# NetSDK Intelligent Analysis Server

**Programming Manual**

V1.0.2

# Foreword

## Purpose

Welcome to use NetSDK (hereinafter referred to be "SDK") programming manual (hereinafter referred to as "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

## Safety Instructions

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

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

## Revision History

| Version | Revision Content | Release Time |
|---------|------------------|--------------|
| V1.0.3 | Change callback functions of login and device searching. | March 2020 |
| V1.0.1 | Delete some library files in "table 1-1." | January 2019 |
| V1.0.0 | First release. | December 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 prevail.
- We are not liable for any loss caused by the operations that do not comply with the manual.
- The manual would be updated according to the latest laws and regulations of related jurisdictions. For detailed information, refer to the paper manual, CD-ROM, QR code or our official website. If there is inconsistency between paper manual and the electronic version, the electronic version shall prevail.
- All the designs and software are subject to change without prior written notice. The product updates might cause some differences between the actual product and the manual. Please contact the customer service for the latest program and supplementary documentation.
- There still might be deviation in technical data, functions and operations description, or errors in print. If there is any doubt or dispute, we reserve the right of final explanation.
- Upgrade the reader software or try other mainstream reader software if the manual (in PDF format) cannot be opened.
- All trademarks, registered trademarks and the company names in the manual are the properties of their respective owners.
- Please visit our website, contact the supplier or customer service if there is any problem occurring when using the device.
- If there is any uncertainty or controversy, we reserve the right of final explanation.

# 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.

Table 1-1 Files of Windows development kit

| Library type | Library file name | Library file description |
|---|---|---|
| Function library | dhnetsdk.h | Header file |
| | dhnetsdk.lib | Lib file |
| | dhnetsdk.dll | Library file |
| | avnetsdk.dll | Library file |
| Configuration library | avglobal.h | Header file |
| | dhconfigsdk.h | Configuration Header file |
| | dhconfigsdk.lib | Lib file |
| | dhconfigsdk.dll | Library file |
| Auxiliary library of playing (coding and decoding) | dhplay.dll | Playing library |
| | 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 1-2 Files of Linux development kit

| Library type | Library file name | Library file description |
|---|---|---|
| Function library | dhnetsdk.h | Header file |
| | libdhnetsdk.so | Library file |
| | libavnetsdk.so | Library file |
| Configuration library | avglobal.h | Header file |
| | dhconfigsdk.h | Configuration Header file |
| | libdhconfigsdk.so | Configuration library |
| Auxiliary library of "libavnetsdk.so" | libInfra.so | Infrastructure library |
| | libNetFramework.so | Network infrastructure library |
| | libStream.so | Media transmission structure package library |
| | libStreamSvr.so | Streaming service |

📖

- 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

Table 2-1 Interfaces of SDK initialization

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

### 2.1.3 Process

Figure 2-1 Process of SDK initialization
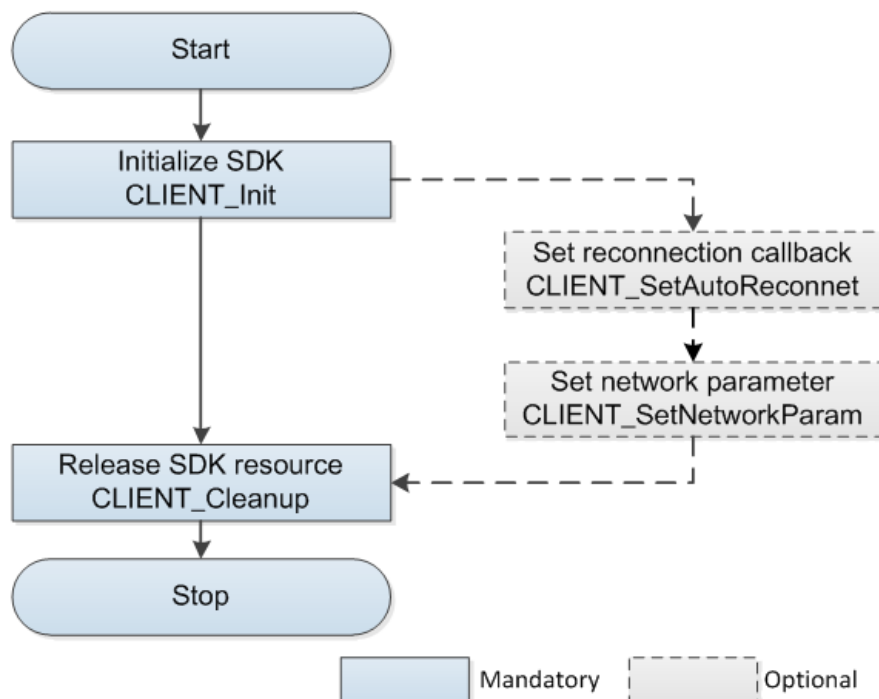
## Process Description

> Step 1  Call **CLIENT_Init** to initialize SDK.
>
> Step 2  (Optional) Call **CLIENT_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

Table 2-2 Interfaces of device initialization

| Interface | Implication |
|---|---|
| CLIENT_StartSearchDevicesEx | 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. |

