NetSDK Programming Manual
(Intelligent Building)

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 intelligent buildings. 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

- Software development engineers, product managers and project managers who use SDK.

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 appearing in the Manual to help you understand the function of each module.

Term	Definition
Protection zone	The alarm input channel can receive the externally detected signal and each becomes a protection zone.
Armed and disarmed	● Armed: The armed area receives, processes, records and transfers the external signals. ● Disarmed: The disarmed area does not receive, process, record and transfer the external signals.
Bypass	When the device is in armed status, the protection zone can still monitor and record the external detector but will not forward to the user. After the device is disarmed, the protection zone of bypass will turn to a normal status, and when it is armed again, it can switch to a protection zone successfully.
Alarm clearing	When the device generates alarm, it will perform some linkage activities, such as buzzer and message. These activities usually last a period. Alarm clearing can stop them ahead of time.
Real-time protection zone	When the device is in armed status, if there is an alarm, the device will record and forward alarm signals immediately.
Time-delay protection zone	When the protection zone is of time-delayed type, you can set the entrance delay or exit delay. ● Entrance delay: The alarm will be activated when user enters the protection zone within the delayed period, but there will be not alarm linkage. After the delayed period, if the protection zone is still armed, there will be alarm linage activated, if disarmed, there will be no alarm linkage. ● Exit delay: The device will enter the armed status after the end of exit delay.
24 hour protection zone	Once the 24 hour protection zone has been configured, the setting gets effective immediately. You cannot arm or disarm this setting so it is applicable to fire alarm scenarios.
Scenario mode	The alarm host has two scenario modes: "Outside" and "Home". Each of the modes has relevant configurations which get effective after you selected.
Outside/Home	When the scenarios switch to "Outside" or "Home", the planned protection zone will be armed and the others become bypass zones.
Separation	A kind of configuration to the intrusion alarm detecting circuit which cannot report alarms till being reset manually.

Term	Definition
Analog alarm channel (analog protection zone)	The device has multiple alarm input channels to receive the external detection signals. When the channels are analog type, they are called analog alarm channels which can connect to analog detector and collect analog data.
Stress card	A type of access card. When the user is forced to open the access, the stress card will be recognized by the system, and then the alarm will be generated.

Table of Contents

1 Overview

1.1 General

The Manual introduces SDK interfaces that include 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, voice 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

Library type	Library file name	Library file description
Dependent library of "avnetsdk.dll"	Infra.dll	Infrastructure library
	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
Dependent library of "libavnetsdk.so"	libInfra.so	Infrastructure library
	libJson.so	Network infrastructure library
	libNetFramework.so	Media transmission structure package library
	libStream.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

1.2.1 Supported System

- 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

1.2.2 Supported Devices

- Access Control

DH-BSC1221A, DH-BSC1201B, DH-BSC1201C, DH-BSC1201C-S, DH-BSC1204B, DH-BSC1204C, DH-BSC1204C-S, DH-BSR1100A, DH-BSR1101A, DH-BSR1100B, DH-BSR1102A, DH-BSM100B, DH-BSM102A, AL-280(LED) , AL-500(LED) , AL-280L, AL-280PU, AL-280Z, AL-500L, AL-500PU, AL-500Z, AL-100, FS-078, DH-SL5101CG, DH-SL5101CR, DH-SL5101CS, DH-SL5102RR, DH-SL5102RS, DH-SL5102RG

- Video Intercom

VTO1210B-X, VTO1220B, VTO1210C-X, VTO9231D, VTO9241D, VTO6210B, VTO2000A, VTO6100C, VTO2111D, VTH1550CH, VTH1510CH, VTH1510A, VTH5221D, VTH5241D, VTT201, VTT2610C, VTA8111A, VTS8240B, VTS8420B, VTS5420B, VTS1500A

- Alarm Host

ARC2008C, ARC2008C-G, DH-ARC2008C, DH-ARC2008C-G, DHI-ARC2008C, DHI-ARC2008C-G, OEM-ARC2008C, OEM-ARC2008C-G, ARC2016C, ARC2016C-G, ARC2016C-G, DH-ARC2016C, DH-ARC2016C-G, DHI-ARC2016C, DHI-ARC2016C-G, OEM-ARC2016C, OEM-ARC2016C-G, ARC9016C, ARC9016C-G,DH-ARC9016C, DH-ARC9016C-, DHI-ARC9016C, DHI-ARC9016C-G, OEM-ARC9016C, OEM-ARC9016C-G, ARC5808C-C, ARC5808C, DH-ARC5808C-E, DH-ARC5808C-C, DH-ARC5808C, DHI-ARC5808C, DHI-ARC5808C-C, OEM-ARC5808C-E, OEM-ARC5808C-C, OEM-ARC5808C, ARC5408C-C, ARC5408C, DH-ARC5408C-E, DH-ARC5408C-C, DH-ARC5408C, DHI-ARC5408C, DHI-ARC5408C-C, OEM-ARC5408C-E, OEM-ARC5408C-C, OEM-ARC5408C

1.3 Application Scenario

- Typical Scenario.

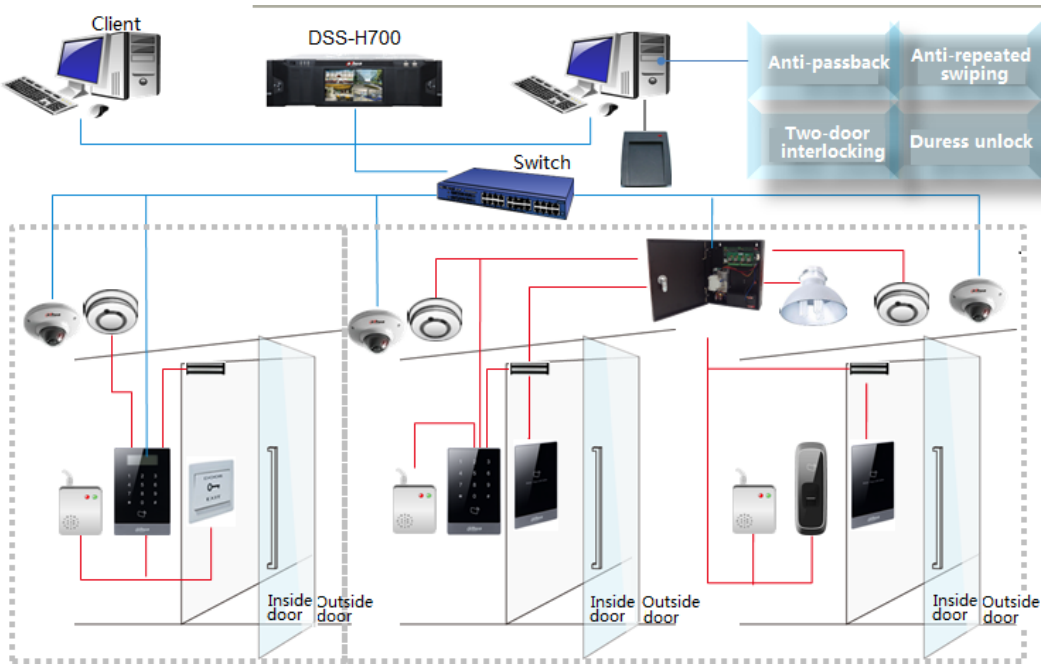

Figure 1-1

- Micro access control for small-sized office.

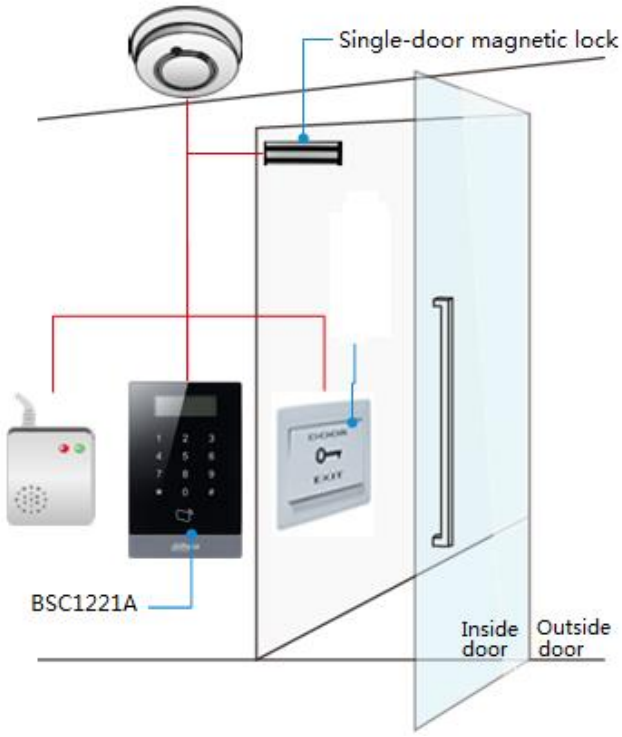

Figure 1-2

- Network access control for middle and small-sized intelligent building, treasury house and

jail monitoring projects.

