
	[out]pCheckAuthCodeOut	Corresponds to structure of NET_OUT_CHECK_AUTHCODE.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.5 CLIENT_ResetPwd

Item	Description	
Name	Reset the password.	
Function	BOOL CLIENT_ResetPwd(const NET_IN_RESET_PWD *pResetPwdIn, NET_OUT_RESET_PWD *pResetPwdOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pResetPwdIn	Corresponds to structure of NET_IN_RESET_PWD.
	[out]pResetPwdOut	Corresponds to structure of NET_OUT_RESET_PWD.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.6 CLIENT_GetPwdSpecification

Item	Description	
Name	Get password rules.	
Function	BOOL CLIENT_GetPwdSpecification(const NET_IN_PWD_SPECI *pPwdSpeciIn, NET_OUT_PWD_SPECI *pPwdSpeciOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pPwdSpeciIn	Corresponds to structure of NET_IN_PWD_SPECI.
	[out]pPwdSpeciOut	Corresponds to structure of NET_OUT_PWD_SPECI.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.7 CLIENT_StopSearchDevices

Item	Description	
Name	Stop searching.	
Function	BOOL CLIENT_StopSearchDevices (LLONG lSearchHandle);	
Parameter	[in] lSearchHandle	Searching handle.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	Multi-thread calling is not supported.	

3.3 Device Login

3.3.1 CLIENT_LoginEx2

Item	Description
Name	Login the device.

Item	Description	
Function	LLONG CLIENT_LoginEx2(const char *pchDVRIP, WORD wDVRPort, const char *pchUserName, const char *pchPassword, EM_LOGIN_SPAC_CAP_TYPE emSpecCap, void* pCapParam, LPNET_DEVICEINFO_Ex lpDeviceInfo, int *error);	
Parameter	[in]pchDVRIP	Device IP.
	[in]wDVRPort	Device port.
	[in]pchUserName	User name.
	[in]pchPassword	Password.
	[in]emSpecCap	Login category.
	[in]pCapParam	Login category parameter.
	[out]lpDeviceInfo	Device information.
	[out]error	Error for login failure.
Return value	● Success: Not 0. ● Failure: 0.	
Note	None.	

The following table shows information about error code:

Code of error	Meaning
1	Wrong password.
2	The user name does not exist.
3	Login timeout.
4	The account has logged in.
5	The account has been locked.
6	The account has been blacklisted.
7	The device resource is insufficient and the system is busy.
8	Sub connection failed.
9	Main connection failed.
10	Exceeds the maximum allowed number of user connections.
11	Lacks the dependent libraries such as avnetsdk or avnetsdk.
12	USB flash disk is not inserted or the USB flash disk information is wrong.
13	The IP at client is not authorized for login.

Table 3-1

3.3.2 CLIENT_Logout

Item	Description
Name	Logout the device.

Item	Description	
Function	BOOL CLIENT_Logout(LLONG lLoginID);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginEx2.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.4 Real-time Monitoring

3.4.1 CLIENT_RealPlayEx

Item	Description	
Name	Open the real-time monitoring	
Function	LLONG CLIENT_RealPlayEx(LLONG lLoginID, int nChannelID, HWND hWnd, DH_RealPlayType rType);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginEx2
	[in]nChannelID	Video channel number is a round number starting from 0.
	[in]hWnd	Window handle valid only under Windows system
	[in]rType	Preview type
Return value	● Success: not 0 ● Failure: 0	
Note	Windows system: ● When hWnd is valid, the corresponding window displays picture. ● When hWnd is NULL, get the video data through setting a callback and send to user for handle.	

The following table shows information about preview type:

Preview type	Meaning
DH_RType_Realplay	Real-time preview.
DH_RType_Multiplay	Multi-picture 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 preview—1 picture.
DH_RType_Multiplay_4	Multi-picture preview—4 pictures.
DH_RType_Multiplay_8	Multi-picture preview—8 pictures.

Preview type	Meaning
DH_RType_Multiplay_9	Multi-picture preview—9 pictures.
DH_RType_Multiplay_16	Multi-picture preview—16 pictures.
DH_RType_Multiplay_6	Multi-picture preview—6 pictures.
DH_RType_Multiplay_12	Multi-picture preview—12 pictures.
DH_RType_Multiplay_25	Multi-picture preview—25 pictures.
DH_RType_Multiplay_36	Multi-picture preview—36 pictures.

Table 3-2

3.4.2 CLIENT_StopRealPlayEx

Item	Description	
Name	Stop the real-time monitoring.	
Function	BOOL CLIENT_StopRealPlayEx(LLONG lRealHandle);	
Parameter	[in]lRealHandle	Return value of CLIENT_RealPlayEx.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.4.3 CLIENT_SaveRealData

Item	Description	
Name	Save the real-time monitoring data as file.	
Function	BOOL CLIENT_SaveRealData(LLONG lRealHandle, const char *pchFileName);	
Parameter	[in] lRealHandle	Return value of CLIENT_RealPlayEx.
	[in] pchFileName	Save path.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.4.4 CLIENT_StopSaveRealData

Item	Description	
Name	Stop saving the real-time monitoring data as file.	
Function	BOOL CLIENT_StopSaveRealData(LLONG lRealHandle);	
Parameter	[in] lRealHandle	Return value of CLIENT_RealPlayEx.
Return value	● Success: TRUE. ● Failure: FALSE.	