## 2.2.3 Process

### 2.2.3.1 Device Initialization

Figure 2-2 Process of device initialization
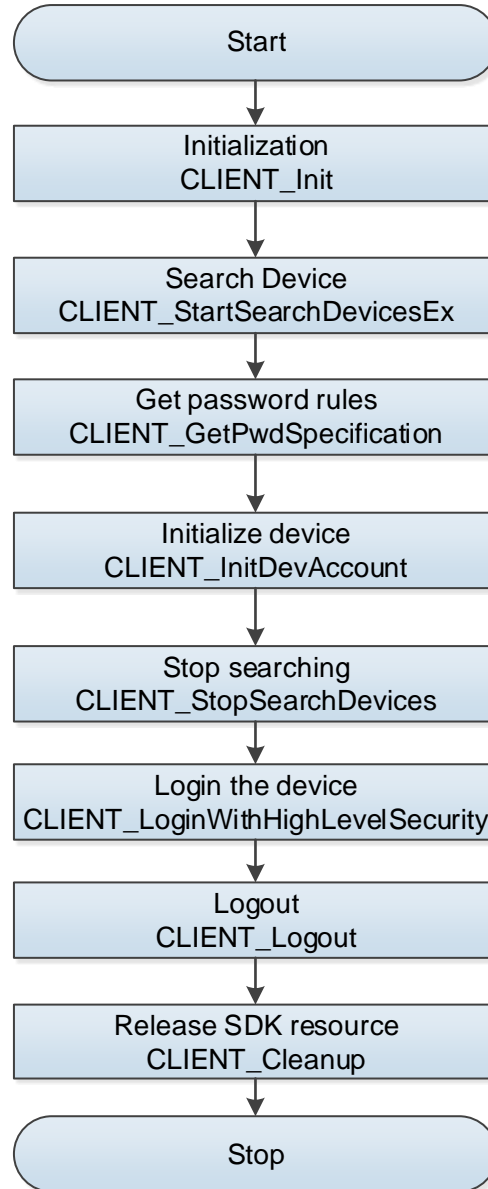
```
                    ┌─────────────────┐
                    │      Start      │
                    └─────────────────┘
                             │
                             ▼
                    ┌─────────────────┐
                    │  Initialization │
                    │   CLIENT_Init   │
                    └─────────────────┘
                             │
                             ▼
                  ┌─────────────────────────┐
                  │      Search Device      │
                  │ CLIENT_StartSearchDevicesEx │
                  └─────────────────────────┘
                             │
                             ▼
                  ┌─────────────────────────┐
                  │    Get password rules   │
                  │ CLIENT_GetPwdSpecification │
                  └─────────────────────────┘
                             │
                             ▼
                  ┌─────────────────────────┐
                  │     Initialize device   │
                  │  CLIENT_InitDevAccount  │
                  └─────────────────────────┘
                             │
                             ▼
                  ┌─────────────────────────┐
                  │      Stop searching     │
                  │ CLIENT_StopSearchDevices │
                  └─────────────────────────┘
                             │
                             ▼
                  ┌─────────────────────────────┐
                  │      Login the device       │
                  │ CLIENT_LoginWithHighLevelSecurity │
                  └─────────────────────────────┘
                             │
                             ▼
                  ┌─────────────────────────┐
                  │         Logout          │
                  │     CLIENT_Logout       │
                  └─────────────────────────┘
                             │
                             ▼
                  ┌─────────────────────────┐
                  │   Release SDK resource  │
                  │     CLIENT_Cleanup      │
                  └─────────────────────────┘
                             │
                             ▼
                    ┌─────────────────┐
                    │      Stop       │
                    └─────────────────┘
```

Process Description

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

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

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_LoginWithHighLevelSecurity** 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

Figure 2-3 Process of device initialization
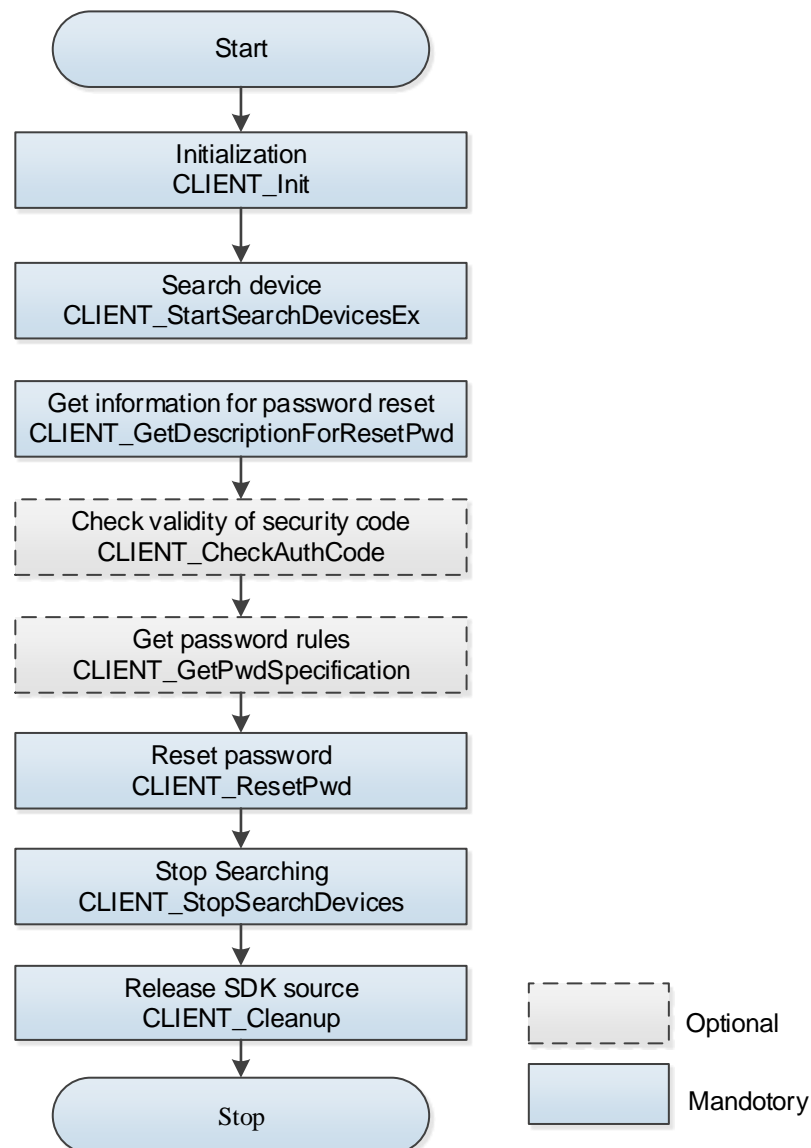


## Process Description

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

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

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_LoginWithHighLevelSecurity** 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_StartSearchDevicesEx 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_StartSearchDevicesEx 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 CLIENT_StartSearchDevice and CLIENT_StartSearchDeviceEx, and CLIENT_SearchDevicesByIPs)

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

Table 2-3 Interfaces of device login

| Interface | Implication |
| --- | --- |
| CLIENT_LoginWithHighLevelSecurity | Log in to the device with high level security. CLIENT_LoginEx2 can still be used,but there are security risks,so it is highly recommended to use the interface CLIENT_LoginWithHighLevelSecurity to log in to the device. |
| CLIENT_Logout | Logout. |