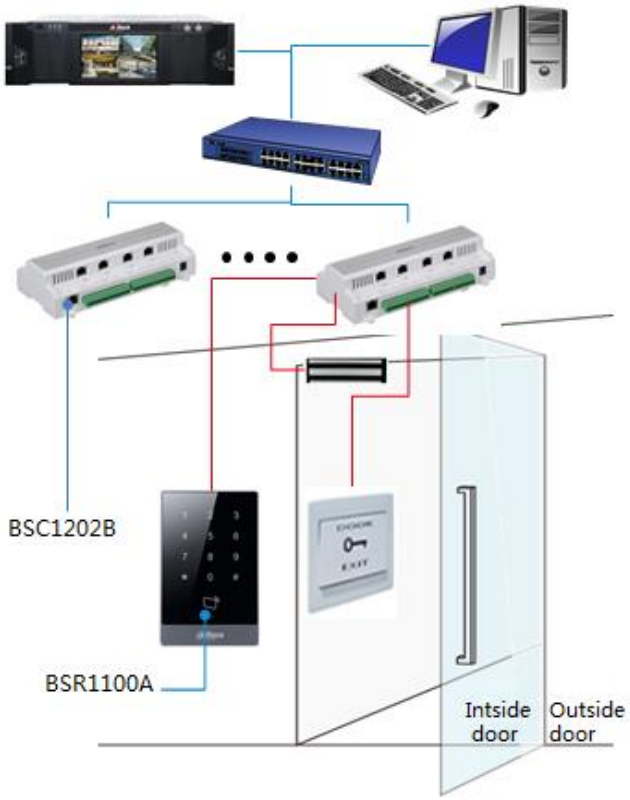

Figure 1-3

- Enhanced access control.

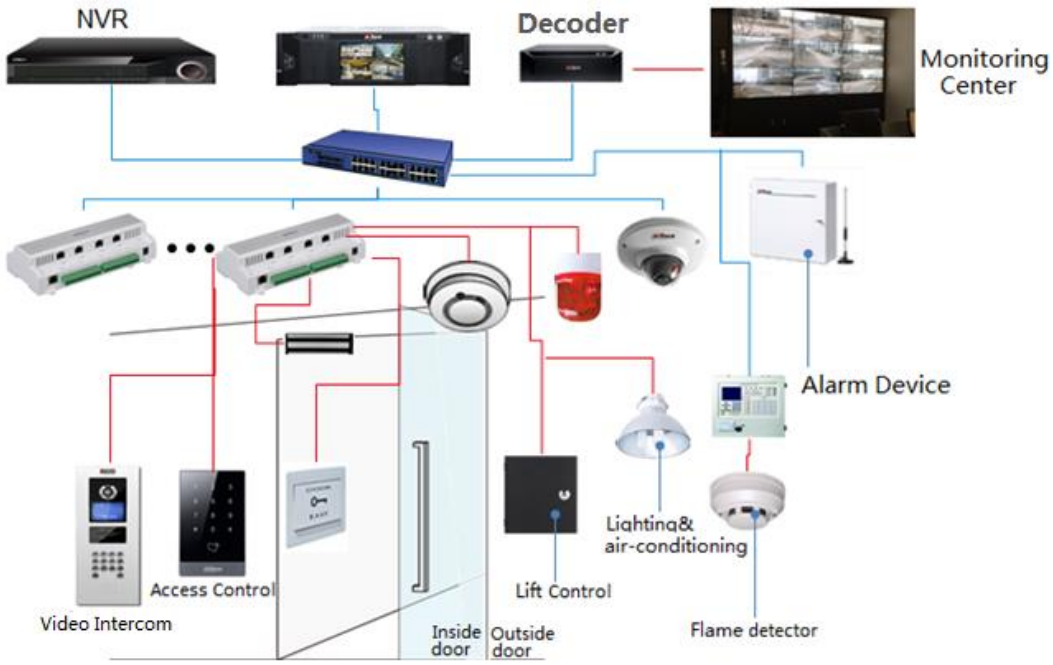

Figure 1-4

- Intelligent access control for high-end office building, government building and villa.

Figure 1-5

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.

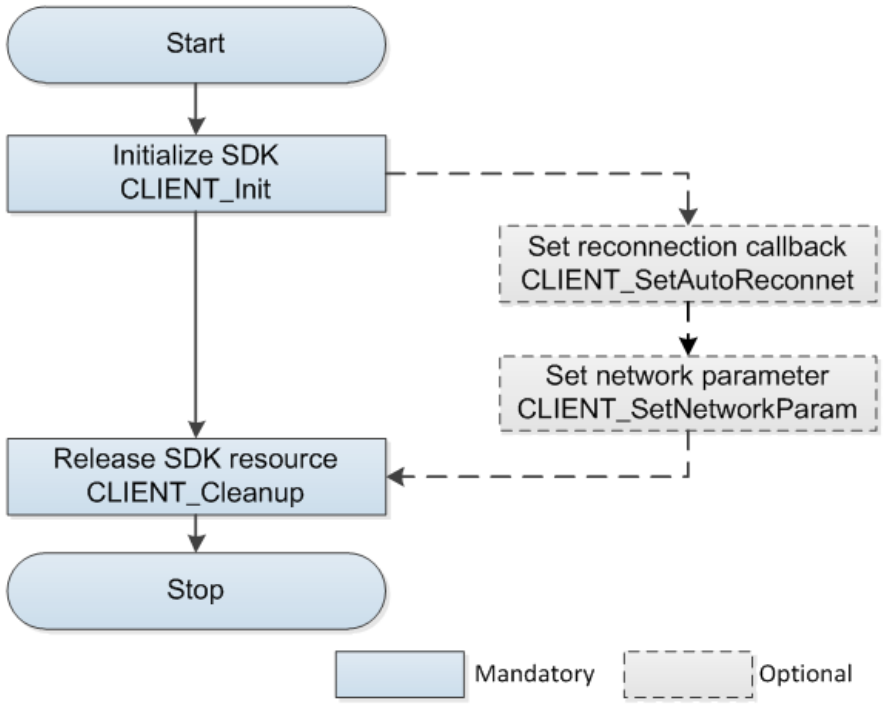

Figure 2-1

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 ths 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.

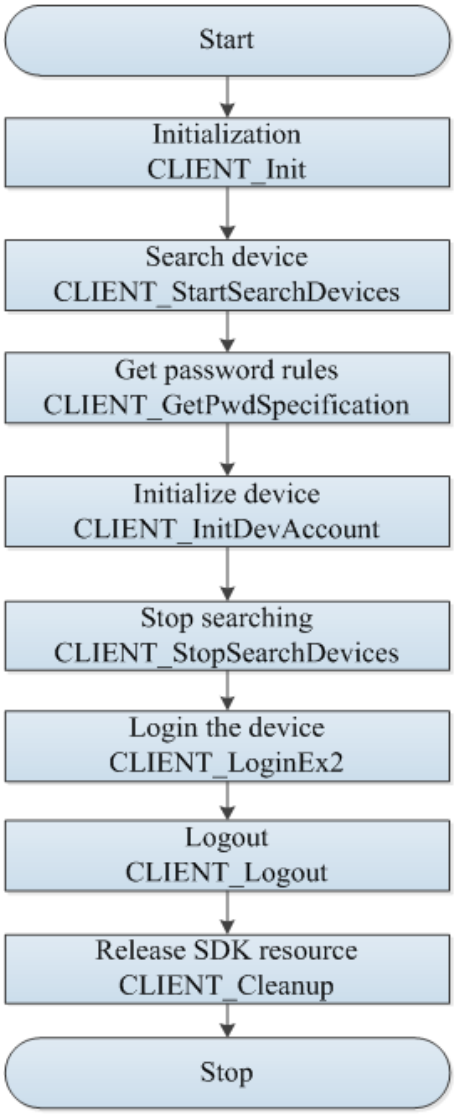

Figure 2-2

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.