Item	Description
Note	None.

3.4.5 CLIENT_SetRealDataCallBackEx2

Item	Description	
Name	Set the callback of real-time monitoring data.	
Function	BOOL CLIENT_SetRealDataCallBackEx2 (LLONG lRealHandle, fRealDataCallBackEx2 cbRealData, LDWORD dwUser, DWORD dwFlag);	
Parameter	[in] lRealHandle	Return value of CLIENT_RealPlayEx.
	[in] cbRealData	Callback of monitoring data flow.
	[in] dwUser	Parameter of callback for monitoring data flow.
	[in] dwFlag	Type of monitoring data in callback. The type is EM_REALDATA_FLAG and supports OR operation.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

The following table shows information about parameter dwFlag:

dwFlag	Meaning
REALDATA_FLAG_RAW_DATA	Initial data labels.
REALDATA_FLAG_DATA_WITH_FRAME_INFO	Data labels with frame information.
REALDATA_FLAG_YUV_DATA	YUV data labels.
REALDATA_FLAG_PCM_AUDIO_DATA	PCM audio data labels.

Table 3-3

3.5 Playback

3.5.1 CLIENT_PlayBackByTimeEx2

Item	Description	
Name	Playback by time.	
Function	LLONG CLIENT_PlayBackByTimeEx2(LLONG lLoginID, int nChannelID, NET_IN_PLAY_BACK_BY_TIME_INFO *pstNetIn, NET_OUT_PLAY_BACK_BY_TIME_INFO *pstNetOut);	
Parameter	[in] lLoginID	The return value of CLIENT_LoginEx2.

Item	Description	
	[in] nChannelID	Device channel number.
	[in] pstNetIn	Query on input condition.
	[out] pstNetOut	Query on output information.
Return value	● Success: Network playback ID. ● Failure: 0.	
Note	For fDataCallBack and fDownLoadPosCallBack in NET_IN_PLAY_BACK_BY_TIME_INFO, see "Chapter 4 Callback Definition". The parameters hWnd and fDownLoadDataCallBack cannot be NULL at the same time; otherwise the interface calling will be failed returned.	

3.5.2 CLIENT_SetDeviceMode

Item	Description	
Name	Set the work mode.	
Function	BOOL CLIENT_SetDeviceMode(LLONG ILoginID, EM_USEDEV_MODE emType, void *pValue);	
Parameter	[in] ILoginID	Return value of CLIENT_LoginEx2.
	[in] emType	Work mode enumeration.
	[in] pValue	The structure corresponding to work mode.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

The following table shows information about work mode enumeration and structure:

emType enumeration	Meaning	Structure
DH_RECORD_STREAM_TYPE	Set the record stream type to query and playback by time. It can be considered as int type. ● 0: Main and sub stream ● 1: Main stream ● 2: Sub stream	None
DH_RECORD_TYPE	Set the record file type to playback and download by time.	NET_RECORD_TYPE

Table 3-4

3.5.3 CLIENT_StopPlayBack

Item	Description
Name	Stop video playback.
Function	BOOL CLIENT_StopPlayBack(LLONG 1PlayHandle

Item	Description	
	);	
Parameter	[in] 1PlayHandle	Return value of playback interface.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.5.4 CLIENT_GetPlayBackOsdTime

Item	Description	
Name	Get the OSD playback time.	
Function	BOOL CLIENT_GetPlayBackOsdTime(LLONG lPlayHandle, LPNET_TIME lpOsdTime, LPNET_TIME lpStartTime, LPNET_TIME lpEndTime);	
Parameter	[in] lPlayHandle	Return value of playback interface.
	[out] lpOsdTime	OSD time.
	[out] lpOsdTime	The start time of current playback file.
	[out] lpEndTime	The end time of current playback file.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.5.5 CLIENT_PausePlayBack

Item	Description	
Name	Pause or resume playback.	
Function	BOOL CLIENT_PausePlayBack(LLONG lPlayHandle, BOOL bPause);	
Parameter	[in] lPlayHandle	Return value of playback interface.
	[out] bPause	Parameters for network playback stops and resumes: 1: Pause 0: Resume
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	Pause the ongoing playing and resume control.	

3.6 Record Download

3.6.1 CLIENT_QueryRecordFile

Item	Description	
Name	Query all record files within a period.	
Function	BOOL CLIENT_QueryRecordFile(LLONG lLoginID, int nChannelId, int nRecordFileType, LPNET_TIME tmStart, LPNET_TIME tmEnd, char* pchCardid, LPNET_RECORDFILE_INFO nriFileinfo, int maxlen, int *filecount, int waittime=1000, BOOL bTime = FALSE);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] nChannelId	Device channel number starting from 0.
	[in] nRecordFileType	Record file type.
	[in] tmStart	Record start time.
	[in] tmEnd	Record end time.
	[in] pchCardid	Card ID.
	[out] nriFileinfo	The returned record file is a LPNET_RECORDFILE_INFO structured data.
	[in] maxlen	The maximum length of nriFileinfo buffer, which unit is byte and recommended to be between " (100~200) *sizeof(NET_RECORDFILE_INFO)".
	[out] filecount	Check the number of returned files only in the cache.
	[in] waittime	Waiting time.
	[in] bTime	Whether to query by time.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	Before playback, call this interface to query the records. When the queried records within the input time are larger than the cache size, it will only return the records that can be stored by cache. Continue with the query if needed.	