## 2.3.3 Process

Figure 2-4 Process of device login
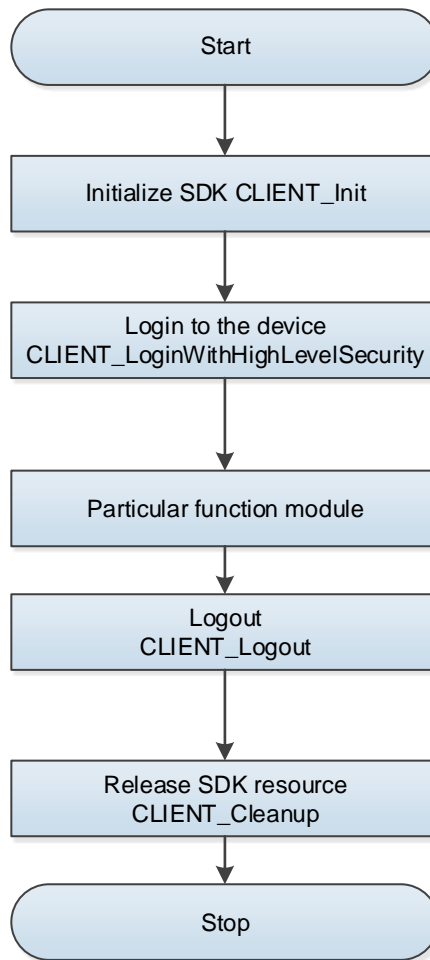
```
┌─────────────────────────────────────┐
│               Start                  │
└─────────────────────────────────────┘
                   │
                   ▼
┌─────────────────────────────────────┐
│        Initialize SDK CLIENT_Init    │
└─────────────────────────────────────┘
                   │
                   ▼
┌─────────────────────────────────────┐
│           Login to the device        │
│   CLIENT_LoginWithHighLevelSecurity  │
└─────────────────────────────────────┘
                   │
                   ▼
┌─────────────────────────────────────┐
│        Particular function module    │
└─────────────────────────────────────┘
                   │
                   ▼
┌─────────────────────────────────────┐
│               Logout                 │
│           CLIENT_Logout              │
└─────────────────────────────────────┘
                   │
                   ▼
┌─────────────────────────────────────┐
│         Release SDK resource         │
│           CLIENT_Cleanup             │
└─────────────────────────────────────┘
                   │
                   ▼
┌─────────────────────────────────────┐
│               Stop                   │
└─────────────────────────────────────┘
```

## Process Description

Step 1  Call **CLIENT_Init** to initialize SDK.
Step 2  Call **CLIENT_LoginWithHighLevelSecurity** 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.

Table 2-4 Common error code

| 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. |

For more information about error codes, see "CLIENT_LoginWithHighLevelSecurity interface" in *Network SDK Development Manual.chm*. The example code to avoid error code 3 is as follows.

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

## 2.3.4 Example Code

```
NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;
memset(&stInparam, 0, sizeof(stInparam));
stInparam.dwSize = sizeof(stInparam);
strncpy(stInparam.szIP, "192.168.1.108", sizeof(stInparam.szIP) - 1);
strncpy(stInparam.szPassword, "123456", sizeof(stInparam.szPassword) - 1);
strncpy(stInparam.szUserName, "admin", sizeof(stInparam.szUserName) - 1);
stInparam.nPort = 37777;
stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam;
memset(&stOutparam, 0, sizeof(stOutparam));
stOutparam.dwSize = sizeof(stOutparam);
LLONG lLoginID = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);
```

# 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

Table 2-5 Interfaces of real-time monitoring

| 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. |

## 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.

Figure 2-5 Process of playing by SDK decoding library
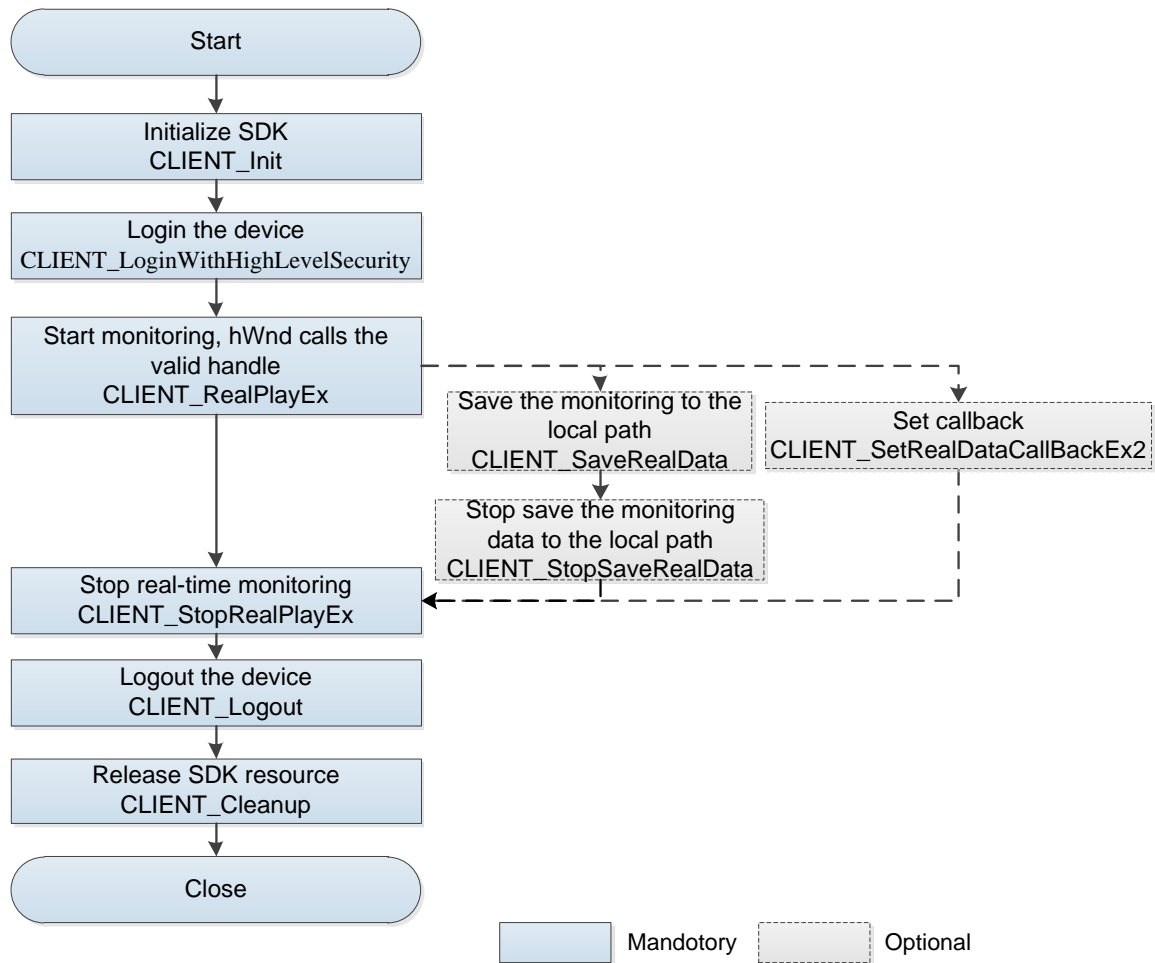


## Process Description

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

Step 2 Call **CLIENT_LoginWithHighLevelSecurity** 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.

