NetSDK Programming Manual
(Intelligent Analysis Server)

V1.0.1

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 analysis sever. 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
2	V1.0.1	Delete some library files in "Table 1-1."	January, 2019

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
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.
Video Channel	A channel number for video stream transmission between SDK and devices. For example, if NVR communicates with the front-end devices such as SD, IPC, NVR manages SD or IPC as a video channel. If SDK directly connects to a front-end device, usually the channel number is 0.
Login Handle	A unique ID for communication with device. It is of a round number and not 0 and maintained within SDK.
Monitoring Handle	A unique ID for data transmission with a video channel. It is of a round number and not 0.
Subscription Handle	A unique ID for subscribing real-time upload of intelligent event. It is of a round number and not 0.
Capability Set	A set of capabilities for a single function or a set of functions. The capability set might be different dependent on device model.
Detection of Crossing fence	Automatic detection of crossing fence behavior.
Detection of Invading Tripwire	Automatic detection of invading tripwire behavior.
Detection of Invading Area	Automatic detection of invading warning area behavior: Crossing area and staying in the area.
Detection of Abandoning Objects	Automatic detection of the emerging and still objects.
Detection of Safeguarding Objects	Alarm for the selected objects that has been shaded for a long time or moved from the place.
Detection of Moving Objects	Alarm for the objects that has been moved from the detection covered place.
Detection of Wandering	Automatic detection of invading behavior that stays longer than the preset time.
Detection of Video Abnormal	Automatic detection of abnormal behaviors such as scene changes, video loss, video shade, excessive brightness, excessive darkness, definition, stripe, noise, and color cast.
Detection of Climbing	Automatic detection of climbing behavior in the warning area.

Detection of Fighting	Automatic detection of fighting behavior in the warning area.
Detection of Off-post	Automatic detection of leaving the post in the warning area.
Detection on Getting up	Automatic detection when the target is getting up.
Detection on Crossing Over Warning Line	Automatic detection of crossing over warning line.
Detection on Invading Warning Line	Automatic detection of invading warning line, including entering warning area, leaving warning area, and within the warning area.
Detection of Normal Face	Detection of normal face in face monitoring video.
Detection of Abnormal Face	Detection of abnormal face in face monitoring video, including covering of mouse or eyes.
Detection of Neighboring Faces	Detection of two faces that appear in the monitoring area indicating one is following the other.
Illegal Sticky Notes	The sticky notes on the cash slot monitoring video.
Detection of Entering Operation Area	Detection of entering operation area in the cash slot monitoring video.
Detection of Leaving Operation Area	Detection of leaving operation area in the cash slot monitoring video.
Detection of Staying Operation Area	Detection of staying operation area in the cash slot monitoring video.

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, report and snapshot of intelligent events, and guest flow statistics.

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
	fisheye.dll	Fisheye correction library
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
Auxiliary library of	libInfra.so	Infrastructure library

Library type	Library file name	Library file description
"libavnetsdk.so"	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 decode and play the streams.
- The auxiliary library decodes the audio and video streams for the functions such as monitoring, playback, and bidirectional talk, and collects the local audios.
- If the function library includes libavnetsdk.so or avnetsdk.dll, the corresponding auxiliary 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

1.3 Application Scenario

- General functions such as login, monitoring, voice talk, and alarm.
- Functions as a client to subscribe the intelligent event from the intelligent analysis server, handle the report service of intelligent events, subscribe guest flow statistics, and report the statistics result.

.

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.

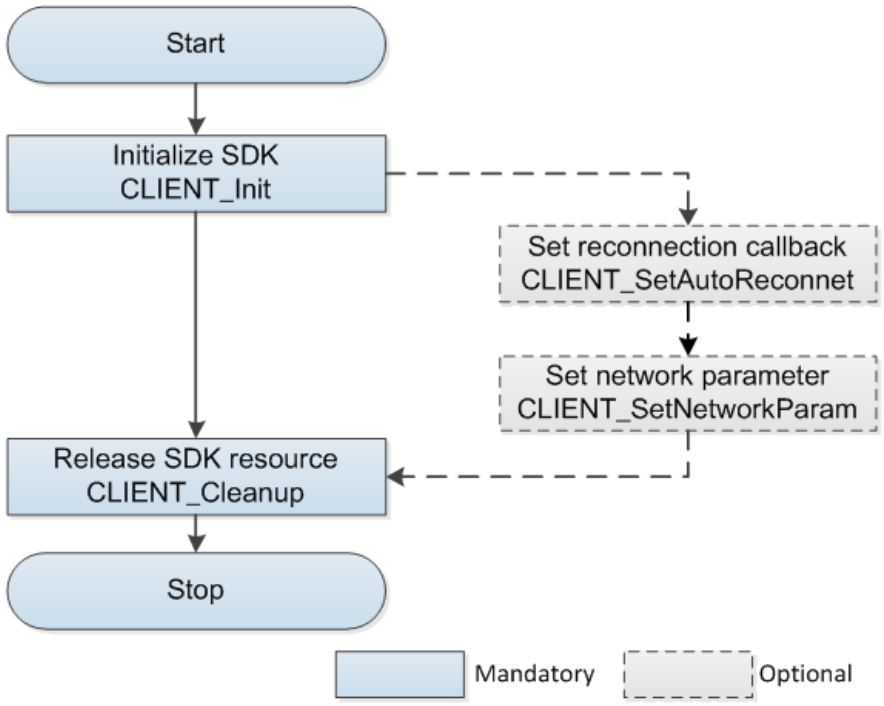

Figure 2-1

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

- **CLIENT_Init** and **CLIENT_Cleanup** should be used as a pair, which can be called by single thread for multiple times. However, it is suggested to call the pair for only one time globally.
- 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, alarm and snapshot subscription can be resumed after reconnection is successful.

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.

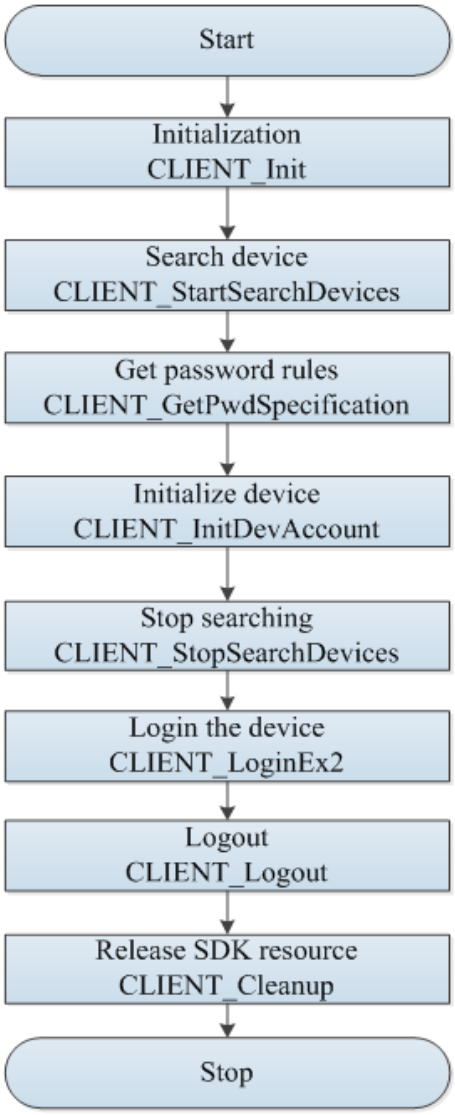

Figure 2-2

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.