The following table shows information about record file and card ID:

Value	Record file type	Card ID
0	All record files	NULL
1	External alarm	NULL
2	Alarm by dynamical detection	NULL
3	All the alarms	NULL
4	Card IP query	Card ID

Value	Record file type	Card ID
5	Combined conditions query	Card ID && Transaction type && Transaction amount (If you want to skip a field, set as blank)
6	Record location and deviation length	NULL
8	Pictures queried by card ID (Only supported by some models of HB-U and NVS)	Card ID
9	Query pictures (Only supported by some models of HB-U and NVS)	NULL
10	Query by field	FELD1&&FELD2&&FELD3&& (If you want to skip a field, set as blank)

Table 3-5

3.6.2 CLIENT_FindFile

Item	Description	
Name	Open the record query handle.	
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	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] nChannelId	Device channel number starting from 0.
	[in] nRecordFileType	Record file type. For details, see Table 3-5.
	[in] cardid	Card ID, which is valid for card query. For other situations, enter **NULL**.
	[in] time_start	Query the record start time.
	[in] time_end	Query the record end time.
	[in] bTime	Whether to query by time.
	[in] waittime	Query timeout.
Return value	● Success: Query handle. ● Failure: 0.	
Note	None.	

3.6.3 CLIENT_FindNextFile

Item	Description

Item	Description	
Name	Find the record file.	
Function	int CLIENT_FindNextFile(LLONG lFindHandle, LPNET_RECORDFILE_INFO lpFindData);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindFile(Query handle of opening record).
	[in] lpFindData	Record file cache used to output the queried record file.
Return value	● 1: Get one record successfully. ● 0: Getting all the records. ● -1: Parameter error.	
Note	Before calling this interface, call CLIENT_FindFile to open the query handle.	

3.6.4 CLIENT_FindClose

Item	Description	
Name	Close the record query handle.	
Function	BOOL CLIENT_FindClose(LLONG lFindHandle);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindFile (open the record query handle).
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	Call CLIENT_FindFile to open the query handle. After query is finished, call this function to close the query handle.	

3.6.5 CLIENT_DownloadByRecordFileEx

Item	Description	
Name	Download record by file.	
Function	LLONG CLIENT_DownloadByRecordFileEx(LLONG lLoginID, LPNET_RECORDFILE_INFO lpRecordFile, char *sSavedFileName, fDownLoadPosCallBack cbDownLoadPos, LDWORD dwUserData, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser, void* pReserved = NULL);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] lpRecordFile	The information pointer of record file.

Item	Description	
	[in] sSavedFileName	The record file name and full save path.
	[in] cbDownLoadPos	Download progress callback.
	[in] dwUserData	Call the customized data.
	[in] fDownLoadDataCallBack	Data callback.
	[in] pReserved	Parameter reserved and the default is NULL.
Return value	● Success: Download ID. ● Failure: 0.	
Note	● For callback declaration of fDownLoadPosCallBack and fDataCallBack, see "Chapter 4 Callback Definition." ● sSavedFileName is not blank, and the record data is input into the file corresponding with the path. ● fDownLoadDataCallBack is not blank, and the record data is returned through callback.	

3.6.6 CLIENT_DownloadByTimeEx

Item	Description	
Name	Download record by time.	
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	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] nChannelId	The device channel number starting from 0.
	[in] nRecordFileType	Query type of file. 0: All record files. 1: External alarm. 2: Records detected dynamically. 3: All alarms. 4: Query record by card ID. 5: Query by combined conditions. 8: Query pictures by Card ID. 9: Query pictures. 10: Query by field.
	[in] tmStart	Start time of download.

Item	Description	
	[in] tmEnd	End time of download.
	[in] sSavedFileName	The record file name and full save path.
	[in]cbTimeDownLoadPos	Download progress callback.
	[in] dwUserData	Download progress callback customized data.
	[in] fDownLoadDataCallBack	Data callback.
	[in] dwUserData	Download data callback customized data.
	[in] pReserved	Parameter reserved and the default is NULL.
Return value	● Success: Download ID. ● Failure: 0.	
Note	● For callback declaration of fDownLoadPosCallBack and fDataCallBack, see "Chapter 4 Callback Definition." ● sSavedFileName is not blank, and the record data is input into the file corresponding with the path. ● fDownLoadDataCallBack is not blank, and the record data is returned through callback.	

3.6.7 CLIENT_GetDownloadPos

Item	Description	
Name	Query the record downloading progress.	
Function	BOOL CLIENT_GetDownloadPos(LLONG lFileHandle, int *nTotalSize, int *nDownLoadSize);	
Parameter	[in] lFileHandle	Return value of download interface.
	[out] nTotalSize	The total length of download and the unit is KB.
	[out] nDownLoadSize	The downloaded length and the unit is KB.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	● Get the current location of the record to be downloaded to apply to display interface that does not need to display real-time download progress. It is similar to the function of download callback. ● Calculate the progress without using the callback. You can call this interface regularly to get the current progress.	