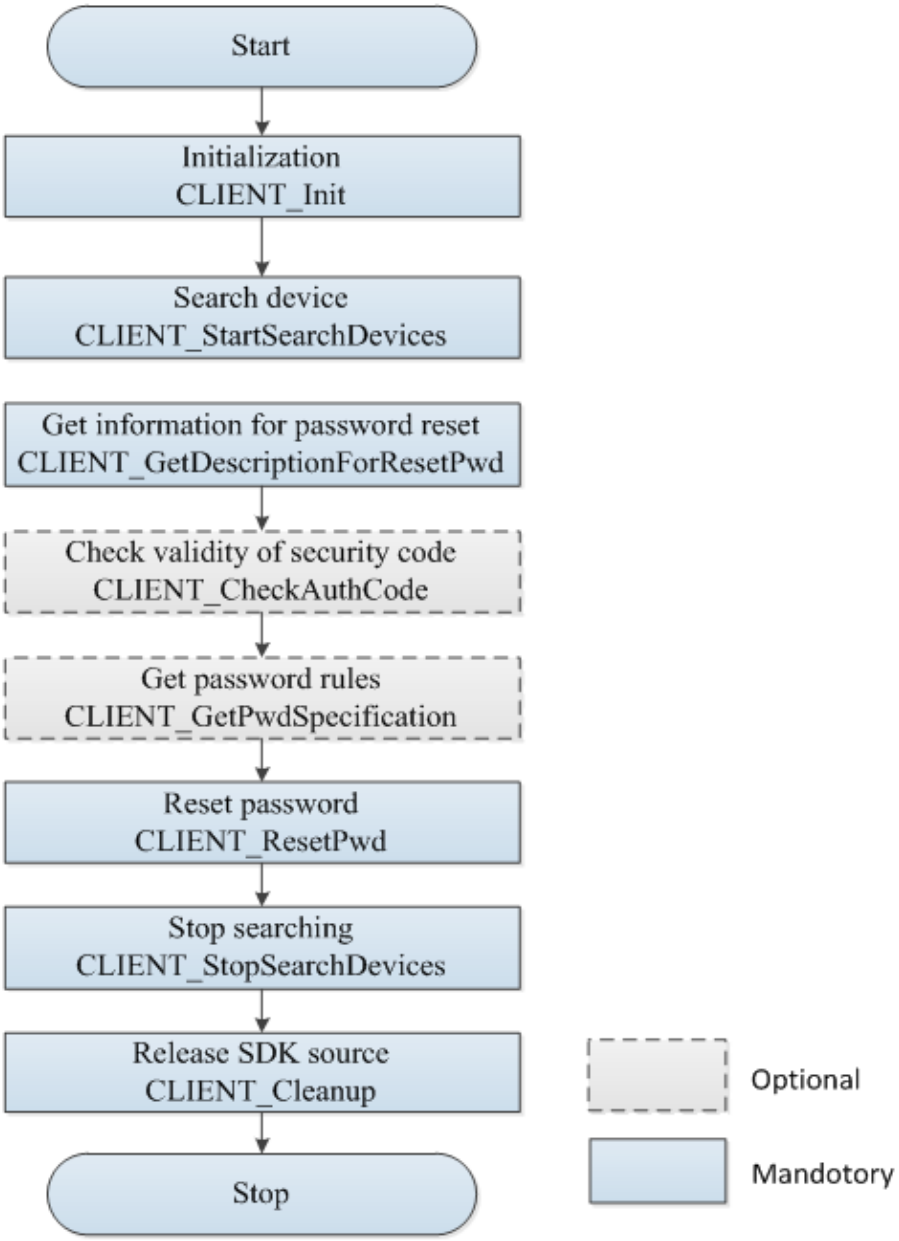

Figure 2-3

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.

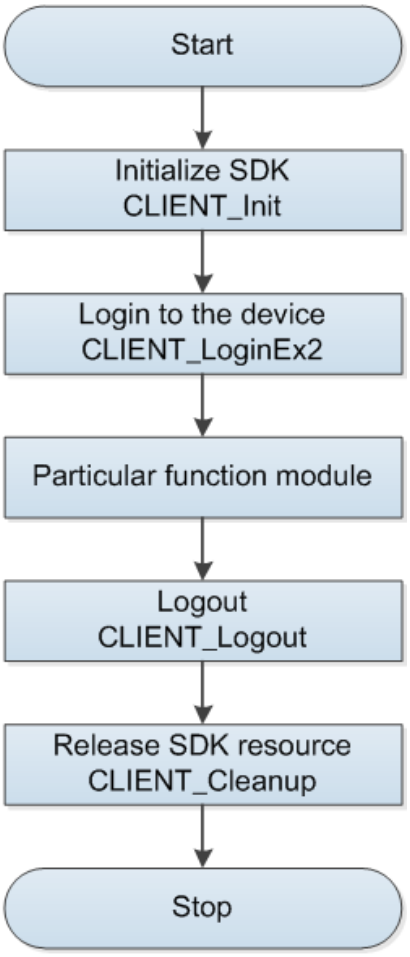

Figure 2-4

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 through saving the callback stream or calling the SDK interface.

2.4.2 Interface Overview

Interface	Implication
CLIENT_RealPlayEx	Start real-time monitoring
CLIENT_StopRealPlayEx	Stop real-time monitoring
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

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.

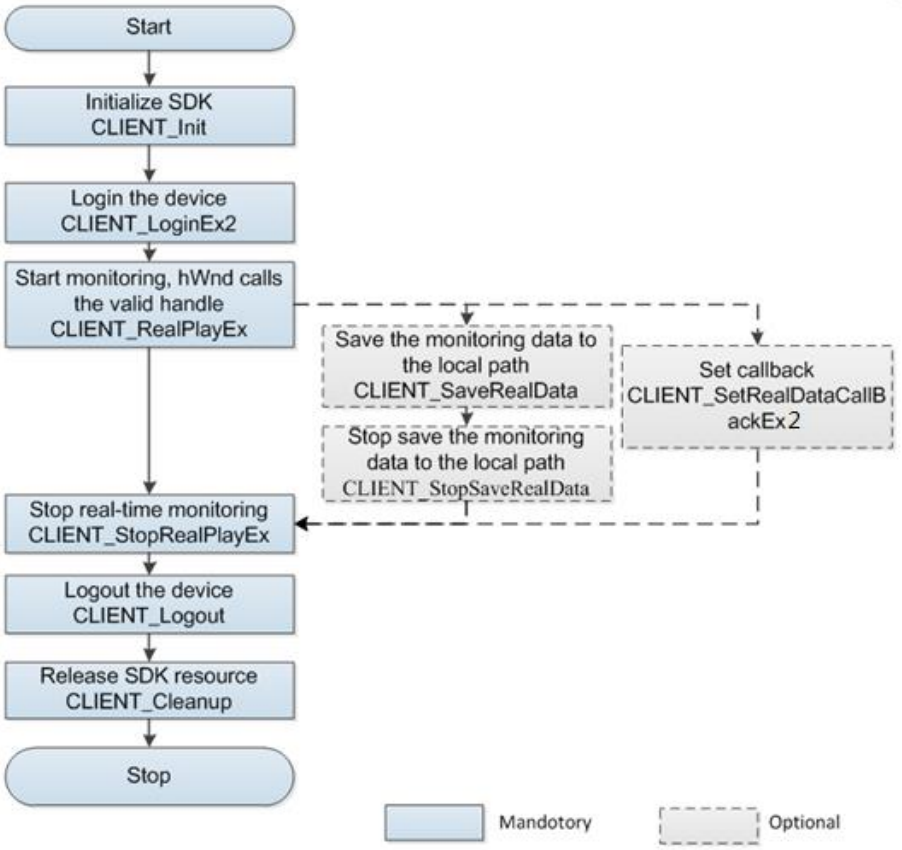

Figure 2-5

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: Because some devices do not support opening the monitoring function on the same channel for multiple times, these devices might fail from the second opening. 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 private library. See "The 3rd Party Decoding Library."