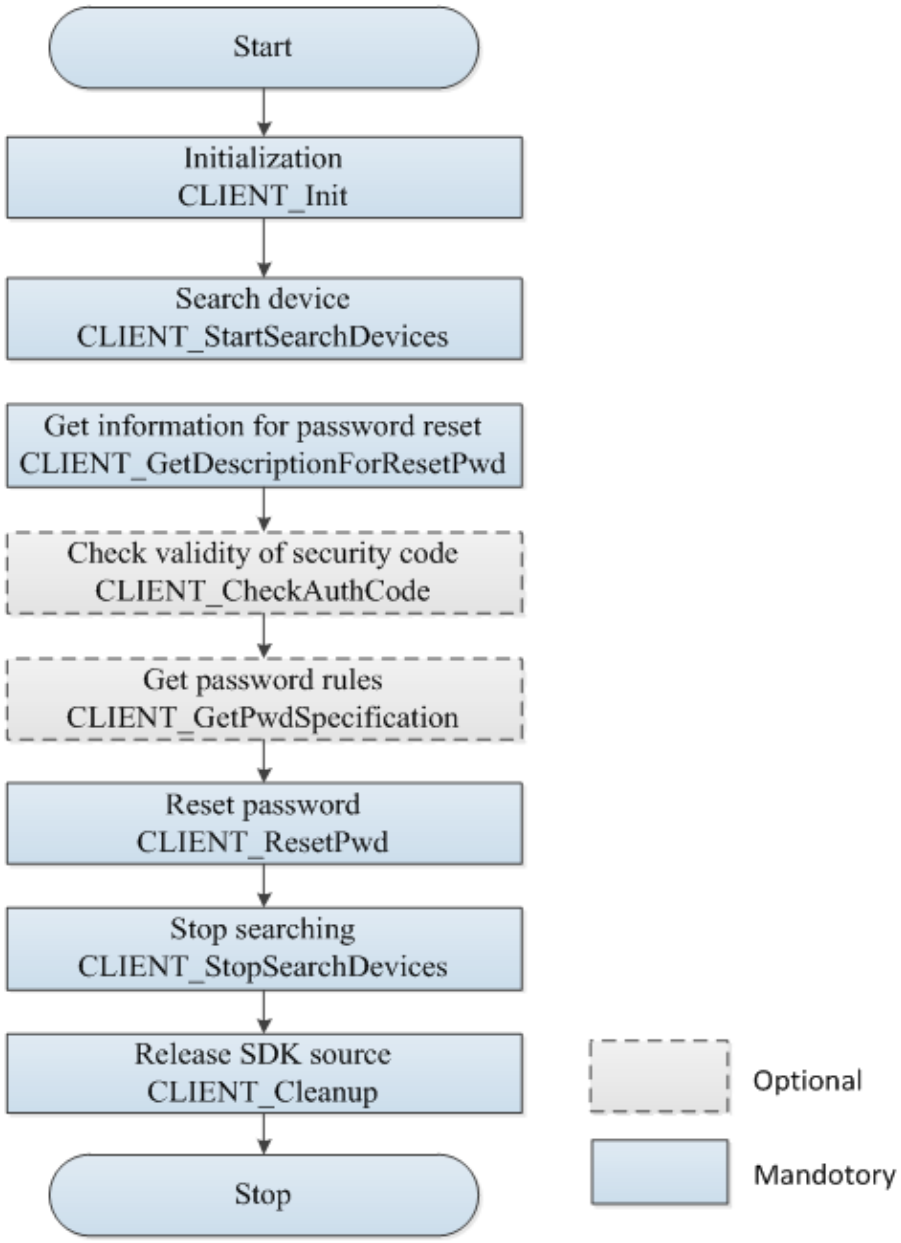

Figure 2-3

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_GetDescriptionForResetPw** 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.

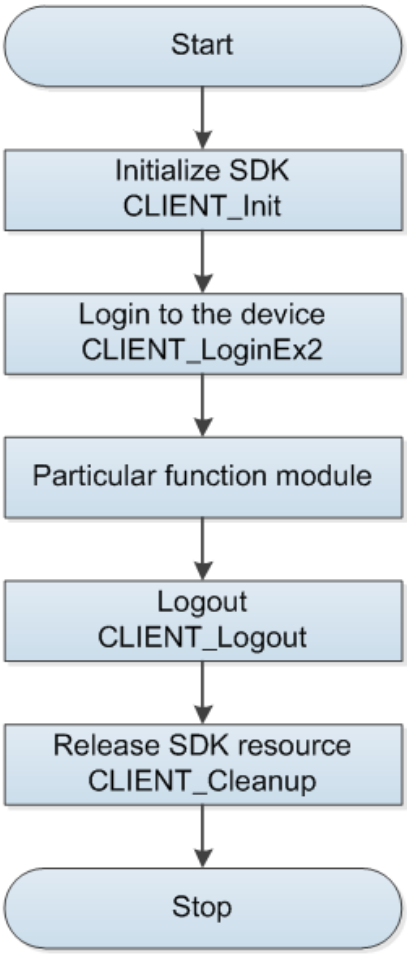

Figure 2-4

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.
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.

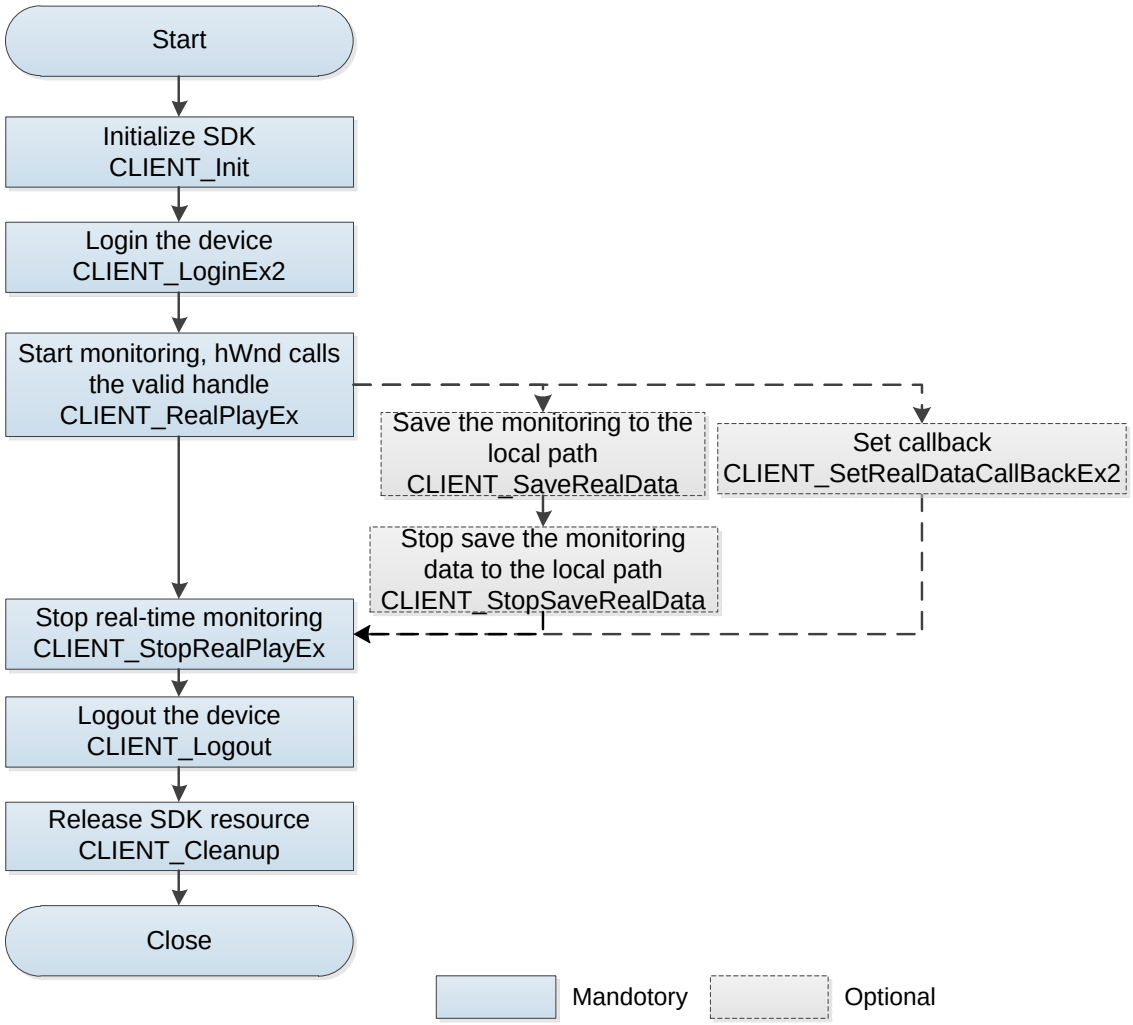

Figure 2-5

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 enable 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 1-1.
- 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.