3.6.8 CLIENT_StopDownload

Item	Description	
Name	Stop record downloading.	
Function	BOOL CLIENT_StopDownload(LLONG lFileHandle);	
Parameter	[in] lFileHandle	Return value of download interface

Item	Description
Return value	● Success: TRUE. ● Failure: FALSE.
Note	Stop downloading after it is completed or partially completed according to particular situation.

3.7 PTZ Control

3.7.1 CLIENT_DHPTZControlEx2

Item	Description	
Name	PTZ control.	
Function	BOOL CLIENT_DHPTZControlEx2(LLONG lLoginID, int nChannelID, DWORD dwPTZCommand, LONG lParam1, LONG lParam2, LONG lParam3, BOOL dwStop , void* param4);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] nChannelID	Video channel number that is a whole number and starts from 0.
	[in] dwPTZCommand	Control command type.
	[in] lParam1	Parameter 1.
	[in] lParam2	Parameter 2.
	[in] lParam3	Parameter 3.
	[in] dwStop	Stop mark, which is valid for operations of eight directions. When performing other operations, enter **FALSE** for this parameter.
	[in] param4	Support the following extension command: DH_EXTPTZ_MOVE_ABSOLUTELY DH_EXTPTZ_MOVE_CONTINUOUSLY DH_EXTPTZ_GOTOPRESET DH_EXTPTZ_SET_VIEW_RANGE DH_EXTPTZ_FOCUS_ABSOLUTELY DH_EXTPTZ_HORSECTORSCAN DH_EXTPTZ_VERSECTORSCAN DH_EXTPTZ_SET_FISHEYE_EPTZ
Return value	● Success: TRUE. ● Failure: FALSE.	

Item	Description
Note	For the relationship between dwPTZCommand and Param1, Param2 and Param3, see Table 3-6.

For the relationship between dwPTZCommand and Param1, Param2 and Param3, see Table 3-6.

dwPTZCommand macro definition	Function	param1	param2	param3
DH_PTZ_UP_CONTROL	Up	None	Vertical speed (1–8)	None
DH_PTZ_DOWN_CONTROL	Down	None	Vertical speed (1–8)	None
DH_PTZ_LEFT_CONTROL	Left	None	Horizontal speed (1–8)	None
DH_PTZ_RIGHT_CONTROL	Right	None	Horizontal speed (1–8)	None
DH_PTZ_ZOOM_ADD_CONTROL	Zoom+	None	Multi-speed	None
DH_PTZ_ZOOM_DEC_CONTROL	Zoom-	None	Multi-speed	None
DH_PTZ_FOCUS_ADD_CONTROL	Focus+	None	Multi-speed	None
DH_PTZ_FOCUS_DEC_CONTROL	Focus-	None	Multi-speed	None
DH_PTZ_APERTURE_ADD_CONTROL	Aperture+	None	Multi-speed	None
DH_PTZ_APERTURE_DEC_CONTROL	Aperture -	None	Multi-speed	None
DH_PTZ_POINT_MOVE_CONTROL	Move to preset point	None	Value of preset point	None
DH_PTZ_POINT_SET_CONTROL	Set	None	Value of preset point	None
DH_PTZ_POINT_DEL_CONTROL	Delete	None	Value of preset point	None
DH_PTZ_POINT_LOOP_CONTROL	Cruise among points	Cruise route	None	76:Start 99:Automatic 96:Stop
DH_PTZ_LAMP_CONTROL	Lamp wiper	0x01: Start x00: Stop	None	None
DH_EXTPTZ_LEFTTOP	Left top	Vertical speed (1–8)	Horizontal speed (1–8)	None
DH_EXTPTZ_RIGHTTOP	Right top	Vertical speed (1–8)	Horizontal speed (1–8)	None
DH_EXTPTZ_LEFTDOWN	Left bottom	Vertical speed (1–8)	Horizontal speed (1–8)	None