2.4.3.2 The 3rd Party Decoding Library

SDK calls back the real-time monitoring stream to you and then you call PlaySDK to perform decoding playing.

For the process of calling the 3rd party decoding, see Figure 2-6.

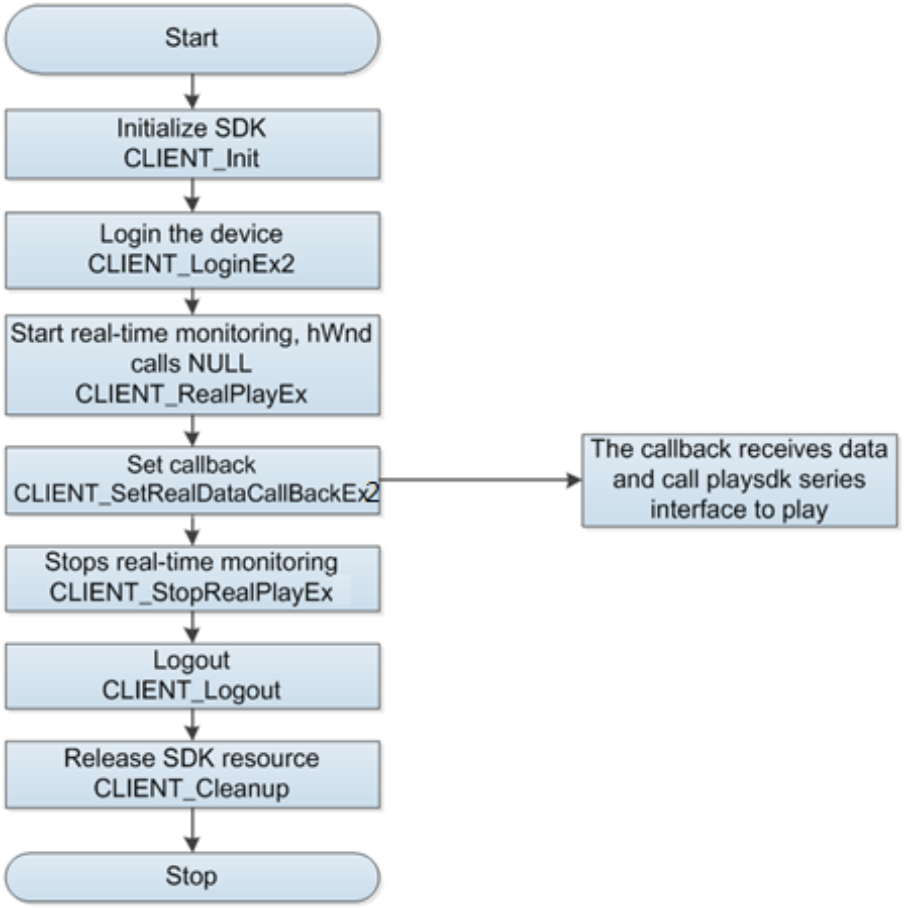

Figure 2-6

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; //Original data label
	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)
{
    //To get the stream data from the device, you need to call the interface of PlaySDK. For details, see SDK
monitoring demo.
    printf("receive real data, param: lRealHandle[%p], dwDataType[%d], pBuffer[%p], dwBufSize[%d]\n",
lRealHandle, dwDataType, pBuffer, dwBufSize);
}
```

2.5 Voice Talk

2.5.1 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.

2.5.2 Interface Overview

Interface	Implication
CLIENT_GetDevProtocolType	Obtain the talk type supported by the device
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)
CLIENT_SetDeviceMode	Set audio talk mode

Table 2-6

2.5.3 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 2-7.

NOTE

The voice talk mode is divided into 2nd generation and 3rd generation which share the same process and differentiated by the parameters set by CLIENT_SetDeviceMode. You can use CLIENT_GetDevProtocolType to get the voice talk mode supported by the device.

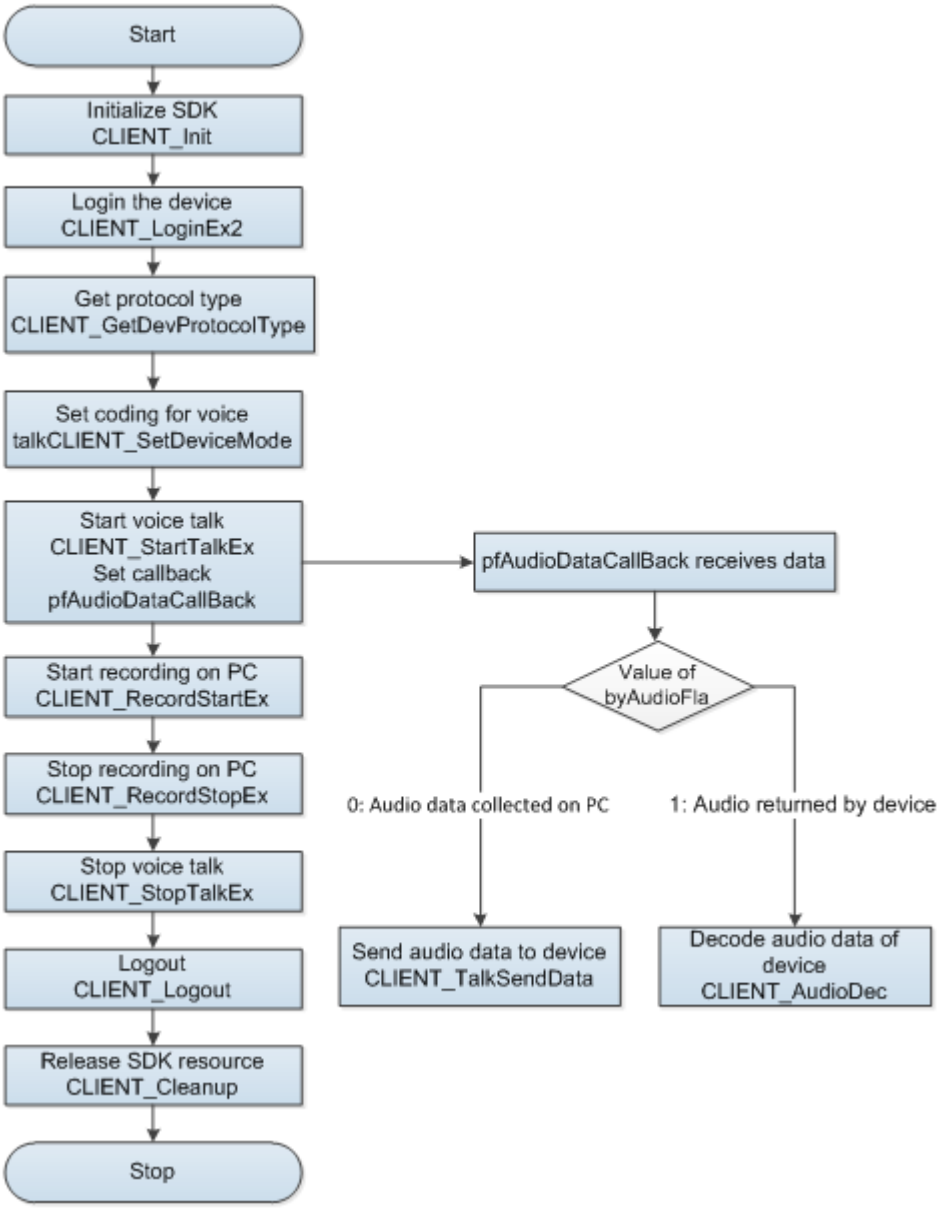

Figure 2-7

Process Description

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

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

Step 3 Call **CLIENT_GetDevProtocolType** to get protocol for the 2nd generation or the 3rd generation of voice talk.

Step 4 Call **CLIENT_SetDeviceMode** to set decoding information of voice talk.

- For the 2nd generation: Set encoding mode, client mode and speak mode. The **emType** is set as **DH_TALK_ENCODE_TYPE**，**DH_TALK_CLIENT_MODE** and **DH_TALK_SPEAK_PARAM**.

- For the 3rd generation: Set encoding mode, client mode and speak mode. The **emType** is set as **DH_TALK_ENCODE_TYPE, DH_TALK_CLIENT_MODE** and **DH_TALK_MODE3**.

Step 5 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 6 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 7 After using the voice talk function, call **CLIENT_RecordStopEx** to stop recording.

Step 8 Call **CLIENT_StopTalkEx** to stop voice talk.

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

- Voice encoding format: The example uses the common PCM format. SDK supports accessing the voice encoding format supported by the device. The example code is detailed in 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.

2.5.4 Example Code