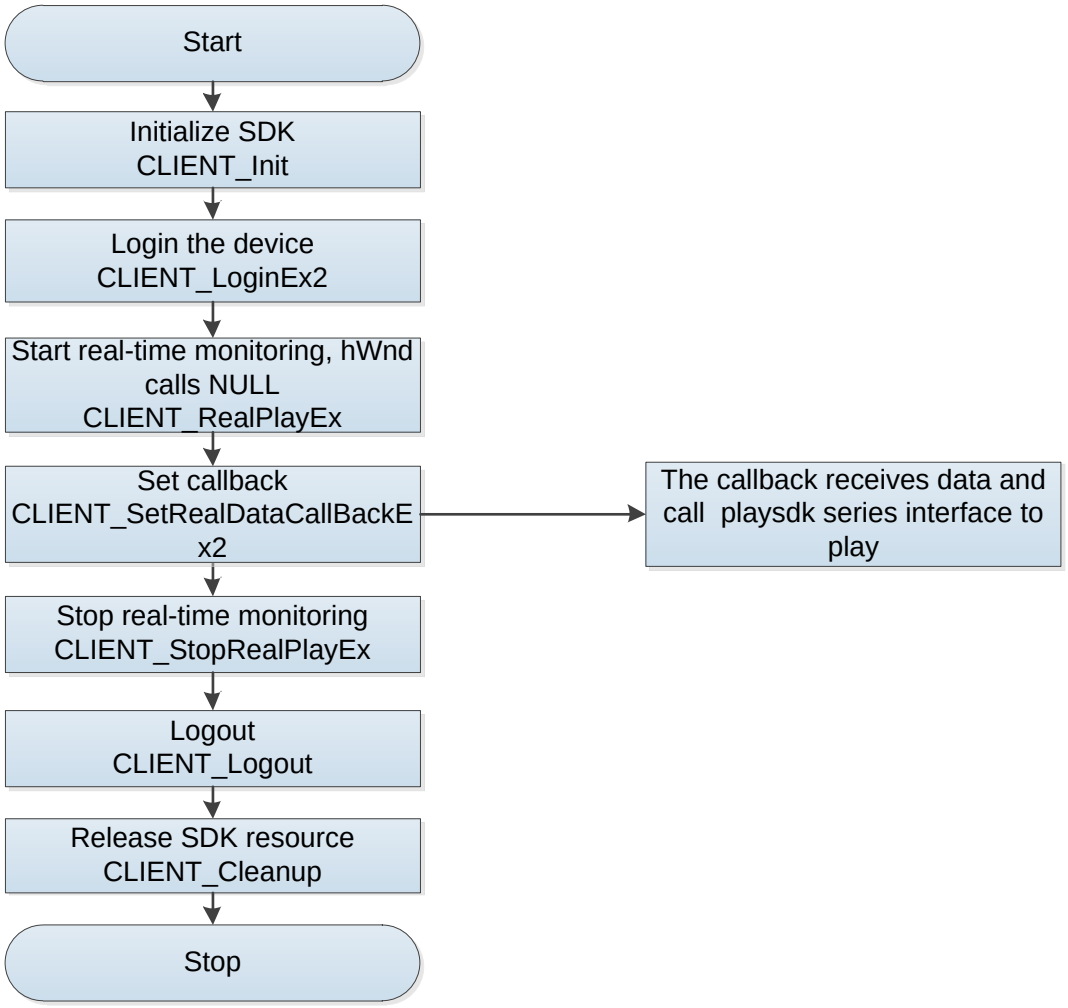

Figure 2-6

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 enable 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 Library

```
// Take opening the main stream monitoring of channel 1 as an example. The parameter hWnd is a handle of interface window.
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 Detention Dedicated

2.5.1 Introduction

Product Description

Prison dedicated function integrates audio & video data collecting, wire transmission, storage, and intelligent detection and alarm. Besides the common outdoor intelligent detection functions, it has the functions such as fight detecting, detecting on getting up, off-post detecting, detecting on climbing, and video abnormal. This product is widely used in the particular places such as prison and detention house.

Product Model

This function mainly applies to DH-IVS-IP7200 series device.

SDK Access Function

You can access to SDK to subscribe the intelligent event of behavior analysis from the DH-IVS-IP7200 series device, and obtains the subscribed intelligent event and data information.

NOTE

Before subscribing the behavior detecting event, you need to configure the rules triggered by behavior detecting at Web.

For the behavior detecting supported by the DH-IVS-IP7200 series product, see Table 2-6.

Scenario	Supported behavior detection

Scenario	Supported behavior detection
Ordinary	Analyze and detect the behaviors such as crossing fence, invading tripwire, abandoning objects, safeguarding objects, moving objects, wandering detecting, and video abnormal.
Detention	Analyze and detect the behaviors such as sound abnormal, invading area, climbing detection, fighting detection, off-post detection, detecting upon getting up, video abnormal, and wandering detection.

Table 2-6

2.5.2 Interface Overview

Interface	Implication
CLIENT_RealLoadPictureEx	Start analyzing intelligent events and subscribing data.
CLIENT_StopLoadPic	Stop analyzing intelligent events and subscribing data.
fAnalyzerDataCallBack	Intelligent events analysis data.

Table 2-7

2.5.3 Process

For the process of subscribing intelligent events on behaviors analysis, see Figure 2-7.

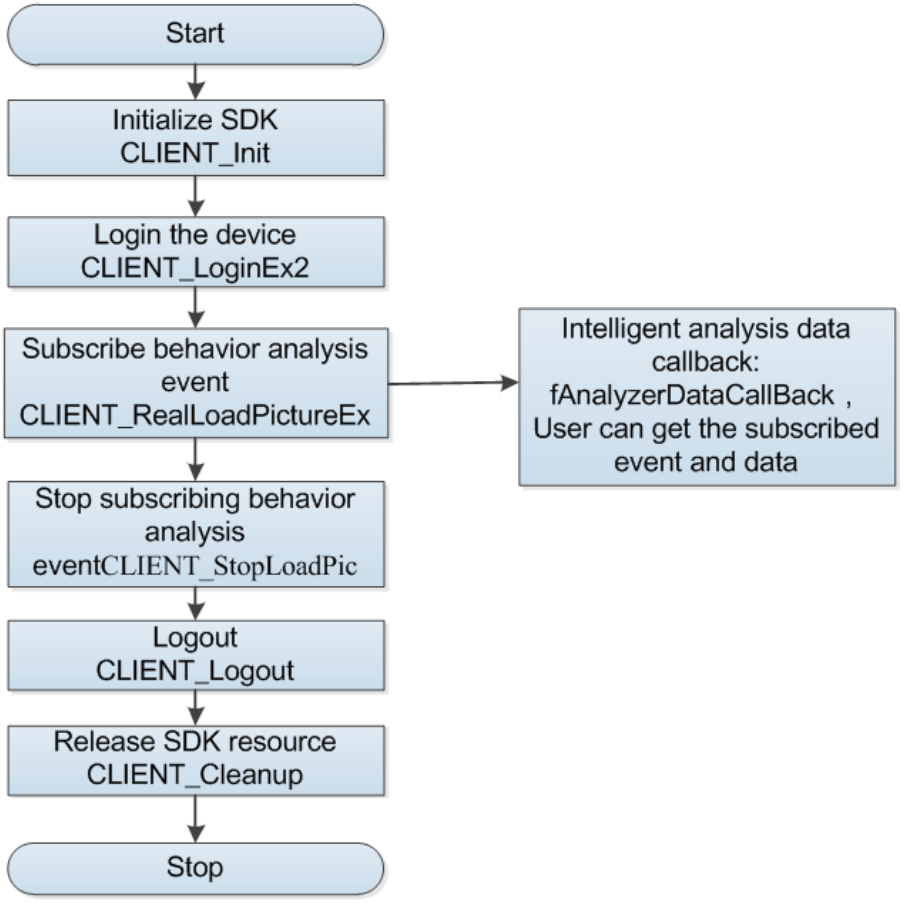

Figure 2-7

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 Call **CLIENT_RealLoadPictureEx** to start subscribing intelligent event.
Step 4 Call **fAnalyzerDataCallBack** to handle the subscribed intelligent event and data.
Step 5 Call **CLIENT_StopLoadPic** to stop subscribing the intelligent event.
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

- Set data receiving cache: Because SDK default cache is 2 M, when the data is over 2 M, call **CLIENT_SetNetworkParam** to set the receiving cache, otherwise the data pack will be lost.
- Handle the data of callback: It is not recommended to perform the time consuming operations such as I/O or delay in parameter fAnalyzerDataCallBack, for example, local picture saving, database insertion. In Windows system, you can use postMessage to throw out the data and start threading treatment.

2.5.4 Example Code