dwPTZCommand macro definition	Function	param1	param2	param3
DH_EXTPTZ_RIGHTDOWN	Right bottom	Vertical speed (1–8)	Horizontal speed (1–8)	None
DH_EXTPTZ_ADDTOLOOP	Add preset point to cruise	Cruise route	Value of preset point	None
DH_EXTPTZ_DELFROMLOOP	Delete preset point in cruise	Cruise route	Value of preset point	None
DH_EXTPTZ_CLOSELOOP	Delete cruise	Cruise route	None	None
DH_EXTPTZ_STARTPANCRUISE	Start horizontal rotation	None	None	None
DH_EXTPTZ_STOPPANCRUISE	Stop horizontal rotation	None	None	None
DH_EXTPTZ_SETLEFTBORDER	Set left border	None	None	None
DH_EXTPTZ_RIGHTBORDER	Set right border	None	None	None
DH_EXTPTZ_STARTLINESCAN	Start line scan	None	None	None
DH_EXTPTZ_CLOSELINESCAN	Stop line scan	None	None	None
DH_EXTPTZ_SETMODESTART	Set mode start	Mode route	None	None
DH_EXTPTZ_SETMODESTOP	Set mode stop	Mode route	None	None
DH_EXTPTZ_RUNMODE	Running mode	Mode route	None	None
DH_EXTPTZ_STOPMODE	Stop mode	Mode route	None	None
DH_EXTPTZ_DELETEMODE	Delete mode	Mode route	None	None
DH_EXTPTZ_REVERSECOMM	Reverse command	None	None	None
DH_EXTPTZ_FASTGOTO	Fast positioning	Horizontal coordinate (0–8192)	Vertical coordinate (0–8192)	Zoom (4)
DH_EXTPTZ_AUXIOPEN	Open auxiliary switch	Auxiliary point	None	None
DH_EXTPTZ_AUXICLOSE	Close auxiliary switch	Auxiliary point	None	None
DH_EXTPTZ_OPENMENU	Open SD menu	None	None	None
DH_EXTPTZ_CLOSEMENU	Close menu	None	None	None
DH_EXTPTZ_MENUOK	Menu confirm	None	None	None

dwPTZCommand macro definition	Function	param1	param2	param3
DH_EXTPTZ_MENUCANCEL	Menu cancel	None	None	None
DH_EXTPTZ_MENUUP	Menu up	None	None	None
DH_EXTPTZ_MENUDOWN	Menu down	None	None	None
DH_EXTPTZ_MENULEFT	Menu left	None	None	None
DH_EXTPTZ_MENURIGHT	Menu right	None	None	None
DH_EXTPTZ_ALARMHANDLE	Alarm action with PTZ	Alarm input channel	Alarm action type: ● Preset point ● Line scan ● Cruise	Linkage value, such as preset point number
DH_EXTPTZ_MATRIXSWITCH	Matrix switch	Monitor device number (video output number)	Video input number	Matrix number
DH_EXTPTZ_LIGHTCONTROL	Light controller	Refer to DH_PTZ_LAMP_CONTROL	None	None
DH_EXTPTZ_EXACTGOTO	3D positioning	Horizontal angle (0–3600)	Vertical coordinate (0–900)	Zoom (1–128)
DH_EXTPTZ_RESETZERO	Reset to zero		None	None
DH_EXTPTZ_UP_TELE	UP +TELE	Speed (1–8)	None	None
DH_EXTPTZ_DOWN_TELE	DOWN +TELE	Speed (1–8)	None	None
DH_EXTPTZ_LEFT_TELE	LEFT +TELE	Speed (1–8)	None	None
DH_EXTPTZ_RIGHT_TELE	RIGHT +TELE	Speed (1–8)	None	None
DH_EXTPTZ_LEFTUP_TELE	LEFTUP +TELE	Speed (1–8)	None	None
DH_EXTPTZ_LEFTDOWN_TELE	LEFTDOWN +TELE	Speed (1–8)	None	None
DH_EXTPTZ_TIGHTUP_TELE	TIGHTUP +TELE	Speed (1–8)	None	None
DH_EXTPTZ_RIGHTDOWN_TELE	RIGHTDOWN +TELE	Speed (1–8)	None	None
DH_EXTPTZ_UP_WIDE	UP +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_DOWN_WIDE	DOWN +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_LEFT_WIDE	LEFT +WIDE	Speed (1–8)	None	None

dwPTZCommand macro definition	Function	param1	param2	param3
DH_EXTPTZ_RIGHT_WIDE	RIGHT +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_LEFTUP_WIDE	LEFTUP +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_LEFTDOWN_WIDE	LEFTDOWN +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_RIGHTUP_WIDE	RIGHTUP +WIDE	Speed (1–8)	None	None
DH_EXTPTZ_RIGHTDOWN_WIDE	RIGHTDOWN +WIDE	Speed (1–8)	None	None

Table 3-6

3.8 Voice Talk

3.8.1 CLIENT_StartTalkEx

Item	Description	
Name	Open voice talk.	
Function	LLONG CLIENT_StartTalkEx(LLONG lLoginID, pfAudioDataCallBack pfcb, LDWORD dwUser);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] pfcb	Audio data callback.
	[in] dwUser	Parameter of audio data callback.
Return value	● Success: Not 0. ● Failure: 0.	
Note	None.	

3.8.2 CLIENT_StopTalkEx

Item	Description	
Name	Stop voice talk.	
Function	BOOL CLIENT_StopTalkEx(LLONG lTalkHandle);	
Parameter	[in] lTalkHandle	Return value of CLIENT_StartTalkEx.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.8.3 CLIENT_RecordStartEx

Item	Description	
Name	Start local recording.	
Function	BOOL CLIENT_RecordStartEx(LLONG lLoginID);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	Valid only in Windows system.	

3.8.4 CLIENT_RecordStopEx

Item	Description	
Name	Stop local recording.	
Function	BOOL CLIENT_RecordStopEx(LLONG lLoginID);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	Valid only in Windows system.	