Figure 2-6 Process of calling the third party play library
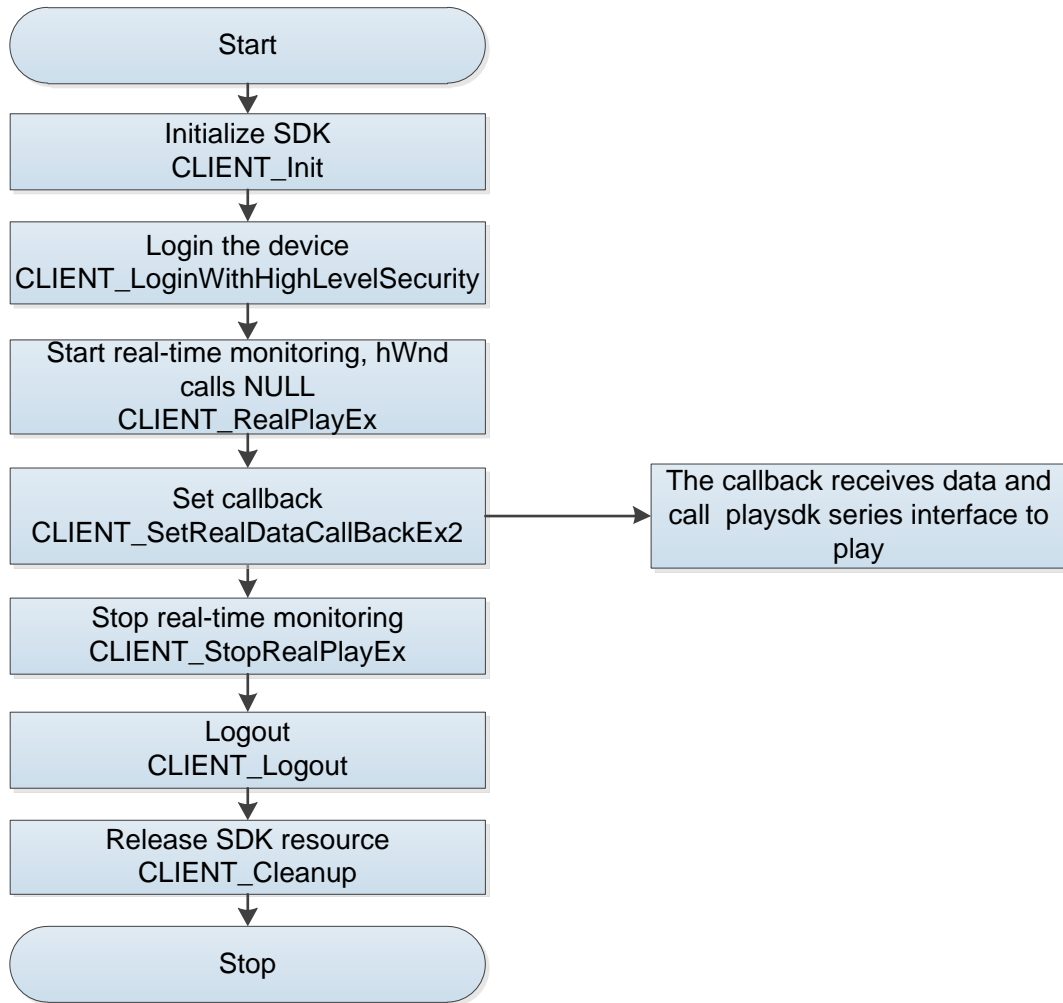


## Process Description

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

Step 2    Call **CLIENT_LoginWithHighLevelSecurity** 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.

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

Table 2-6 Behavior detecting supported by the DH-IVS-IP7200 series product

| 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. |

## 2.5.2 Interface Overview

Table 2-7 Interfaces of detection dedicated

| 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. |

## 2.5.3 Process

Figure 2-7 Process of subscribing intelligent events on behaviors analysis
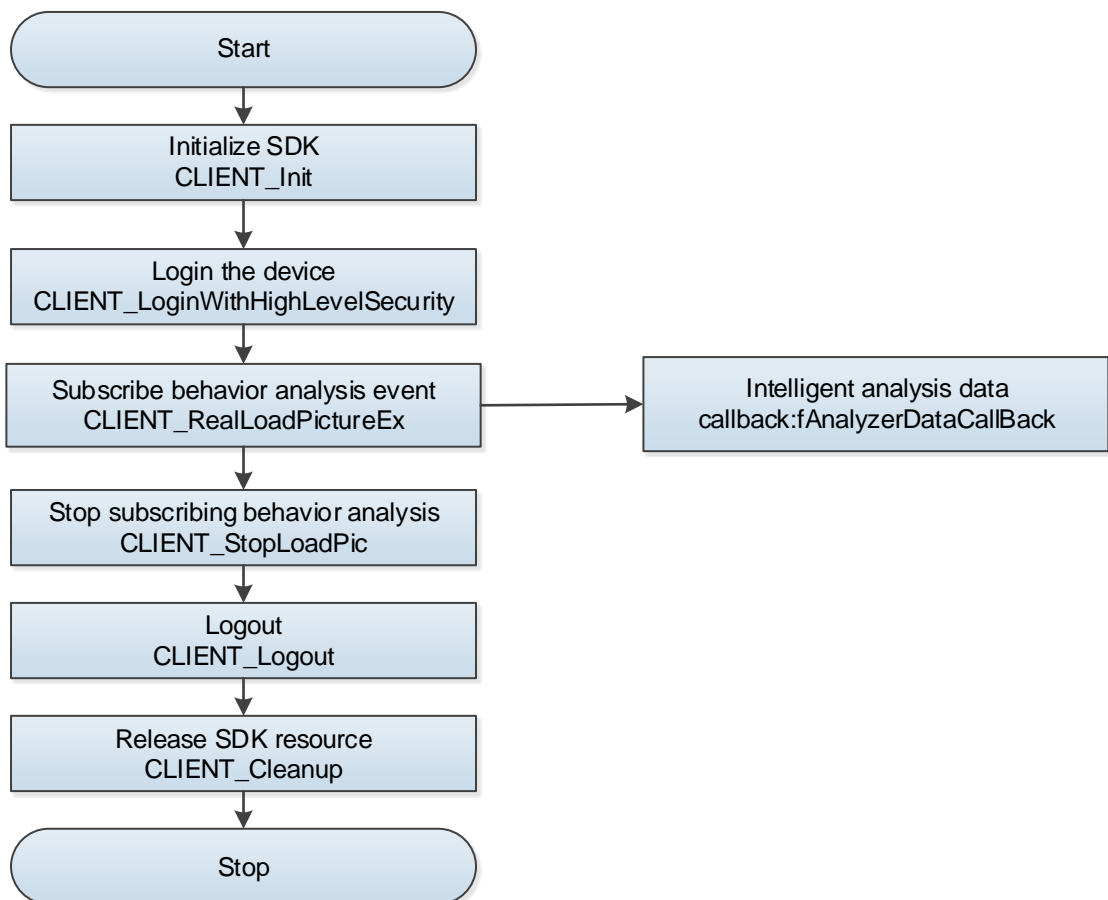
## Process Description

> Step 1  Call **CLIENT_Init** to initialize SDK.
> Step 2  Call **CLIENT_LoginWithHighLevelSecurity** 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:

        {
```

```c
                    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.