```
// Callback of intelligent analysis data
// It is not recommended to call SDK interface in this callback.
//Set this callback through "CLIENT_RealLoadPictureEx/CLIENT_RealLoadPicture." When there is an intelligent picture reported at the device end, SDK will call this function.
// "nSequence" represents the situation when the reported pictures are the same. "0" represents the first appearance, "2" represents the last appearance or the appearance only once, and "1" represents there will be more appearances from now on.
// "int nState = *(int*) reserved" represents the current status of the callback data. "0" represents the current is real-time data, "1" represents the current callback data is offline, and "2" represents the offline data transmission finished.
// The return value is abolished, so it does not have any meaning.
int  CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    switch(dwAlarmType)
    {
        // Crossing the fence (corresponding to DEV_EVENT_CROSSFENCEDETECTION_INFO)
        case EVENT_IVS_CROSSFENCEDETECTION:
        {
```

```
            DEV_EVENT_CROSSFENCEDETECTION_INFO* pCrossFence = NULL;
            pCrossFence = (DEV_EVENT_CROSSFENCEDETECTION_INFO*)pAlarmInfo;
            ivsInfoEx.stuEventInfo[0].alarmAction = pCrossFence->bEventAction;
            ivsInfoEx.stuEventInfo[0].alarmType = dwAlarmType;
            strncpy(ivsInfoEx.stuEventInfo[0].szRuleName, pCrossFence->szName,
            (sizeof(pCrossFence->szName) >= sizeof(ivsInfoEx.stuEventInfo[0].szRuleName)
            ? sizeof(ivsInfoEx.stuEventInfo[0].szRuleName) : sizeof(pCrossFence->szName)));
           ivsInfoEx.stuEventInfo[0].nObjectNum = 1;
           memcpy(&ivsInfoEx.stuEventInfo[0].stuObject[0],&pCrossFence->stuObject,
            sizeof(pCrossFence->stuObject));
        }
            break;
    // Tripwire/warning line invasion(Corresponding to DEV_EVENT_CROSSREGION_INFO)
    case EVENT_IVS_CROSSLINEDETECTION:
            {
                //.....................................................
            }
            break;
    .......
    default:
            printf("other event type[%d]\n", dwAlarmType);
            break;
}
// Download picture
if (dwBufSize > 0 && NULL != pBuffer)
{
    // In order to prevent receiving several pictures simultaneously, use " i " to remark.
    static int i;
    char szPicturePath[256] = "";
    time_t stuTime;
    time(&stuTime);
    char szTmpTime[128] = "";
    strftime(szTmpTime, sizeof(szTmpTime) - 1, "%y%m%d_%H%M%S", gmtime(&stuTime));
    _snprintf(szPicturePath, sizeof(szPicturePath)-1, "%d_%s.jpg", ++i, szTmpTime);
```

```
            FILE* pFile = fopen(szPicturePath, "wb");
            if (NULL == pFile)
            {
                return 0;
            }
            int nWrite = 0;
            while(nWrite != dwBufSize)
            {
                nWrite += fwrite(pBuffer + nWrite, 1, dwBufSize - nWrite, pFile);
            }

            fclose(pFile);
        }
        return 1;
}
// Intelligent event subscription code
{
        // Resource initialization
        .....
        // Device login
        .....
        // Subscribe the alarm by intelligent picture
        LDWORD dwUser = 0;
        int nChannel = 0;
        // Every setting corresponds to one channel and one type of event.
        // If you need to set the channel to upload all types of events, you can set dwAlarmType as EVENT_IVS_ALL.
        // If you need to set one channel to upload two types of event, please call CLIENT_RealLoadPictureEx twice and import different types of events.
        lRealLoadHandle = CLIENT_RealLoadPictureEx(lLoginHandle, nChannel, EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, dwUser, NULL);
        //.....

        // Stop subscribing alarm pictures
        CLIENT_StopLoadPic(lRealLoadHandle))
```