3.8.5 CLIENT_TalkSendData

Item	Description	
Name	Send audio data to device.	
Function	LONG CLIENT_TalkSendData(LLONG lTalkHandle, char *pSendBuf, DWORD dwBufSize);	
Parameter	[in] lTalkHandle	Return value of CLIENT_StartTalkEx.
	[in] pSendBuf	Pointer of audio data block that needs sending.
	[in] dwBufSize	Length of audio data black that needs sending. The unit is byte.
Return value	● Success: Length of audio data block. ● Failure: -1.	
Note	None.	

3.8.6 CLIENT_AudioDecEx

Item	Description

Item	Description	
Name	Decode audio data.	
Function	BOOL CLIENT_AudioDecEx(LLONG lTalkHandle, char *pAudioDataBuf, DWORD dwBufSize);	
Parameter	[in] lTalkHandle	Return value of CLIENT_StartTalkEx
	[in] pAudioDataBuf	Pointer of audio data block that needs decoding.
	[in] dwBufSize	Length of audio data black that needs decoding. The unit is byte.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.9 Video Snapshot

3.9.1 CLIENT_SnapPictureToFile

Item	Description	
Name	Capture image.	
Function	BOOL CLIENT_SnapPictureToFile(LLONG lLoginID, const NET_IN_SNAP_PIC_TO_FILE_PARAM* pInParam, NET_OUT_SNAP_PIC_TO_FILE_PARAM* pOutParam, int nWaitTime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] pInParam	Input parameter.
	[out] pOutParam	Output parameter.
	[in] nWaitTime	Timeout. The unit is millisecond.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	● Synchronous interface. The device captures picture and sends to the user. ● This function is required on some devices.	

3.9.2 CLIENT_CapturePictureEx

Item	Description
Name	Capture image.
Function	BOOL CLIENT_CapturePictureEx(LLONG hPlayHandle, const char *pchPicFileName,

	NET_CAPTURE_FORMATS eFormat);	
Parameter	[in] hPlayHandle	Return value of CLIENT_RealPlayEx.
	[in] pchPicFileName	The file path to be saved must be the absolute path.
	[in] eFormat	Picture format.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	● Synchronous interface. Write the picture data into file. ● The picture is captured from the real-time monitoring data stream sent by the device.	

3.10 Alarm Upload

3.10.1 CLIENT_SetDVRMessCallBack

Item	Description	
Name	Set alarm callback.	
Function	void CLIENT_SetDVRMessCallBack(fMessCallBack cbMessage, LDWORD dwUser);	
Parameter	[in] cbMessage	● Message callback that can call the device status, such as alarm status. ● When setting as 0, the calling is prohibited.
	[in] dwUser	The user customized data.
Return value	None.	
Note	● Sets device message callback to obtain the device current status. It has nothing to do with the calling sequence. There's no calling by default. ● The callback fMessCallBack is valid after calling the alarm message subscription interface CLIENT_StartListenEx first.	

3.10.2 CLIENT_StartListenEx

Item	Description	
Name	Subscribe alarm.	
Function	BOOL CLIENT_StartListenEx(LLONG lLoginID);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
Return value	● Success: TRUE. ● Failure: FALSE.	

Item	Description
Note	The message from the subscribed device is called from the set value of CLIENT_SetDVRMessCallBack.

3.10.3 CLIENT_StopListen

Item	Description	
Name	Stop alarm subscription.	
Function	BOOL CLIENT_StopListen(LLONG lLoginID);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.11 Storage

3.11.1 CLIENT_QueryDevState

Item	Description	
Name	Directly get the connection status of remote device.	
Function	BOOL CLIENT_QueryDevState(LLONG lLoginID, int nType, char *pBuf, int nBufLen, int *pRetLen, int waittime=1000);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] nType	Query the information type. When the connection status of remote device is obtained, the type becomes DH_DEVSTATE_VIRTUALCAMERA.
	[out] pBuf	Receives the data cache returned from query. When the connection status of remote device is obtained, the corresponding structure is DHDEV_VIRTUALCAMERA_STATE_INFO.
	[in] nBufLen	Cache length. The unit is byte.
	[out] pRetLen	The returned data length. The unit is byte.
	[in] waittime	Waiting time for query. The default is 1000ms.
Return value	● Success: TRUE. ● Failure: FALSE.	

Item	Description
Note	None.

3.11.2 CLIENT_QueryDevInfo

Item	Description	
Name	Directly get the connection status of remote device.	
Function	BOOL CALL_METHOD CLIENT_QueryDevInfo(LLONG lLoginID, int nQueryType, void* pInBuf, void* pOutBuf, void *pReserved = NULL, int nWaitTime = 1000);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] nQueryType	Query type: When the connection status of device is obtained, nQueryType becomes NET_QUERY_GET_CAMERA_STATE.
	[in] pInBuf	Input cache: When the connection status of device is obtained, the corresponding structure is NET_IN_GET_CAMERA_STATEINFO.
	[out] pOutBuf	Output cache: When the connection status of device is obtained, the corresponding structure is NET_OUT_GET_CAMERA_STATEINFO.
	[in] pReserved	Reserved.
	[in] nWaitTime	Waiting time for query. The default is 1000ms.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.11.3 CLIENT_AttachCameraState

Item	Description	
Name	Subscribe the remote device status.	
Function	LLONG CLIENT_AttachCameraState(LLONG lLoginID, const NET_IN_CAMERASTATE* pstInParam, NET_OUT_CAMERASTATE *pstOutParam, int nWaitTime = 3000);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] pstInParam	Subscription input.
	[out] pstOutParam	Subscription output.
	[in] nWaitTime	Waiting time for query. The default is 3000ms.