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.
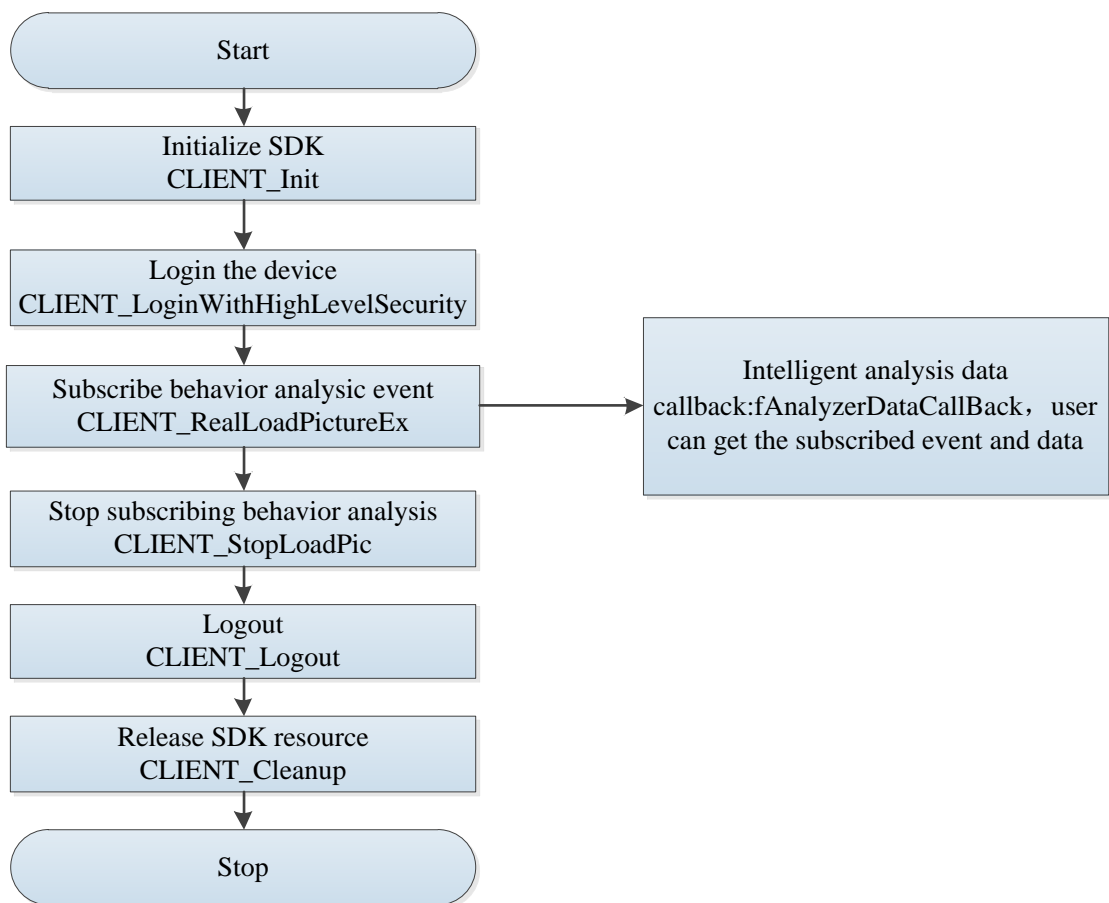
## 2.6.2 Interface Overview

Table 2-8 Interfaces of intelligent ATM

| 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. |

## 2.6.3 Process

Figure 2-8 Process of subscribing intelligent events on behaviors analysis in the intelligent ATM

```
                    ┌─────────────────────┐
                    │       Start         │
                    └─────────────────────┘
                              │
                    ┌─────────────────────┐
                    │   Initialize SDK    │
                    │   CLIENT_Init       │
                    └─────────────────────┘
                              │
                    ┌─────────────────────────────┐
                    │   Login the device          │
                    │ CLIENT_LoginWithHighLevelSecurity │
                    └─────────────────────────────┘
                              │
       ┌─────────────────────────────────┐      ┌──────────────────────────────────┐
       │ Subscribe behavior analysic event│─────▶│   Intelligent analysis data      │
       │   CLIENT_RealLoadPictureEx      │      │ callback:fAnalyzerDataCallBack,  │
       └─────────────────────────────────┘      │ user can get the subscribed event│
                              │                  │          and data                │
       ┌─────────────────────────────────┐      └──────────────────────────────────┘
       │ Stop subscribing behavior analysis│
       │   CLIENT_StopLoadPic            │
       └─────────────────────────────────┘
                              │
                    ┌─────────────────────┐
                    │      Logout         │
                    │   CLIENT_Logout     │
                    └─────────────────────┘
                              │
                    ┌─────────────────────┐
                    │  Release SDK resource│
                    │   CLIENT_Cleanup    │
                    └─────────────────────┘
                              │
                    ┌─────────────────────┐
                    │       Stop          │
                    └─────────────────────┘
```

## Process Description

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

Step 2 Call **CLIENT_LoginWithHighLevelSecurity** 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;
    }
```