```
// Get to know the device supports the 2nd generation or the 3rd generation of voice talk.
EM_DEV_PROTOCOL_TYPE emTpye = EM_DEV_PROTOCOL_UNKNOWN;
CLIENT_GetDevProtocolType(g_lLoginHandle, &emTpye);

DHDEV_TALKDECODE_INFO curTalkMode = {0};
curTalkMode.encodeType = DH_TALK_PCM;
curTalkMode.nAudioBit = 16;
curTalkMode.dwSampleRate = 8000;
curTalkMode.nPacketPeriod = 25;
CLIENT_SetDeviceMode(lLoginHandle, DH_TALK_ENCODE_TYPE, &curTalkMode); //Set encoding mode for voice talk
CLIENT_SetDeviceMode(lLoginHandle, DH_TALK_CLIENT_MODE, NULL);//Set voice talk at client mode

//Set voice talk parameters according to protocol type
if (emTpye == EM_DEV_PROTOCOL_V3) //This parameter is only required by the 3rd generation
{
    NET_TALK_EX stuTalk = {sizeof(stuTalk)};
        stuTalk.nAudioPort = RECEIVER_AUDIO_PORT;     //User customized receive port
```

```
            stuTalk.nChannel    = 0;
            stuTalk.nWaitTime = 5000;
            CLIENT_SetDeviceMode(m_lLoginHandle, DH_TALK_MODE3, &stuTalk)
}
// Start voice talk
lTalkHandle = CLIENT_StartTalkEx(lLoginHandle, AudioDataCallBack, (LDWORD)NULL);
// Start local recording
CLIENT_RecordStartEx(lLoginHandle);
// Stop local recording
CLIENT_RecordStopEx(lLoginHandle)
// Stop voice talk
CLIENT_StopTalkEx(lTalkHandle);
// Process the voice talk callback
void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE
byAudioFlag, LDWORD dwUser)
{
if(0 == byAudioFlag)
      {
            // Send the audio data detected by local PC to devic
            CLIENT_TalkSendData(lTalkHandle, pDataBuf, dwBufSize);
      }
else if(1 == byAudioFlag)
      {
            // Send the audio data sent from device to SDK for decoding playing
            CLIENT_AudioDec(pDataBuf, dwBufSize);
      }
}
```

2.6 Alarm Event

2.6.1 Introduction

Alarm report can be realized through the following way: Use SDK to login the device and subscribe alarm function from the device. When the device detects the alarm event, it will send to SDK immediately. The user can get the corresponding alarm information through callback.

2.6.2 Interface Overview

Interface	Implication

Interface	Implication
CLIENT_SetDVRMessCallBack	Set alarm callback.
CLIENT_StartListenEx	Subscribe alarm.
CLIENT_StopListen	Stop alarm subscription.

Table 2-7

2.6.3 Process

For the process of alarm report, see Figure 2-8.

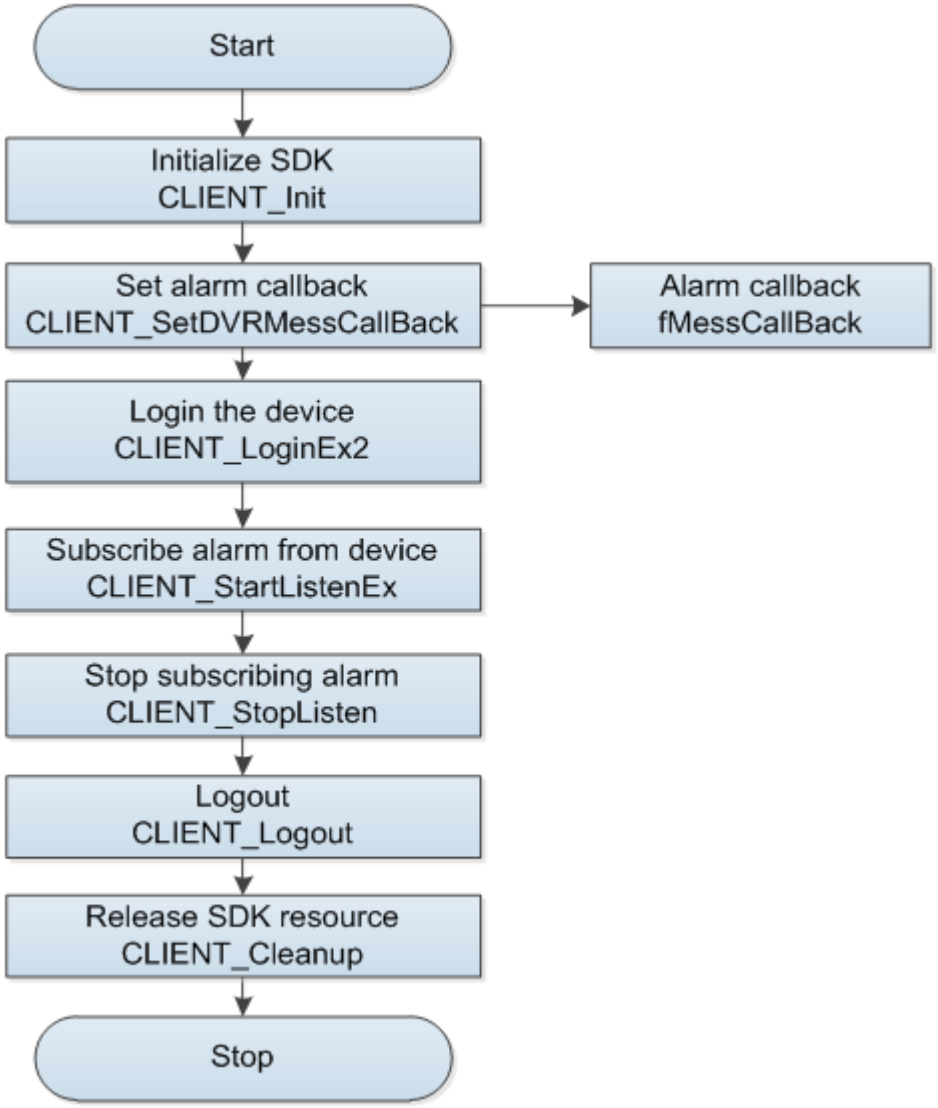

Figure 2-8

Process Description

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

Step 2 Call **CLIENT_SetDVRMessCallBack** to set alarm callback.

NOTE

This interface should be called before subscribing alarms.

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

Step 4 Call **CLIENT_StartListenEx** to subscribe alarms from the device. After successful

subscription, use the callback set by CLIENT_SetDVRMessCallBack to inform the user of the alarm events reported by the device.

NOTE

For the alarm events about alarm host, access and voice talk, see "6.6fMessCallBack."

Step 5 Call **CLIENT_StopListen** to stop subscribing alarms from 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 alarm events that were reported before are no longer reported, check if the device is disconnected. If yes, please be noted that there will be no alarm reported after the device is reconnected, and in this cause you need to cancel the subscription and then subscribe again.
- It is recommended to handle the alarm information of the callback fMessCallBack in somewhere else to avoid blocking the callback operations.

2.6.4 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_ACCESS_CTL_EVENT) // Event about access. For more events
about access, see "6.6 fMessCallBack."
    {
        ALARM_ACCESS_CTL_EVENT_INFO* pstAccessInfo =
                                        (ALARM_ACCESS_CTL_EVENT_INFO*)pBuf;
         // Then you can get the corresponding alarm information through pstAccessInfo.
    }
    ……..
}
// Set alarm callback
CLIENT_SetDVRMessCallBack(afMessCallBack,0);
// Subscribe alarm
CLIENT_StartListenEx(lLoginHandle);
// Stop alarm subscription
CLIENT_StopListen(lLoginHandle);
```

3.1 Armed and Disarmed Status

3.1.1 Introduction

- Armed: All the protection zones are in armed status and can receive, process, record and transfer external signals.
- Disarmed: All the protection zones do not receive, process, record and transfer external signals.