```
    // Exit
    CLIENT_Logout(lLoginHandle))
    CLIENT_Cleanup();
}
```

2.6 Intelligent ATM

2.6.1 Introduction

Product Description

The intelligent Automatic Teller Machine (ATM) detects and analyzes the front-end devices according to the analysis videos, rules and functions configured by the platform, and gives an alarm when the illegal situations such as sticky notes, abnormal faces, wandering, abandoned objects. This product mainly applies to Dahua intelligent analysis server DH-IVS-IF70XX series of financial industry.

Product Model

The Intelligent ATM function mainly applies to the following models:

- 16 channels intelligent analysis server basic: DH-IVS -IF7016-B
- 16 channels intelligent analysis server advanced: DH-IVS –IF7016-A
- 16 channels intelligent analysis server full: DH-IVS –IF7016-F
- 24 channels intelligent analysis server basic: DH-IVS –IF7024-B
- 24 channels intelligent analysis server advanced: DH-IVS –IF7024-A
- 24 channels intelligent analysis server full: DH-IVS –IF7024-F

SDK Access Function

You can access to SDK to subscribe the intelligent event of behavior analysis from the DH-IVS-IF70XX series device, and obtain the subscribed intelligent event and data information.

NOTE

Before subscribing the behavior detecting event, you need to configure the rules triggered by behavior detecting at Web.

DH-IVS-IF70XX series product support the following behaviors detection: crossing the warning line, invading the warning area, wandering, objects abandoning, objects moving, face abnormal, similar faces, illegal sticky notes, and entering, leaving and staying in the operation area.

2.6.2 Interface Overview

Interface	Implication
CLIENT_RealLoadPictureEx	Start analyzing intelligent events and

Interface	Implication
	subscribing data.
CLIENT_StopLoadPic	Stop analyzing intelligent events and subscribing data.
fAnalyzerDataCallBack	Intelligent events analysis data.

Table 2-8

2.6.3 Process

For the process of subscribing intelligent events on behaviors analysis in the intelligent ATM, see Figure 2-8.

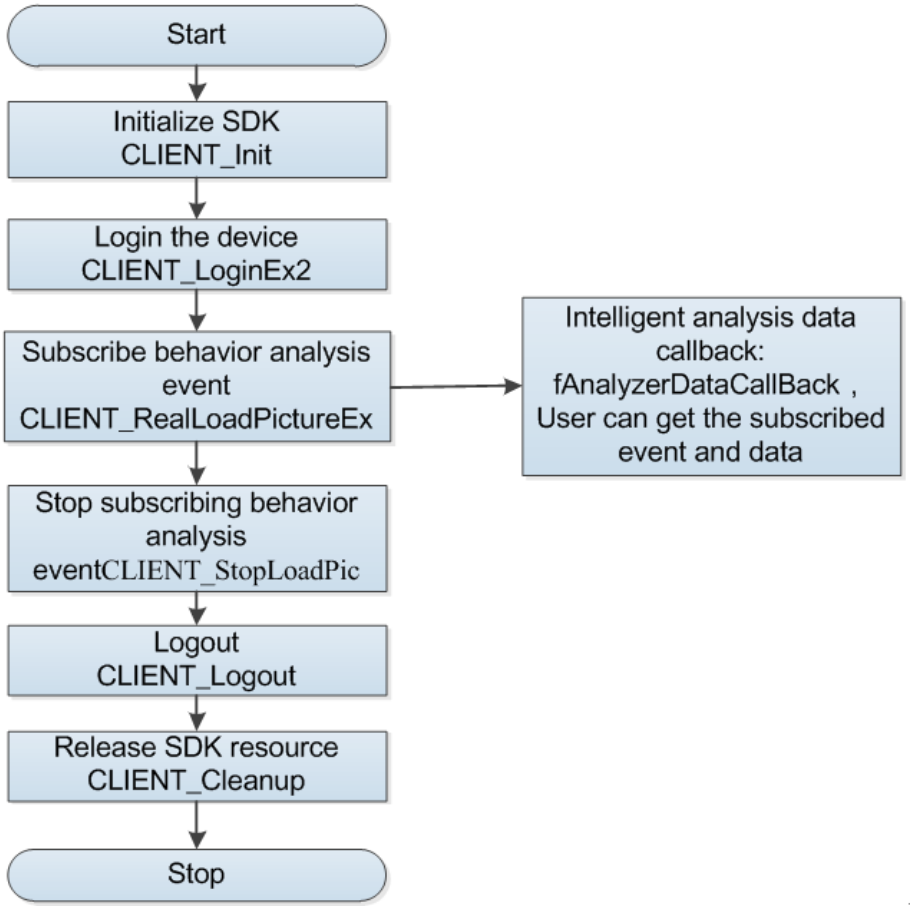

Figure 2-8

Process Description

Step 1 Call **CLIENT_Init** to initialize SDK.
Step 2 Call **CLIENT_LoginEx2** to login the device.
Step 3 Call **CLIENT_RealLoadPictureEx** to start subscribing intelligent event.
Step 4 Call **fAnalyzerDataCallBack** to handle the subscribed intelligent event and data.
Step 5 Call **CLIENT_StopLoadPic** to stop the intelligent event subscription.
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

- Set data receiving cache: Because SDK default cache is 2 M, when the data is over 2 M, call **CLIENT_SetNetworkParam** to set the receiving cache, otherwise the data pack will be lost.
- Handle the data of callback: It is not recommended to perform the time consuming operations such as I/O or delay in parameter fAnalyzerDataCallBack, for example, local picture saving, database insertion. In Windows system, you can use postMessage to throw out the data and start threading treatment.

2.6.4 Example Code

```
// Callback of intelligent analysis data
// It is not recommended to call SDK interface in this callback.
//Set this callback through "CLIENT_RealLoadPictureEx/CLIENT_RealLoadPicture." When there is an intelligent picture uploaded at the device end, SDK will call this function.
// "nSequence" represents the situation when the reported pictures are the same. "0" represents the first appearance, "2" represents the last appearance or the appearance only once, and "1" represents there will be more appearances from now on.
// "int nState = *(int*) reserved" represents the current status of the callback data. "0" represents the current is real-time data, "1" represents the current callback data is offline, and "2" represents the offline data transmission finished.
// The return value is abolished, so it does not have any meaning.
int  CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    int nAlarmChn = 0;
    IVS_CFG_ANALYSEVENTS_INFOEX ivsInfoEx = {0};
    ivsInfoEx.nEventsNum = 1;
    switch(dwAlarmType)
    {
        // Crossing the fence (corresponding to DEV_EVENT_CROSSFENCEDETECTION_INFO)
        case EVENT_IVS_CROSSFENCEDETECTION:
        {
            DEV_EVENT_CROSSFENCEDETECTION_INFO* pCrossFence = NULL;
            pCrossFence = (DEV_EVENT_CROSSFENCEDETECTION_INFO*)pAlarmInfo;
            ivsInfoEx.stuEventInfo[0].alarmAction = pCrossFence->bEventAction;
            ivsInfoEx.stuEventInfo[0].alarmType = dwAlarmType;
            strncpy(ivsInfoEx.stuEventInfo[0].szRuleName, pCrossFence->szName,
            (sizeof(pCrossFence->szName) >= sizeof(ivsInfoEx.stuEventInfo[0].szRuleName)
```

```
                ? sizeof(ivsInfoEx.stuEventInfo[0].szRuleName) : sizeof(pCrossFence->szName)));
            ivsInfoEx.stuEventInfo[0].nObjectNum = 1;
            memcpy(&ivsInfoEx.stuEventInfo[0].stuObject[0],&pCrossFence->stuObject,
              sizeof(pCrossFence->stuObject));
          }
            break;
        // Tripwire/warning line invasion(Corresponding to DEV_EVENT_CROSSREGION_INFO)
        case EVENT_IVS_CROSSLINEDETECTION:
            {
                //.....................................................
            }
            break;
      .......
        default:
            printf("other event type[%d]\n", dwAlarmType);
            break;
}
// Download picture
if (dwBufSize > 0 && NULL != pBuffer)
{
        // In order to prevent receiving several pictures simultaneously, use " i " to remark.
        static int i;
        char szPicturePath[256] = "";
        time_t stuTime;
        time(&stuTime);
        char szTmpTime[128] = "";
        strftime(szTmpTime, sizeof(szTmpTime) - 1, "%y%m%d_%H%M%S", gmtime(&stuTime));
        _snprintf(szPicturePath, sizeof(szPicturePath)-1, "%d_%s.jpg", ++i, szTmpTime);
        FILE* pFile = fopen(szPicturePath, "wb");
        if (NULL == pFile)
        {
            return 0;
        }
        int nWrite = 0;
```

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

            fclose(pFile);
        }
        return 1;
}
// Intelligent event subscription code
{
        // Resource initialization
        .....
        // Device login
        .....
        // Subscribe the alarm by intelligent picture
        LDWORD dwUser = 0;
        int nChannel = 0;
        // Every setting corresponds to one channel and one type of event.
        // If you need to set the channel to upload all types of events, you can set dwAlarmType as EVENT_IVS_ALL.
        // If you need to set one channel to upload two types of event, please call CLIENT_RealLoadPictureEx twice and import different types of events.
        lRealLoadHandle = CLIENT_RealLoadPictureEx(lLoginHandle, nChannel, EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, dwUser, NULL);
        //.....

        // Stop subscribing alarm pictures
        CLIENT_StopLoadPic(lRealLoadHandle))
        // Exit the device
        CLIENT_Logout(lLoginHandle))
        CLIENT_Cleanup();
}
```

2.7 Guest Flow Statistics

2.7.1 Introduction

Product Description

The intelligent analysis server can accord to the front-end devices that are installed in the business area to calculate accurately the guest flow through each entrance. Such function is widely applied to the industries such as large-scale commercial complex, tourism, public security, cultural expo, and chain operation business.