Item	Description
Return value	● Success: Not 0. ● Failure: 0.
Note	For the state callback (fCameraStateCallBack) in the input parameter, see "Chapter 4 Callback Definition."

3.11.4 CLIENT_DetachCameraState

Item	Description	
Name	Stop subscribing the state of remote device.	
Function	BOOL CLIENT_DetachCameraState(LLONG lAttachHandle);	
Parameter	[in] lAttachHandle	Return value of subscribing remote device state.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.11.5 CLIENT_MatrixGetCameras

Item	Description	
Name	Get the remote device information.	
Function	BOOL CLIENT_MatrixGetCameras(LLONG lLoginID, const DH_IN_MATRIX_GET_CAMERAS* pInParam, DH_OUT_MATRIX_GET_CAMERAS* pOutParam, int nWaitTime = 1000);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] pInParam	Input parameter.
	[out] pOutParam	Output parameter.
	[in] nWaitTime	Waiting time for query. The default is 1000ms.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.11.6 CLIENT_QueryChannelName

Item	Description
Name	Get the channel name.

Item	Description	
Function	BOOL CLIENT_QueryChannelName(LLONG lLoginID, char *pChannelName, int maxlen, int *nChannelCount, int waittime=1000);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[out] pChannelName	Channel name buffer (In general, each channel name has 32 bytes. You need to prepare the buffer that is equal to or larger than 16*32).
	[in] maxlen	Buffer length. The unit is byte.
	[out] nChannelCount	Total channel numbers.
	[in] waittime	Waiting time for query. The default is 1000ms.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.11.7 CLIENT_GetNewDevConfig

Item	Description	
Name	Get the channel name.	
Function	BOOL CLIENT_GetNewDevConfig(LLONG lLoginID, char* szCommand, int nChannelID, char* szOutBuffer, DWORD dwOutBufferSize, Int *error, int waittime=500);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2.
	[in] szCommand	Command parameter to get the channel name. szCommand is CFG_CMD_VIDEOIN or CFG_CMD_CHANNELTITLE.
	[in] nChannelID	Device channel number starting from zero.
	[out] szOutBuffer	Output buffer.
	[out] dwOutBufferSize	Output buffer size.
	[out] error	Error code: 0: Success 1: Failure 2: Illegal data 3: Cannot be set for the moment 4: No authority
	[in] waittime	Waiting time for query. The default is 500ms.

Item	Description
Return value	● Success: TRUE. ● Failure: FALSE.
Note	None.

3.11.8 CLIENT_ParseData

Item	Description	
Name	Analyze data.	
Function	BOOL CLIENT_ParseData(char* szCommand, char* szInBuffer, LPVOID lpOutBuffer, DWORD dwOutBufferSize, void* pReserved);	
Parameter	[in] szCommand	Command parameter: Same with szCommand of the CLIENT_GetNewDevConfig.
	[in] szInBuffer	Input buffer: Same with szOutBuffer of CLIENT_GetNewDevConfig.
	[out] lpOutBuffer	Output buffer: Respectively corresponding to structure CFG_VIDEO_IN_INFO and AV_CFG_ChannelName of szCommand of CLIENT_GetNewDevConfig.
	[in] dwOutBufferSize	Output buffer size.
	[in] pReserved	Reserved.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

4 Callback Definition

4.1 fSearchDevicesCB

Item	Description	
Name	Callback of searching devices.	
Function	typedef void(CALLBACK *fSearchDevicesCB)(DEVICE_NET_INFO_EX * pDevNetInfo, void* pUserData);	
Parameter	[out]pDevNetInfo	The searched device information.
	[out]pUserData	User data.
Return value	None.	
Note	None.	

4.2 fDisConnect

Item	Description	
Name	Disconnection callback.	
Function	typedef void (CALLBACK *fDisConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
Parameter	[out] lLoginID	Return value of CLIENT_LoginEx2.
	[out] pchDVRIP	IP of the disconnected device.
	[out] nDVRPort	Port of the disconnected device.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.3 fHaveReConnect

Item	Description
Name	Reconnection callback.

Item	Description	
Function	typedef void (CALLBACK *fHaveReConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
Parameter	[out] lLoginID	Return value of CLIENT_LoginEx2.
	[out] pchDVRIP	IP of the reconnected device.
	[out] nDVRPort	Port of the reconnected device.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.4 fRealDataCallBackEx2

Item	Description	
Name	Callback of real-time monitoring data.	
Function	typedef void (CALLBACK * fRealDataCallBackEx2)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LLONG param, LDWORD dwUser);	
Parameter	[out] lRealHandle	Return value of CLIENT_RealPlayEx.
	[out] dwDataType	Data type: ● 0: Initial data. ● 1: Data with frame information. ● 2: YUV data. ● 3: PCM audio data.
	[out] pBuffer	Address of monitoring data block.
	[out] dwBufSize	Length (unit: byte) of the monitoring data block.
	[out] param	Callback parameter structure. Different dwDataType value corresponds to different type. ● The param is blank pointer when dwDataType is 0. ● The param is the pointer of tagVideoFrameParam structure when dwDataType is 1. ● The param is the pointer of tagCBYUVDataParam structure when dwDataType is 2. ● The param is the pointer of