3.1.2 Interface Overview

Interface	Implication
CLIENT_ControlDeviceEx	Device control.

Table 3-1

3.1.3 Process

For the process of setting the armed and disarmed status, see Figure 3-1.

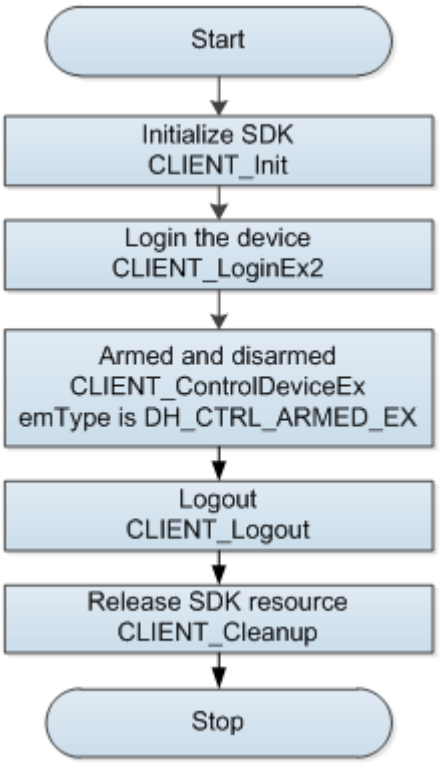

Figure 3-1

Process Description

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

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

Step 3 Call **CLIENT_ControlDeviceEx** to control the device to be armed or disarmed. The **emType** value is **DH_CTRL_ARMED_EX**.

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.

3.1.4 Example Code

```
CTRL_ARM_DISARM_PARAM_EX stuParam = {sizeof(stuParam)};
stuParam.stuIn.dwSize = sizeof(stuParam.stuIn);
stuParam.stuOut.dwSize = sizeof(stuParam.stuOut);

stuParam.stuIn.emState = NET_ALARM_MODE_ARMING;
stuParam.stuIn.emSceneMode = NET_SCENE_MODE_OUTDOOR;
stuParam.stuIn.szDevPwd = "admin";
CLIENT_ControlDeviceEx(g_lLoginHandle, DH_CTRL_ARMED_EX, &stuParam, NULL,3000);
```

3.2 Protection Zone Status Setting

3.2.1 Introduction

You can set the protection zone status to control the normal, bypass and separation status of alarm channels.

3.2.2 Interface Overview

Interface	Implication
CLIENT_ControlDeviceEx	Device control.

Table 3-2

3.2.3 Process

For the process of setting the protection zone status, see Figure 3-2.

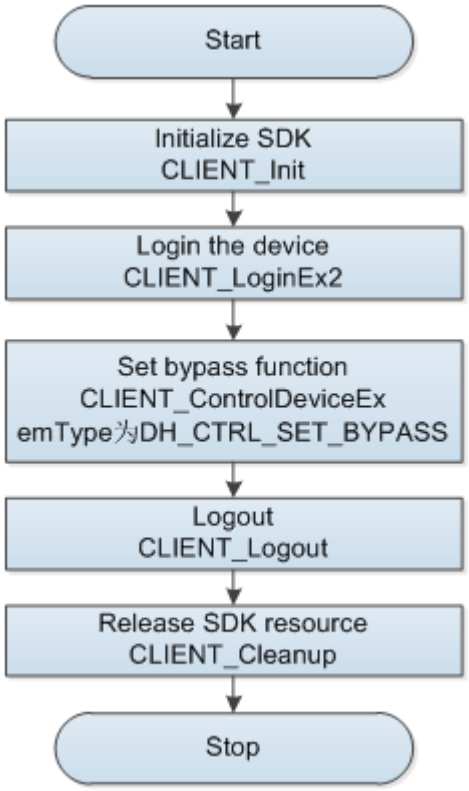

Figure 3-2

Process Description

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

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

Step 3 Call **CLIENT_ControlDeviceEx** to control the device to set the protection zone status. The **emType** value is **DH_CTRL_SET_BYPASS**.

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.

3.2.4 Example Code

```
NET_CTRL_SET_BYPASS stuParam = {sizeof(stuParam)};
stuParam.dwSize = sizeof(stuParam);
stuParam.emMode = NET_BYPASS_MODE_BYPASS; // Set the protection zone status as bypass
stuParam.szDevPwd = "admin";
stuParam.nExtendedCount = 1;
int nExtendChn[1] = {1};
stuParam.pnExtended = nExtendChn;
stuParam.nLocalCount = 2;
int nLocalChn[2] = {2,3};
stuParam.pnLocal = nLocalChn;
CLIENT_ControlDeviceEx(g_lLoginHandle, DH_CTRL_SET_BYPASS, &stuParam, NULL, 3000);
```

3.3 Protection Zone Status Query

3.3.1 Introduction

Check the protection zone status, including alarm input, alarm output, and alarm signal.

3.3.2 Interface Overview

Interface	Implication
CLIENT_QueryDevState	Status query.

Table 3-3

3.3.3 Process

For the process of querying protection zone status, see Figure 3-3.

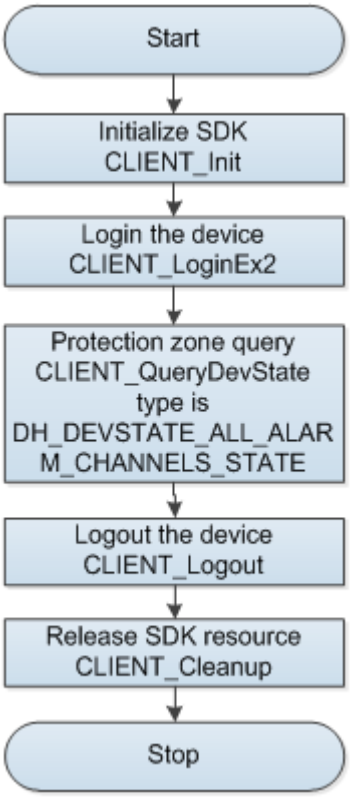

Figure 3-3

Process Description

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

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

Step 3 Call **CLIENT_QueryDevState** to query the protection zone status. The **type** value is **DH_DEVSTATE_ALL_ALARM_CHANNELS_STATE**.

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.

3.3.4 Example Code

```
NET_CLIENT_ALARM_CHANNELS_STATE stuAlarmChannelState = {sizeof(stuAlarmChannelState)};
stuAlarmChannelState.emType = NET_ALARM_CHANNEL_TYPE_ALL; // Query all the channels status;
int nNum = 2;

// Initialize the filed relevant to alarm signals
stuAlarmChannelState.nAlarmBellCount = nNum;
stuAlarmChannelState.pbAlarmBellState = new BOOL[stuAlarmChannelState.nAlarmBellCount];
memset(stuAlarmChannelState.pbAlarmBellState, 0, stuAlarmChannelState.nAlarmBellCount * sizeof(BOOL));

// Initialize the filed relevant to alarm input
stuAlarmChannelState.nAlarmInCount = nNum;
stuAlarmChannelState.pbAlarmInState = new BOOL[stuAlarmChannelState.nAlarmInCount];
memset(stuAlarmChannelState.pbAlarmInState, 0, stuAlarmChannelState.nAlarmInCount * sizeof(BOOL));

// Initialize the filed relevant to alarm output
stuAlarmChannelState.nAlarmOutCount = nNum;
stuAlarmChannelState.pbAlarmOutState = new BOOL[stuAlarmChannelState.nAlarmOutCount];
memset(stuAlarmChannelState.pbAlarmOutState, 0, stuAlarmChannelState.nAlarmOutCount * sizeof(BOOL));

// Initialize the filed relevant to alarm input of expanded module
stuAlarmChannelState.nExAlarmInCount = nNum;
stuAlarmChannelState.pbExAlarmInState = new BOOL[stuAlarmChannelState.nExAlarmInCount];
memset(stuAlarmChannelState.pbExAlarmInState, 0, stuAlarmChannelState.nExAlarmInCount * sizeof(BOOL));
stuAlarmChannelState.pnExAlarmInDestionation = new int[1024];

// Initialize the filed relevant to alarm output of expanded module
stuAlarmChannelState.nExAlarmOutCount = nNum;
stuAlarmChannelState.pbExAlarmOutState = new BOOL[stuAlarmChannelState.nExAlarmOutCount];
memset(stuAlarmChannelState.pbExAlarmOutState, 0, stuAlarmChannelState.nExAlarmOutCount *
sizeof(BOOL));
stuAlarmChannelState.pnExAlarmOutDestionation = new int[1024];

int nRetLen = 0;
CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_ALL_ALARM_CHANNELS_STATE,
(char*)&stuAlarmChannelState, sizeof(NET_CLIENT_ALARM_CHANNELS_STATE), &nRetLen, 3000);
```

4.1 Access Control

4.1.1 Introduction

The access control manages opening and closing of the access.