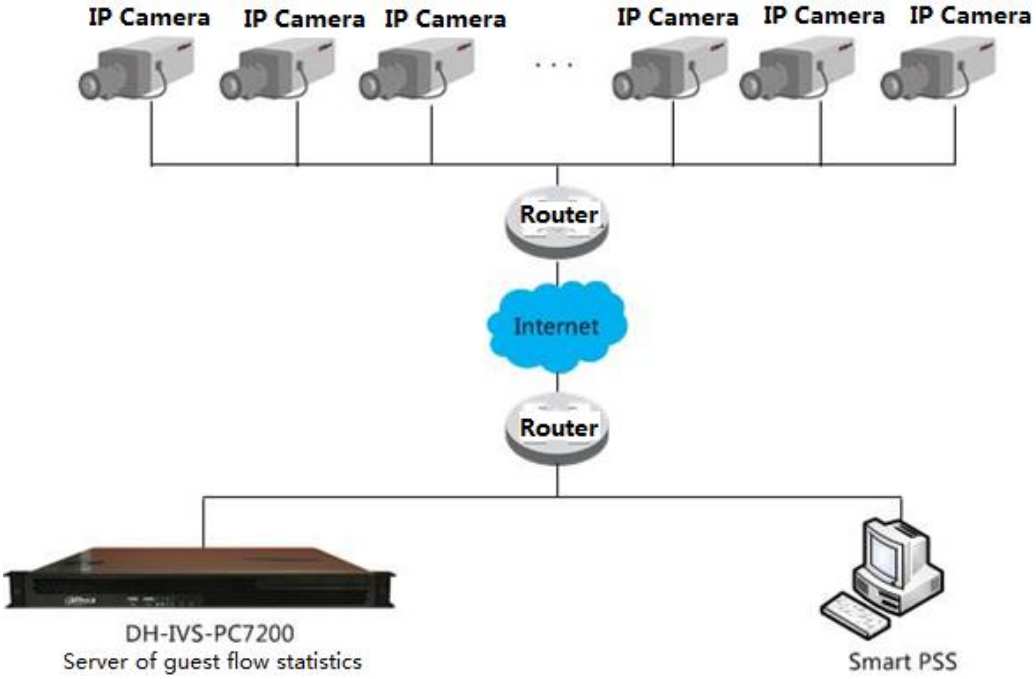

Figure 2-9

Product Model

This function mainly applies to DH-IVS-PC7200 series device.

SDK Access Function

SDK access realizes the following functions:

- Count the real-time guest flow information collected by the front-end devices
- Query the historical guest flow information collected by the intelligent analysis devices.

NOTE

Before subscribing the service of guest flow statistics, you need to configure the rules of guest flow statistics at Web.

2.7.2 Interfaces Overview

Interface	Description
CLIENT_AttachVideoStatSummary	Start subscribing the real-time guest flow statistics.
CLIENT_DetachVideoStatSummary	Stop subscribing the real-time guest flow statistics.
fVideoStatSumCallBack	Guest flow information.
CLIENT_StartFindNumberStat	Start querying the historical guest flow information.
CLIENT_DoFindNumberStat	Continue to inquire the historical guest flow information.
CLIENT_StopFindNumberStat	Stop querying the historical guest flow information.

Table 2-9

2.7.3 Process

The guest flow statistics mainly apply to the following two scenarios:

- Real-time guest flow statistics
  After SDK has subscribed the guest flow service from the device, the device will report the real-time guest flow information to SDK.
- Query of historical guest flow information
  You can specify the start time and end time to inquire the guest flow information, and the device returns the guest flow information for the specified period.

2.7.3.1 SDK Real-time Guest Flow Statistics

For the process of SDK real-time guest flow statistics, see Figure 2-10.

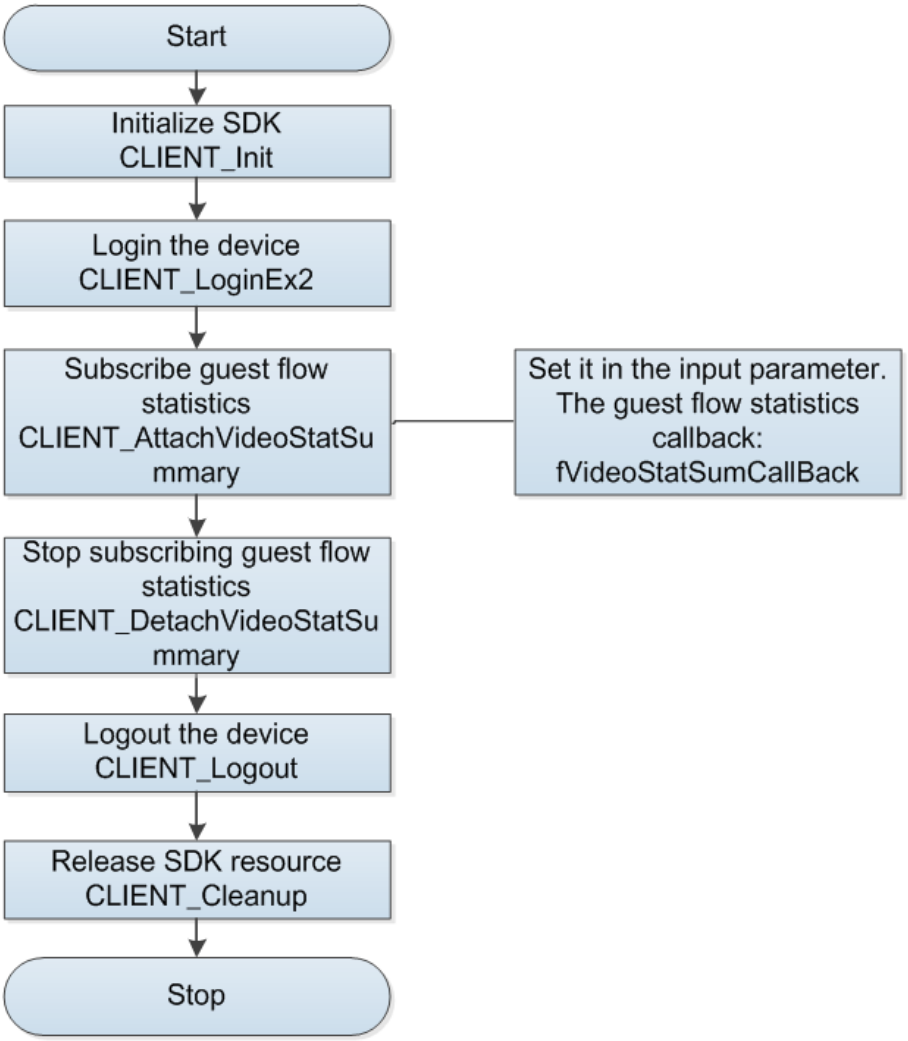

Figure 2-10

Process Description

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

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

Step 3 Call **CLIENT_AttachVideoStatSummary** to subscribe the guest flow statistics. You can set callback fVideoStatSumCallBack in the input parameter to get the gust flow data.

Step 4 Call **CLIENT_DetachVideoStatSummary** to stop subscription.

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.

2.7.3.2 Historical Guest Flow Statistics Information Inquiry

For the process of querying historical guest flow statistics, see Figure 2-11.