```c
        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 Network diagram of guest flow statistics
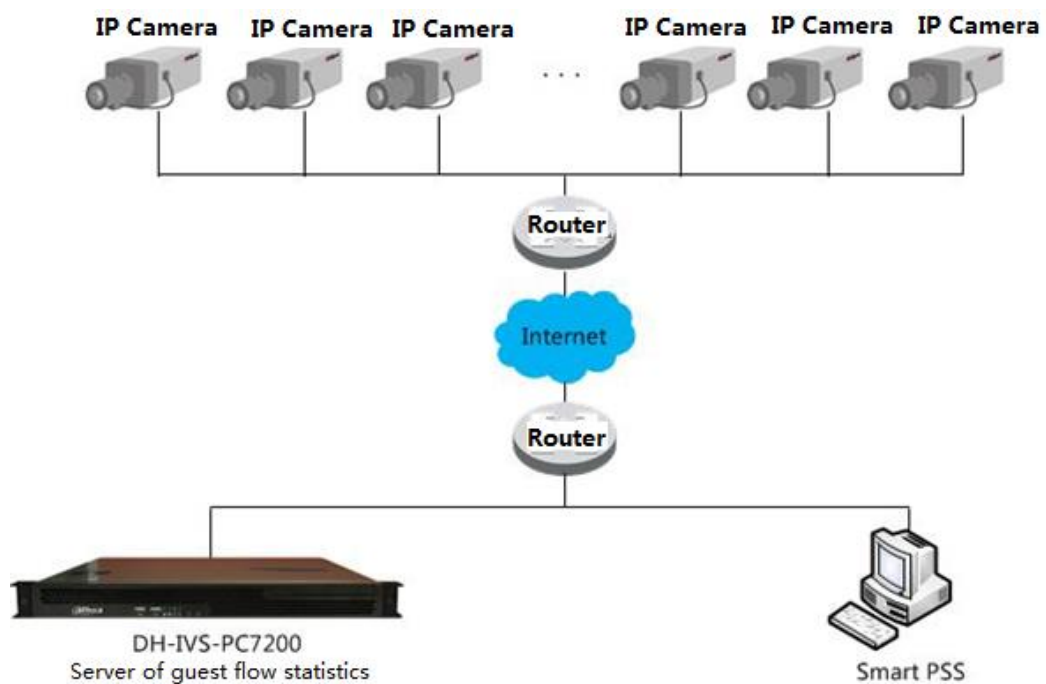


### Product Model

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.

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

Table 2-9 Interfaces of guest flow statistics

| 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. |

## 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

Figure 2-10 Process of SDK real-time guest flow statistics
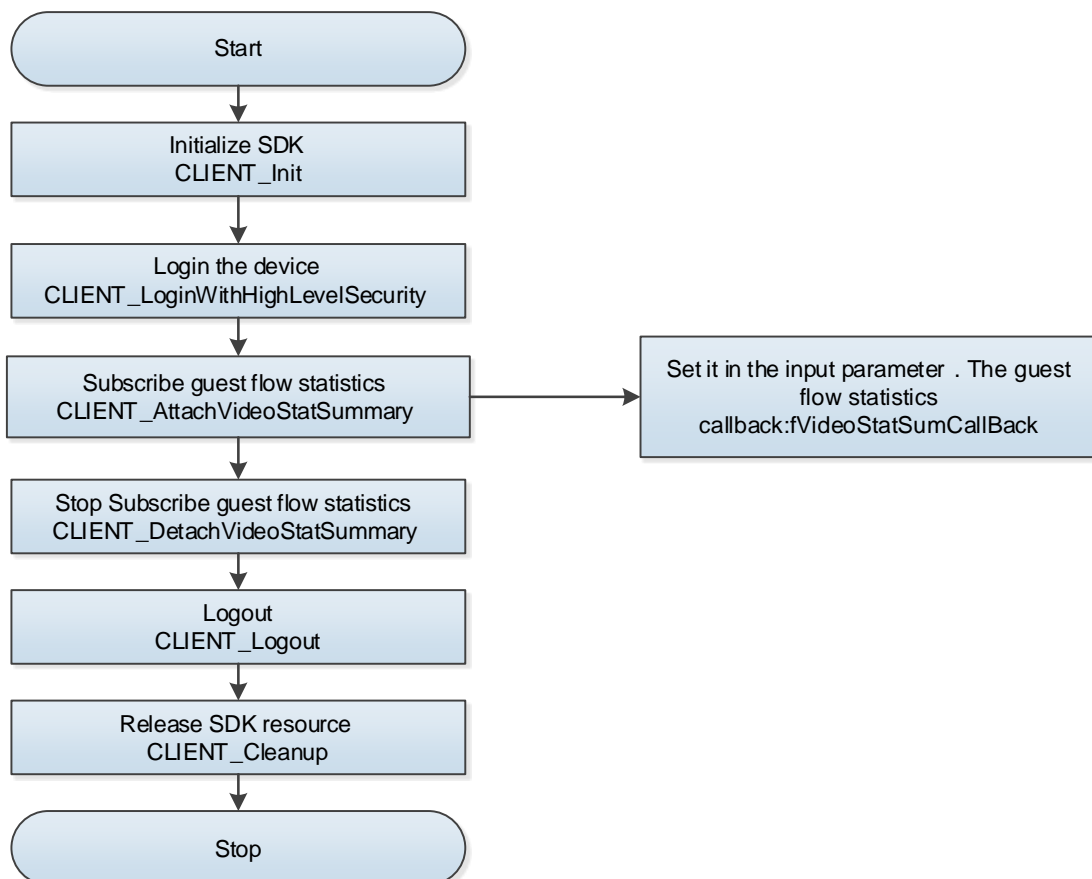
## Process Description

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

Step 2　Call **CLIENT_LoginWithHighLevelSecurity** 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

Figure 2-11 Process of querying historical guest flow statistics
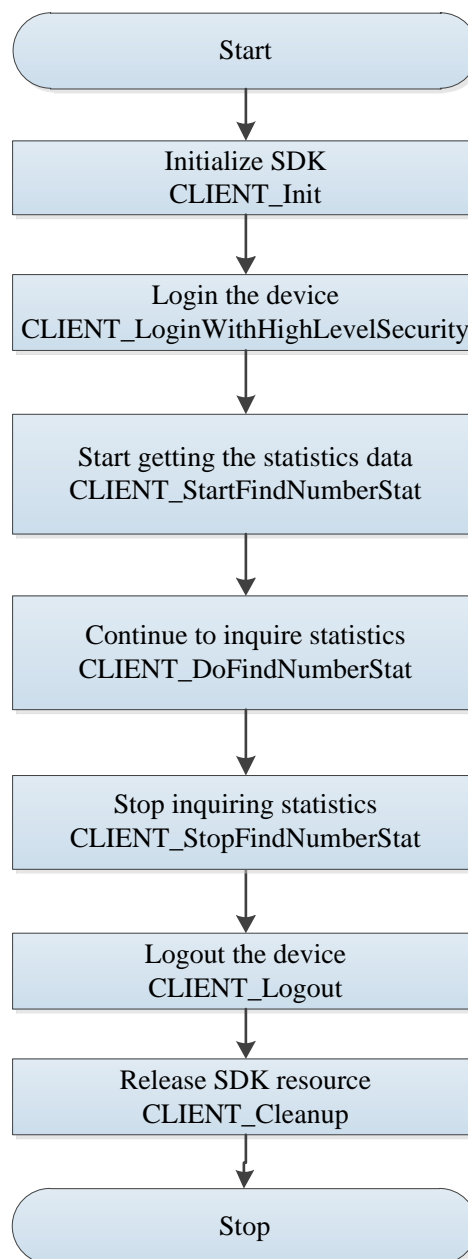
## Process Description

# 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->stuExite
dSubtotal.nTotal);

}

int main(int argc, char* argv[])

{

    // Initialization

    CLIENT_Init(NULL,0);

     .........