Item	Description	
		tagCBPCMDataParam structure when dwDataType is 3.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.5 pfAudioDataCallBack

Item	Description	
Name	Callback of audio data of voice talk.	
Function	typedef void (CALLBACK *pfAudioDataCallBack)(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag, LDWORD dwUser);	
Parameter	[out] lTalkHandle	Return value of CLIENT_StartTalkEx.
	[out] pDataBuf	Address of audio data block.
	[out] dwBufSize	Length of the audio data block. The unit is byte.
	[out] byAudioFlag	Data type: ● 0: Local collecting. ● 1: Device sending.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.6 fDownLoadPosCallBack

Item	Description	
Name	Callback of playback and download by file.	
Function	typedef void (CALLBACK *fDownLoadPosCallBack)(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);	
Parameter	[out]lPlayHandle	Return value of playback or download.
	[out]dwTotalSize	Total size. The unit is KB.
	[out]dwDownLoadSize	The downloaded size. The unit is KB ● -1: Current playback stopped.

Item	Description	
		● -2: Write file failed.
	[out]dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.7 fDataCallBack

Item	Description	
Name	Callback of playback and download data.	
Function	typedef int (CALLBACK *fDataCallBack)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);	
Parameter	[out]lPlayHandle	Return value of playback or download interface.
	[out] dwDataType	0 (original data).
	[out] pBuffer	Data buffer.
	[out] dwBufSize	Buffer length. The unit is byte.
	[out] dwUser	User parameter of the callback.
Return value	● 0: This callback failed and the next callback will return the same data. ● 1: This callback succeeded and the next callback will return the subsequent data.	
Note	● Set this callback in the record playback interface such as CLIENT_PlayBackByTimeEx. ● If the parameter hWnd is not NULL during setting the callback, the callback is considered to be succeeded no matter which value is returned, and the next callback will return the subsequent data. ● The user can identify through parameter lRealHandle the corresponding callback data of which stream decoding.	

4.8 fTimeDownLoadPosCallBack

Item	Description
Name	Callback of download by time.
Function	typedef void (CALLBACK *fTimeDownLoadPosCallBack)(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, int index,

Item	Description	
	NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser);	
Parameter	[out]lPlayHandle	Return value of download interface.
	[out] dwTotalSize	Total size of playback. The unit is KB.
	[out]dwDownLoadSize	The size that has been played. The unit is KB. ● -1: Current download finished. ● -2: Write file failed.
	[out] index	Index.
	[out] recordfileinfo	Record file information.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.9 fMessCallBack

Item	Description	
Name	Alarm callback.	
Function	BOOL (CALLBACK *fMessCallBack)(LONG lCommand, LLONG lLoginID, char *pBuf, DWORD dwBufLen, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
Parameter	[out] lCommand	Alarm type. For details, see Table 4-1.
	[out] lLoginID	Return value of login interface.
	[out] pBuf	Receives the buffer of alarm data. The entered data is different dependent on the listen data and value of lCommand.
	[out] dwBufLen	Length of pBuf. The unit is byte.
	[out] pchDVRIP	Device IP.
	[out] nDVRPort	Port.
	[out] dwUser	User parameter of the callback.
Return value	Void.	
Note	In general, set the callback when initializing. Provide the different treatment to callback dependent on the device ID and command value.	

For the information about alarm type, see Table 4-1.

Alarm type	Description	pBuf

Alarm type	Description	pBuf
DH_MOTION_ALARM_EX	Alarm by automatic detection	The number of data byte is the same with video channel number. Each byte represents the alarm state by dynamic detection of a video channel. ● 1: Alarm. ● 0: No alarm.
DH_ALARM_STORAGE_FAILURE	Alarm by hard disk damage	ALARM_STORAGE_FAILURE data group.
EVENT_ALARM_VIDEOLOSS	Alarm by video loss	None.
DH_ALARM_FRONTDISCONNECT	Alarm by IPC disconnection	ALARM_FRONTDISCONNET_INFO.
DH_ALARM_ALARM_EX	External alarm	The number of data byte is the same with video channel number. Each byte represents the alarm state of alarm channel. ● 1: Alarm. ● 0: No alarm.

Table 4-1

4.10 fCameraStateCallBack

Item	Description	
Name	Callback of remote device state.	
Function	void (CALLBACK *fCameraStateCallBack) (LLONG lLoginID, LLONG lAttachHandle, const NET_CB_CAMERASTATE *pBuf, int nBufLen, LDWORD dwUser);	
Parameter	[out] lLoginID	Return value of login interface.
	[out] lAttachHandle	Return value of subscription.
	[out] pBuf	State of front-end device.
	[out] nBufLen	The length of returned data.
	[out] dwUser	User parameter of the callback.
Note	After subscribing the remote device state, if the state of front-end device changes, the information of changed device will be reported.	