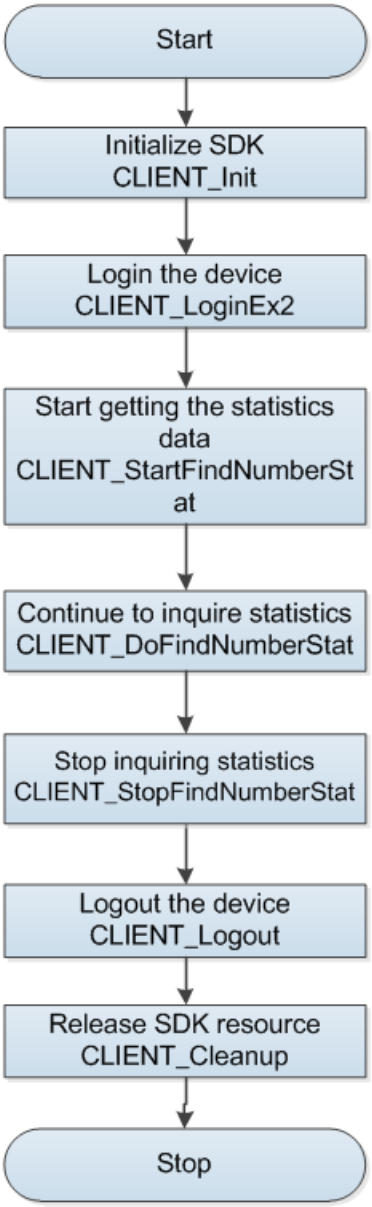

Figure 2-11

Process Description

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

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

Step 3 Call **CLIENT_StartFindNumberStat** to start obtaining the guest flow statistics.

Step 4 Call **CLIENT_DoFindNumberStat** to continue inquiring the guest flow statistics within a certain period.

Step 5 Call **CLIENT_StopFindNumberStat** to stop inquiring.

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.

2.7.4 Example Code

2.7.4.1 Upload of Real-time Guest Flow Statistics

```
//
#include "stdafx.h"
#include "dhnetsdk.h"
#pragma comment(lib,"dhnetsdk.lib")
//Handle the received data in the callback according to the situation.
void _stdcall VideoStatSumCallback(LLONG lAttachHandle, NET_VIDEOSTAT_SUMMARY* pBuf, DWORD dwBufLen, LDWORD dwUser)
{
    printf("Infomation:\n");
    printf("Channel ID:%d;\n rule name:%s;\ entered subtotal:%d;\n exited subtotal:%d\n",pBuf->nChannelID,pBuf->szRuleName,pBuf->stuEnteredSubtotal.nTotal,pBuf->stuExitedSubtotal.nTotal);
}
int main(int argc, char* argv[])
{
    // Initialization
    CLIENT_Init(NULL,0);
      .....
    // Login
    LLONG lLoginID = CLIENT_LoginEx(zDevIP, nPort, zUserName, zPsd, 0, NULL, &deviceInfo, &err);
    if (lLoginID)
    {
        NET_IN_ATTACH_VIDEOSTAT_SUM InParam={sizeof(NET_IN_ATTACH_VIDEOSTAT_SUM)};
        InParam.nChannel=0;
        InParam.cbVideoStatSum=VideoStatSumCallback;

        NET_OUT_ATTACH_VIDEOSTAT_SUM OutParam={0};
        OutParam.dwSize=sizeof(OutParam);
        int nWaitTime=5000;     //wait time
        LLONG attachHnd = 0;
        // Subscribe the guest flow statistics
```

```
            attachHnd = CLIENT_AttachVideoStatSummary(lLoginID,&InParam,&OutParam,nWaitTime)
            if(attachHnd)
            {
                printf("CLIENT_AttachVideoStatSummary sucess\n");
            }
            else
            {
                printf("error number:%x",CLIENT_GetLastError());
            }
        }
        else
        {
            printf("login fail\n");
        }
            .....
        // Cancel subscribing the guest flow statistics
        CLIENT_DetachVideoStatSummary(attachHnd);
        // Exit the device
        CLIENT_Logout(lLoginHandle));
        CLIENT_Cleanup();
        return 0;
}
```

2.7.4.2 Query of historical Guest Flow Statistics

```
// AttachVideoStatSummary.cpp : Define the entrance of the control desk application.
//
#include "stdafx.h"
#include "dhnetsdk.h"
#pragma comment(lib,"dhnetsdk.lib")
// Handle the received data in the callback according to the situation.
void _stdcall VideoStatSumCallback(LLONG lAttachHandle, NET_VIDEOSTAT_SUMMARY* pBuf, DWORD dwBufLen, LDWORD dwUser)
{
    printf("Infomation:\n");
```

```
    printf("Channel ID:%d;\n rule name:%s;\n entered subtotal:%d;\n exited subtotal:%d\n",pBuf->nChannelID,pBuf->szRuleName,pBuf->stuEnteredSubtotal.nTotal,pBuf->stuExitedSubtotal.nTotal);
}
int main(int argc, char* argv[])
{
    // Initialization
    CLIENT_Init(NULL,0);
      .........
    // Login
    LLONG lLoginID = CLIENT_LoginEx(zDevIP, nPort, zUserName, zPsd, 0, NULL, &deviceInfo, &err);
    if (lLoginID == 0)
    {
            CLIENT_Cleanup();
        return 0;
    }
        NET_IN_FINDNUMBERSTAT inParam ;
    inParam.dwSize = (uint)Marshal.SizeOf(inParam);
    inParam.nChannelID = nChannelID; // Channel ID to be inquired
    // Set the inquiry start time and end time to the hour for the moment.
    . . . . . .
    inParam.nGranularityType = 1; // Query granularity 0: minute, 1: hour, 3: week, 4: month, 5: season, 6: year
    inParam.nWaittime = 5000; // Timeout of waiting for data

    NET_OUT_FINDNUMBERSTAT outParam;
    outParam.dwSize = sizeof(outParam);
    LLONG findHnd = m_FindHandle = CLIENT_StartFindNumberStat(pLoginHandle, &inParam, &outParam);
    //
    if (findHand == 0)
    {
        printf("find number stat failed! \n");
        goto e_clear;;
    }
    NET_IN_DOFINDNUMBERSTAT inDoFind;
```

```
    inDoFind.dwSize = sizeof(inDoFind);
    inDoFind.nBeginNumber = 0; // Query from 0
    inDoFind.nCount = 10; // Query 10 lines each time
    inDoFind.nWaittime = 5000; // Interface timeout 5s

    NET_OUT_DOFINDNUMBERSTAT outStuDoFindNumStat = {sizeof(NET_OUT_DOFINDNUMBERSTAT)};
    outStuDoFindNumStat.pstuNumberStat = new DH_NUMBERSTAT[10];
    for (int i = 0; i < 10 ; i++)
    {
        outStuDoFindNumStat.pstuNumberStat[i].dwSize = sizeof(DH_NUMBERSTAT);
    }
    outStuDoFindNumStat.nBufferLen = 10 * sizeof(DH_NUMBERSTAT);
    int index = 0;
    do
    {
        if (CLIENT_DoFindNumberStat(findHand, &inDoFind, &outStuDoFindNumStat) > 0)
        {
            for (int i = 0; i < outStuDoFindNumStat.nCount; i++, index++)
            {
                // Query result
            }

            // Query next time
            inDoFind.nBeginNumber += inDoFind.nCount;// Query from the last stop place
        }
        else
        {
            printf("find error: \n");
            break;
        }
    } while (inDoFind.nBeginNumber >= outParam.dwTotalCount);
        .....
    // Stop querying guest flow
    CLINET_StopFindNumberStat(findHand);
```

```
e_clear:
        // Exit the device
CLIENT_Logout(lLoginID))
        CLIENT_Cleanup();
        return 0;
}
```

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 the SDK cleanup interface before the process ends.

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 Subscription of Intelligent Event

3.5.1 CLIENT_RealLoadPictureEx

Item	Description
Name	Start subscribing the intelligent event.