    // Login

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;

    memset(&stInparam, 0, sizeof(stInparam));

    stInparam.dwSize = sizeof(stInparam);

    strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);
```

```
    strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);

    strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);

    stInparam.nPort = g_nPort;

    stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;

    LLONG lLoginID = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);


    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->stuExite
dSubtotal.nTotal);

}

int main(int argc, char* argv[])

{

    // Initialization

    CLIENT_Init(NULL,0);

     .........

    // Login

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam;

    memset(&stInparam, 0, sizeof(stInparam));

    stInparam.dwSize = sizeof(stInparam);

    strncpy(stInparam.szIP, g_szDevIp, sizeof(stInparam.szIP) - 1);

    strncpy(stInparam.szPassword, g_szPasswd, sizeof(stInparam.szPassword) - 1);

    strncpy(stInparam.szUserName, g_szUserName, sizeof(stInparam.szUserName) - 1);

    stInparam.nPort = g_nPort;

    stInparam.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
```

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


    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];
```

```c
        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

Table 3-1 Initialize SDK

| Item | Description | |
|------|-------------|---|
| Name | Initialize SDK. | |
| Function | BOOL CLIENT_Init(<br>    fDisConnect   cbDisConnect,<br>    LDWORD      dwUser<br>); | |
| Parameter | [in]cbDisConnect | Disconnection callback. |
| | [in]dwUser | User parameter of disconnection callback. |
| Return value | ● Success: TRUE.<br>● Failure: FALSE. | |
| Note | ● The precondition for calling other function modules of SDK.<br>● 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

Table 3-2 Clean up SDK

| 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

Table 3-3 Set reconnection callback

| Item | Description | |
|------|-------------|---|
| Name | Set auto reconnection for callback. | |
| Function | void CLIENT_SetAutoReconnect(<br>    fHaveReConnect     cbAutoConnect,<br>    LDWORD              dwUser<br>); | |
| 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

Table 3-4 Set network parameter

| Item | Description | |
|---|---|---|
| Name | Set the related parameters for network environment. | |
| Function | void CLIENT_SetNetworkParam( <br> NET_PARAM    *pNetParam <br> ); | |
| 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_StartSearchDevicesEx

Table 3-5 Search for device

| Item | Description | |
|---|---|---|
| Name | Search the device. | |
| Function | LLONG CLIENT_StartSearchDevicesEx ( <br> NET_IN_STARTSERACH_DEVICE*      pInBuf, <br> NET_OUT_STARTSERACH_DEVICE*    pOutBuf <br> ); | |
| Parameter | [in] pInBuf | Output parameter. Refer to NET_IN_STARTSERACH_DEVICE |
| | [out] pOutBuf | Output parameter. Refer to NET_OUT_STARTSERACH_DEVICE |
| Return value | Searching handle. | |
| Note | Multi-thread calling is not supported. | |

## 3.2.2 CLIENT_InitDevAccount

Table 3-6 Initialize device

| Item | Description |
|---|---|
| Name | Initialize the device. |

| Item | Description |
|---|---|
| Function | BOOL CLIENT_InitDevAccount(<br>    const NET_IN_INIT_DEVICE_ACCOUNT    *pInitAccountIn,<br>    NET_OUT_INIT_DEVICE_ACCOUNT    *pInitAccountOut,<br>    DWORD    dwWaitTime,<br>    char    *szLocalIp<br>); |
| 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.<br>● In case of multiple network card, enter the IP of the host PC for the last parameter. |
| Return value | ● Success: TRUE.<br>● Failure: FALSE. |
| Note | None. |

## 3.2.3 CLIENT_GetDescriptionForResetPwd

Table 3-7 Get information for password reset

| Name | Description |
|---|---|
| Name | Get information for password reset. |
| Function | BOOL CLIENT_GetDescriptionForResetPwd(<br>    const NET_IN_DESCRIPTION_FOR_RESET_PWD    *pDescriptionIn,<br>    NET_OUT_DESCRIPTION_FOR_RESET_PWD    *pDescriptionOut,<br>    DWORD    dwWaitTime,<br>    char    *szLocalIp<br>); |
| 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.<br>● In case of multiple network card, enter the IP of the host PC for the last parameter. |
| Return value | ● Success: TRUE.<br>● Failure: FALSE. |
| Note | None. |

## 3.2.4 CLIENT_CheckAuthCode

Table 3-8 Check the validity of security code

| Item | Description | |
|---|---|---|
| Name | Check the validity of security code. | |
| Function | BOOL CLIENT_CheckAuthCode(<br>    const NET_IN_CHECK_AUTHCODE    *pCheckAuthCodeIn,<br>    NET_OUT_CHECK_AUTHCODE       *pCheckAuthCodeOut,<br>    DWORD                         dwWaitTime,<br>    char                            *szLocalIp<br>); | |
| 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.<br>● In case of multiple network card, enter the IP of the host PC for the last parameter. |
| Return value | ● Success: TRUE.<br>● Failure: FALSE. | |
| Note | None. | |

## 3.2.5 CLIENT_ResetPwd

Table 3-9 Reset the password

| Item | Description | |
|---|---|---|
| Name | Reset the password. | |
| Function | BOOL CLIENT_ResetPwd(<br>    const NET_IN_RESET_PWD         *pResetPwdIn,<br>    NET_OUT_RESET_PWD          *pResetPwdOut,<br>    DWORD                       dwWaitTime,<br>    char                        *szLocalIp<br>); | |
| 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.<br>● In case of multiple network card, enter the IP of the host PC for the last parameter. |
| Return value | ● Success: TRUE.<br>● Failure: FALSE. | |

| Item | Description |
|------|-------------|
| Note | None. |

## 3.2.6 CLIENT_GetPwdSpecification

Table 3-10 Get password rules

| Item | Description | |
|------|-------------|---|
| Name | Get password rules. | |
| Function | BOOL CLIENT_GetPwdSpecification(<br>    const NET_IN_PWD_SPECI       *pPwdSpeciIn,<br>    NET_OUT_PWD_SPECI       *pPwdSpeciOut,<br>    DWORD               dwWaitTime,<br>    char                 *szLocalIp<br>); | |
| 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.<br>● In case of multiple network card, enter the IP of the host PC for the last parameter. |
| Return value | ● Success: TRUE.<br>● Failure: FALSE. | |
| Note | None. | |

## 3.2.7 CLIENT_StopSearchDevices

Table 3-11 Stop searching device