4.1.2 Interface Overview

Interface	Implication
CLIENT_ControlDeviceEx	Device control.

Table 4-1

4.1.3 Process

For the process of access control, see Figure 4-1.

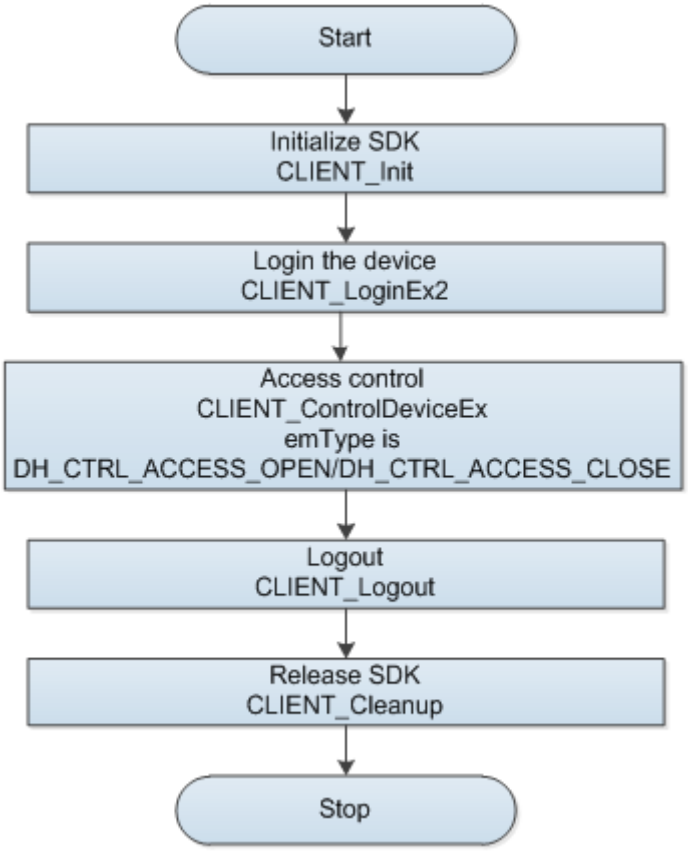

Figure 4-1

Process Description

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

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

Step 3 Call **CLIENT_ControlDeviceEx** to control access

- Opening access: The **emType** value is **DH_CTRL_ACCESS_OPEN**.
- Closing access: The **emType** value is **DH_CTRL_ACCESS_CLOSE**.

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.

4.1.4 Example Code

```
// Open access
NET_CTRL_ACCESS_OPEN stOpen = {sizeof(stOpen)};
stOpen.nChannelID = 0;
strncpy(stOpen.szUserID, “admin”, sizeof(stOpen.szUserID) - 1);
CLIENT_ControlDeviceEx((LLONG)g_lLoginHandle, DH_CTRL_ACCESS_OPEN, &stOpen, NULL, 3000);

// Close access
NET_CTRL_ACCESS_CLOSE stClose = {sizeof(stClose)};
CLIENT_ControlDeviceEx((LLONG)g_lLoginHandle, DH_CTRL_ACCESS_CLOSE, &stClose, NULL, 3000);
```

5 Interface Definition

5.1 SDK Initialization

5.1.1 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.	

5.1.2 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.

5.1.3 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.

5.1.4 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.	

5.2 Device Initialization

5.2.1 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	

5.2.2 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	

5.2.3 CLIENT_GetDescriptionForResetPwd

Item	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	

5.2.4 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	

5.2.5 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	

5.2.6 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	

5.2.7 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	

5.3 Device Login

5.3.1 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 5-1

5.3.2 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	

5.4 Real-time Monitoring

5.4.1 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 5-2

5.4.2 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	

5.4.3 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	

5.4.4 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

5.4.5 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 type and meaning about parameter dwFlag:

dwFlag	**Meaning**
REALDATA_FLAG_RAW_DATA	Original data of device
REALDATA_FLAG_DATA_WITH_FRAME_INFO	Data label with frame information
REALDATA_FLAG_YUV_DATA	YUV data label
REALDATA_FLAG_PCM_AUDIO_DATA	PCM data label

Table 5-3

5.5 Device Control

5.5.1 CLIENT_ControlDeviceEx

Item	Description
Name	Device control

Item	Description	
Function	BOOL CLIENT_ControlDeviceEx(LLONG lLoginID, CtrlType emType, Void *pInBuf, Void *pOutBuf = NULL, int nWaitTime = 1000);	
Parameter	[in] lLoginID	The return value of CLIENT_LoginEx2
	[in] emType	Control type
	[in] pInBuf	Input parameter that is different depending on emType
	[out] pOutBuf	Output parameter that is NULL as default. For some emType, there are output structures.
	[in] waittime	Timeout that is 1000ms by default. You can set the value according to your requirement.
Return value	● Success: TRUE ● Failure: FALSE	
Note	None	

For the cross reference among emType, pInBuf and pOutBuf, see the following table.

emType	Implication	plnBuf	pOutBuf
DH_CTRL_ARMED_EX	Arm and disarm	CTRL_ARM_DISARM_PARAM	NULL
DH_CTRL_SET_BYPASS	Set bypass function	NET_CTRL_SET_BYPASS	NULL

Table 5-4

5.6 Alarm Event

5.6.1 CLIENT_SetDVRMessCallBack

Item	Description	
Name	Set alarm callback	
Function	void CLIENT_SetDVRMessCallBack(fMessCallBack cbMessage, LDWORD dwUser);	
Parameter	[in] cbMessage	Message callback ● Call back the device status such as alarm status ● Callback is prohibited when this parameter is set as 0.
	[in] dwUser	User customized data
Return value	None	

Item	Description
Note	● Set the callback of device message to get the current device status. It is not relevant to calling sequence. By default, SDK does not call back this interface. ● The callback fMessCallBack must call the subscription interface of alarm information to make CLIENT_StartListenEx effective.

5.6.2 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	
Note	When subscribing the device information, the obtained information is called from the value set by CLIENT_SetDVRMessCallBack.	

5.6.3 CLIENT_StopListen

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

5.7 Device Status

5.7.1 CLIENT_QueryDevState

Item	Description
Name	Directly get the connection status of remote device

Item	Description	
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 information type
	[out] pBuf	Receive the data cache returned by query. The data structure is different according to the query type. For details, see Table 5-5.
	[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
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	For the relationship between dwPTZCommand and Param1, Param2 and Param3, see Table 5-5.	

For the relationship between information type and structure, see Table 5-5.

Query item	**nType**	**pBuf**
Query alarm channel status	DH_DEVSTATE_ALL_ALARM_CHANNELS_STATE	NET_CLIENT_ALARM_CHANNELS_STATE
Query power and battery information	DH_DEVSTATE_POWER_STATE	DH_POWER_STATUS

Table 5-5

5.8 Voice Talk

5.8.1 CLIENT_GetDevProtocolType

Item	Description
Name	Get voice talk mode supported by device
Function	BOOL CLIENT_GetDevProtocolType(LLONG lLoginID, EM_DEV_PROTOCOL_TYPE *pemProtocolType);

Item	Description	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2
	[out]pemProtocolType	Protocol type supported by device and corresponds to structure of EM_DEV_PROTOCOL_TYPE
Return value	● Success: TRUE ● Failure: FALSE	
Note	None	

5.8.2 CLIENT_SetDeviceMode

Item	Description	
Name	Set voice talk mode for device	
Function	BOOL CLIENT_SetDeviceMode(LLONG lLoginID, EM_USEDEV_MODE emType, void *pValue);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginEx2
	[in]emType	Enumeration value
	[in] dwUser	The structure data pointer corresponding to enumeration value. See Table 5-6.
Return value	● Success: TRUE ● Failure: FALSE	
Note	None	

For the relationship between emType and pValue, see Table 5-6.