Item	Description	
Function	LLONG CALL_METHOD CLIENT_RealLoadPictureEx(LLONG lLoginID, int nChannelID, DWORD dwAlarmType, BOOL bNeedPicFile, fAnalyzer DataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);	
Parameter	[in] lLoginID	Login handle.
	[in] nChannelID	Device channel number.
	[in] dwAlarmType	Subscribed alarm events type.
	[in] bNeedPicFile	Whether to subscribe the picture file.
	[in] cbAnalyzerData	Intelligent event callback.
	[in] dwUser	Customized data type.
	[in] Reserved	Reserved field.
Return value	● Success: Subscription handle of LLONG type. ● Failure: 0.	
Note	Interface returns failed. Use CLIENT_GetLastError to obtain the error code.	

The following table shows the type of intelligent alarm event:

dwAlarmType macro definition	Macro definition value	Meaning	Call corresponding structure of pAlarmInfo
EVENT_IVS_ALL	0x00000001	All events	No
EVENT_IVS_CROSSFENCEDETECTION	0x0000011F	Crossing fence	DEV_EVENT_CROSSFENCEDETECTION_INFO
EVENT_IVS_CROSSLINEDETECTION	0x00000002	Tripwire invasion	DEV_EVENT_CROSSLINE_INFO
EVENT_IVS_CROSSREGIONDETECTION	0x00000003	Area invasion	DEV_EVENT_CROSSREGION_INFO
EVENT_IVS_LEFTDETECTION	0x00000005	Abandoned objects	DEV_EVENT_LEFT_INFO
EVENT_IVS_PRESERVATION	0x00000008	Safeguarding objects	DEV_EVENT_PRESERVATION_INFO
EVENT_IVS_TAKENAWAYDETECTION	0x00000115	moving objects,	DEV_EVENT_TAKENAWAYDETECTION_INFO
EVENT_IVS_WANDERDETECTION	0x00000007	Wandering events	DEV_EVENT_WANDER_INFO
EVENT_IVS_VIDEOABNORMALDETECTION	0x00000013	Video abnormal	DEV_EVENT_VIDEOABNORMALDETECTION_INFO
EVENT_IVS_AUDIO_ABNORMALDETECTION	0x00000126	Sound abnormal	DEV_EVENT_IVS_AUDIO_ABNORMALDETECTION_INFO

dwAlarmType macro definition	Macro definition value	Meaning	Call corresponding structure of pAlarmInfo
EVENT_IVS_CLIMBDETECTION	0x00000128	Climbing detection	DEV_EVENT_IVS_CLIMB_INFO
EVENT_IVS_FIGHTDETECTION	0x0000000E	Fighting detection	DEV_EVENT_FLOWSTAT_INFO
EVENT_IVS_LEAVEDETECTION	0x00000129	Off-post detection	DEV_EVENT_IVS_LEAVE_INFO
EVENT_IVS_PRISONERRISEDETECTION	0x0000011E	Detection on getting up	DEV_EVENT_PRISONERRISEDETECTION_INFO
EVENT_IVS_PASTEDETECTION	0x00000004	Detection on illegal sticky notes	DEV_EVENT_PASTE_INFO

Table 3-4

3.5.2 CLIENT_StopLoadPic

Item	Description	
Name	Stop subscription of intelligent events.	
Function	BOOL CALL_METHOD CLIENT_StopRealPlayEx(LLONG lRealHandle);	
Parameter	[in] lRealHandle	Subscription handle of intelligent event.
Return value	BOOL type: ● Success: TRUE. ● Failure: FALSE.	
Note	Interface returns failed. Use CLIENT_GetLastError to obtain the error code.	

3.6 Guest Flow Statistics

3.6.1 CLIENT_AttachVideoStatSummary

Item	Description	
Name	Start subscribing guest flow statistics.	
Function	LLONG CALL_METHOD CLIENT_AttachVideoStatSummary(LLONG lLoginID, Const NET_IN_ATTACH_VIDEOSTAT_SUM* pInParam, NET_OUT_ATTACH_VIDEOSTAT_SUM* pOutParam, int nWaitTime);	
Parameter	[in] lLoginID	Login handle

Item	Description	
	[in]pInParam	Input parameter of guest flow statistics.
	[out] pOutParam	Output parameter of guest flow statistics.
	[in]nWaitTime	Timeout period.
Return value	LLONG type: ● Success: Not 0. ● Failure: 0.	
Note	Interface returns failed. Use CLIENT_GetLastError to obtain the error code.	

3.6.2 CLIENT_DetachVideoStatSummary

Item	Description	
Name	Stop subscribing guest flow statistics.	
Function	BOOL CALL_METHOD CLIENT_DetachVideoStatSummary(LLONG lAttachHandle);	
Parameter	[in] lAttachHandle	Subscription handle.
Return value	BOOL type: ● Success: TRUE. ● Failure: FALSE.	
Note	Interface returns failed. Use CLIENT_GetLastError to obtain the error code.	

3.6.3 CLIENT_StartFindNumberStat

Item	Description	
Name	Start querying guest flow statistics.	
Function	LLONG CALL_METHOD CLIENT_StartFindNumberStat(LLONG lLoginID, NET_IN_FINDNUMBERSTAT* pstInParam, NET_OUT_FINDNUMBERSTAT* pstOutParam);	
Parameter	[in] lLoginID	Login handle.
	[in] pstInParam	Input parameter.
	[out] pstOutParam	Output parameter.
Return value	LLONG type: ● Success: Not 0 ● Failure: 0	
Note	Interface returns failed. Use CLIENT_GetLastError to obtain the error code.	

3.6.4 CLIENT_DoFindNumberStat

Item	Description
Name	Continue to query guest flow statistics.

Item	Description	
Function	int CALL_METHOD CLIENT_DoFindNumberStat(LLONG lFindHandle, NET_IN_DOFINDNUMBERSTAT* pstInParam, NET_OUT_DOFINDNUMBERSTAT* pstOutParam);	
Parameter	[in] lFindHandle	Login handle.
	[in]pstInParam	Input parameter.
	[out]pstOutParam	Output parameter.
Return value	int type: ● Success: 1. ● Failure: -1.	
Note	Interface returns failed. Use CLIENT_GetLastError to obtain the error code.	

3.6.5 CLIENT_StopFindNumberStat

Item	Description	
Name	Stop querying guest flow statistics.	
Function	BOOL CALL_METHOD CLIENT_StopFindNumberStat(LLONG lFindHandle);	
Parameter	[in] lFindHandle	Login handle.
Return value	BOOL type: ● Success: TRUE. ● Failure: FALSE.	
Note	Interface returns failed. Use CLIENT_GetLastError to obtain the error code.	

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 fAnalyzerDataCallBack

Item	Description	
Name	Intelligent event callback.	
Function	typedef int (CALLBACK *fAnalyzerDataCallBack)(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved);	
Parameter	[out]lAnalyzerHandle	Return value of CLIENT_RealLoadPictureEx.
	[out]dwAlarmType	Type of intelligent event.
	[out]pAlarmInfo	Cache of event information.
	[out]pBuffer	Cache of pictures.
	[out]dwBufSize	Cache size of pictures.
	[out]dwUser	User parameter of the callback.
	[out]reserved	Reserved.
Return value	None	
Note	None	

4.6 fVideoStatSumCallBack

Item	Description	
Name	Guest flow statistics callback	
Function	Typedef void (CALLBACK *fVideoStatSumCallBack) (LLONG lAttachHandle, NET_VIDEOSTAT_SUMMARY* pBuf, DWORD dwBufLen, LDWORD dwUser);	
Parameter	[in] lAttachHandle	Subscription handle.

Item	Description	
	[out] pBuf	Callback data.
	[out] dwBufLen	Callback data length.
	[out] dwUser	User parameter of the callback.
Return value	Void.	
Note	None.	