Query item	**nType**	**pBuf**
DH_TALK_ENCODE_TYPE	Specify a certain mode to perform voice talk	DHDEV_TALKDECODE_INFO
DH_TALK_CLIENT_MODE	Set client mode for voice talk	None
DH_TALK_SPEAK_PARAM	Set speak parameters for voice talk	NET_SPEAK_PARAM
DH_TALK_MODE3	Set voice talk parameters for the 3rd generation device	NET_TALK_EX

Table 5-6

5.8.3 CLIENT_StartTalkEx

Item	Description
Name	Open voice talk
Function	LLONG CLIENT_StartTalkEx(LLONG lLoginID, pfAudioDataCallBack pfcb, LDWORD dwUser);

Item	Description	
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	

5.8.4 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	

5.8.5 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	This interface is valid only in Windows system.	

5.8.6 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	This interface is valid only in Windows system.	

5.8.7 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.	

5.8.8 CLIENT_AudioDecEx

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	

6 Callback Definition

6.1 fSearchDevicesCB

Item	Description	
Name	Callback of device search	
Function	typedef void(CALLBACK *fSearchDevicesCB)(DEVICE_NET_INFO_EX * pDevNetInfo, void* pUserData);	
Parameter	[out]pDevNetInfo	Information of the searched device
	[out]pUserData	User data
Return value	None	
Note	None	

6.2 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	

6.3 fHaveReConnect

Item	Description
Name	Reconnection callback
Function	typedef void (CALLBACK *fHaveReConnect)(

Item	Description	
	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	

6.4 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: Original data. ● 1: Information with frame. ● 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 tagCBPCMDataParam structure when

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

6.5 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.	

6.6 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 6-1.
	[out] lLoginID	Return value of login interface.

Item	Description	
	[out] pBuf	Receives the buffer of alarm data. The entered data is different dependent on the listen data and value of ICommand.
	[out] dwBufLen	Length of pBuf. The unit is byte.
	[out] pchDVRIP	Device IP.
	[out] nDVRPort	Port.
	[out] dwUser	The user customized data.
Return value	● Correct execution of callback: TRUE. ● Wrong execution of callback: FALSE.	
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 6-1.

Alarm module	**Alarm type**	**lCommand**	**pBuf**
Alarm host	Local alarm	DH_ALARM_ALARM_EX2	ALARM_ALARM_INFO_EX2
	Power damage	DH_ALARM_POWERFAULT	ALARM_POWERFAULT_INFO
	Anti-dismantling	DH_ALARM_CHASSISINTRUDED	ALARM_CHASSISINTRUDED_INFO
	Extending alarm input	DH_ALARM_ALARMEXTENDED	ALARM_ALARMEXTENDED_INFO
	Urgency	DH_URGENCY_ALARM_EX	The data is 16 bytes and each byte indicates a channel status. 1: There is an alarm 0: No alarm
	Battery low voltage	DH_ALARM_BATTERYLOWPOWER	ALARM_BATTERYLOWPOWER_INFO
	Device talks to platform	DH_ALARM_TALKING_INVITE	ALARM_TALKING_INVITE_INFO
	Arming mode changes	DH_ALARM_ARMMODE_CHANGE_EVENT	ALARM_ARMMODE_CHANGE_INFO
	Bypass status of protection zone changes	DH_ALARM_BYPASSMODE_CHANGE_EVENT	ALARM_BYPASSMODE_CHANGE_INFO
	Source signal of alarm input	DH_ALARM_INPUT_SOURCE_SIGNAL	ALARM_INPUT_SOURCE_SIGNAL_INFO
	Alarm clear	DH_ALARM_ALARMCLEAR	ALARM_ALARMCLEAR_INFO
	Subsystem status changes	DH_ALARM_SUBSYSTEM_STATE_CHANGE	ALARM_SUBSYSTEM_STATE_CHANGE_INFO
	Expanded module lost	DH_ALARM_MODULE_LOST	ALARM_MODULE_LOST_INFO

Alarm module	Alarm type	lCommand	pBuf
	PSTN lost	DH_ALARM_PSTN_BREAK_LINE	ALARM_PSTN_BREAK_LINE_INFO
	Analog alarm	DH_ALARM_ANALOG_PULSE	ALARM_ANALOGPULSE_INFO
	Alarm transmission	DH_ALARM_PROFILE_ALARM_TRANSMIT	ALARM_PROFILE_ALARM_TRANSMIT_INFO
	Wireless device low battery	DH_ALARM_WIRELESSDEV_LOWPOWER	ALARM_WIRELESSDEV_LOWPOWER_INFO
	Protection zone status changes	DH_ALARM_DEFENCE_ARMMODE_CHANGE	ALARM_DEFENCE_ARMMODECHANGE_INFO
	Armed or disarmed status of subsystem changes	DH_ALARM_SUBSYSTEM_ARMMODE_CHANGE	ALARM_SUBSYSTEM_ARMMODECHANGE_INFO
	Sensor abnormity	DH_ALARM_SENSOR_ABNORMAL	ALARM_SENSOR_ABNORMAL_INFO
	Patient movement status	DH_ALARM_PATIENTDETECTION	ALARM_PATIENTDETECTION_INFO
Access	Access	DH_ALARM_ACCESS_CTL_EVENT	ALARM_ACCESS_CTL_EVENT_INFO
	Details about access is left unclosed	DH_ALARM_ACCESS_CTL_NOT_CLOSE	ALARM_ACCESS_CTL_NOT_CLOSE_INFO
	Details about breaking in	DH_ALARM_ACCESS_CTL_BREAK_IN	ALARM_ACCESS_CTL_BREAK_IN_INFO
	Details about repeated entrance	DH_ALARM_ACCESS_CTL_REPEAT_ENTER	ALARM_ACCESS_CTL_REPEAT_ENTER_INFO
	Details about forced access	DH_ALARM_ACCESS_CTL_DURESS	ALARM_ACCESS_CTL_DURESS_INFO
	Access opened by several persons	DH_ALARM_OPENDOORGROUP	ALARM_OPEN_DOOR_GROUP_INFO
	Anti-dismantling	DH_ALARM_CHASSISINTRUDED	ALARM_CHASSISINTRUDED_INFO
	Local alarm	DH_ALARM_ALARM_EX2	ALARM_ALARM_INFO_EX2
	Access status	DH_ALARM_ACCESS_CTL_STATUS	ALARM_ACCESS_CTL_STATUS_INFO
	Fingerprint obtaining	DH_ALARM_FINGER_PRINT	ALARM_CAPTURE_FINGER_PRINT_INFO
Video intercom	In direct connect, no response to calling	DH_ALARM_CALL_NO_ANSWERED	NET_ALARM_CALL_NO_ANSWERED_INFO

Alarm module	Alarm type	lCommand	pBuf
	Mobile phone number check	DH_ALARM_TELEPHONE_CHECK	ALARM_TELEPHONE_CHECK_INFO
	VTS status check	DH_ALARM_VTSTATE_UPDATE	ALARM_VTSTATE_UPDATE_INFO
	VTO face detect	DH_ALARM_ACCESSIDENTIFY	NET_ALARM_ACCESSIDENTIFY
	Device invites voice talk	DH_ALARM_TALKING_INVITE	ALARM_TALKING_INVITE_INFO
	Device cancels inviting voice talk	DH_ALARM_TALKING_IGNORE_INVITE	ALARM_TALKING_IGNORE_INVITE_INFO
	Device hangs up voice talk	DH_ALARM_TALKING_HANGUP	ALARM_TALKING_HANGUP_INFO
	Over speed detected by Radar	DH_ALARM_RADAR_HIGH_SPEED	ALARM_RADAR_HIGH_SPEED_INFO

Table 6-1