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

# 3.3 Device Login

## 3.3.1 CLIENT_LoginWithHighLevelSecurity

Table 3-12 Log in with high level security

| Item | Description | |
|------|-------------|---|
| Name | Login the device with high level security. | |
| Function | LLONG   CLIENT_LoginWithHighLevelSecurity (<br>NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY*   pstInParam,<br>NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY* pstOutParam<br>); | |
| Parameter | [in] pstInParam | [in] dwSize |
| | | [in] szIP |
| | | [in] nPort |
| | | [in] szUserName |
| | | [in] szPassword |
| | | [in] emSpecCap |
| | | [in] pCapParam |
| | [out] pstOutParam | [in]dwSize |
| | | [out] stuDeviceInfo |
| | | [out] nError |
| Return value | ● Success: Not 0.<br>● Failure: 0. | |
| Note | Login the device with high level security.<br>CLIENT_LoginEx2 can still be used,but there are security risks,so it is highly recommended to use the latest interface CLIENT_LoginWithHighLevelSecurity to log in to the device. | |

Table 3-13 Error code and meaning

| Error code | 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. |

## 3.3.2 CLIENT_Logout

Table 3-14 Log out

| Item | Description | |
|------|-------------|---|
| Name | Logout the device | |
| Function | BOOL CLIENT_Logout(<br>    LLONG       lLoginID<br>); | |
| Parameter | [in]lLoginID | Return value of CLIENT_LoginWithHighLevelSecurity |
| Return value | ●    Success: TRUE<br>●    Failure: FALSE | |
| Note | None | |

# 3.4 Real-time Monitoring

## 3.4.1 CLIENT_RealPlayEx

Table 3-15 Start the real-time monitoring

| Item | Description | |
|------|-------------|---|
| Name | Open the real-time monitoring. | |
| Function | LLONG CLIENT_RealPlayEx(<br>    LLONG             lLoginID,<br>    int               nChannelID,<br>    HWND         hWnd,<br>    DH_RealPlayType   rType<br>); | |
| Parameter | [in]lLoginID | Return value of CLIENT_LoginWithHighLevelSecurity. |
|  | [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.<br>●    Failure: 0. | |
| Note | Windows system:<br>●    When hWnd is valid, the corresponding window displays picture.<br>●    When hWnd is NULL, get the video data through setting a callback and send to user for handle. | |

Table 3-16 Live view type and meaning

| Preview type | Meaning |
|--------------|---------|
| DH_RType_Realplay | Real-time preview |
| DH_RType_Multiplay | Multi-picture preview |

| Preview type | Meaning |
| --- | --- |
| 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 |
| 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 |

## 3.4.2 CLIENT_StopRealPlayEx

Table 3-17 Stop the real-time monitoring

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

## 3.4.3 CLIENT_SaveRealData

Table 3-18 Save the real-time monitoring data as file

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

# 3.4.4 CLIENT_StopSaveRealData

Table 3-19 Stop saving the real-time monitoring data as file

| Item | Description | |
|---|---|---|
| Name | Stop saving the real-time monitoring data as file | |
| Function | BOOL CLIENT_StopSaveRealData(<br>    LLONG           lRealHandle<br>); | |
| Parameter | [in] lRealHandle | Return value of CLIENT_RealPlayEx |
| Return value | ●   Success: TRUE<br>●   Failure: FALSE | |
| Note | None | |

# 3.4.5 CLIENT_SetRealDataCallBackEx2

Table 3-20 Set the callback of real-time monitoring data

| Item | Description | |
|---|---|---|
| Name | Set the callback of real-time monitoring data | |
| Function | BOOL CLIENT_SetRealDataCallBackEx2(<br>    LLONG              lRealHandle,<br>    fRealDataCallBackEx2   cbRealData,<br>    LDWORD            dwUser,<br>    DWORD             dwFlag<br>); | |
| 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<br>●   Failure: FALSE | |
| Note | None | |

Table 3-21 dwFlag type and parameter

| 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. |

# 3.5 Subscription of Intelligent Event

## 3.5.1 CLIENT_RealLoadPictureEx

Table 3-22 Start subscribing the intelligent event

| Item | Description | |
|------|-------------|--|
| Name | Start subscribing the intelligent event. | |
| Function | LLONG CALL_METHOD CLIENT_RealLoadPictureEx(<br>    LLONG            lLoginID,<br>    int              nChannelID,<br>    DWORD            dwAlarmType,<br>    BOOL             bNeedPicFile,<br>    fAnalyzer             DataCallBack cbAnalyzerData,<br>    LDWORD           dwUser,<br>    void*            Reserved<br>); | |
| 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.<br>● Failure: 0. | |
| Note | Interface returns failed. Use CLIENT_GetLastError to obtain the error code. | |

Table 3-23 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 |

| dwAlarmType macro definition | Macro definition value | Meaning | Call corresponding structure of pAlarmInfo |
|---|---|---|---|
| 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 |
| 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 |

## 3.5.2 CLIENT_StopLoadPic

Table 3-24 Stop subscription of intelligent events

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

# 3.6 Guest Flow Statistics

## 3.6.1 CLIENT_AttachVideoStatSummary

Table 3-25 Start subscribing guest flow statistics

| Item | Description | |
|------|-------------|---|
| Name | Start subscribing guest flow statistics. | |
| Function | LLONG CALL_METHOD CLIENT_AttachVideoStatSummary(<br>        LLONG                                    lLoginID,<br>        Const NET_IN_ATTACH_VIDEOSTAT_SUM*    pInParam,<br>        NET_OUT_ATTACH_VIDEOSTAT_SUM*        pOutParam,<br>        int                                      nWaitTime<br>); | |
| Parameter | [in] lLoginID | Login handle |
| | [in]pInParam | Input parameter of guest flow statistics. |
| | [out] pOutParam | Output parameter of guest flow statistics. |
| | [in]nWaitTime | Timeout period. |
| Return value | LLONG type:<br>● Success: Not 0.<br>● Failure: 0. | |
| Note | Interface returns failed. Use CLIENT_GetLastError to obtain the error code. | |

## 3.6.2 CLIENT_DetachVideoStatSummary

Table 3-26 Stop subscribing guest flow statistics

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

## 3.6.3 CLIENT_StartFindNumberStat

Table 3-27 Start querying guest flow statistics

| Item | Description |
|------|-------------|
| Name | Start querying guest flow statistics. |

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

## 3.6.4 CLIENT_DoFindNumberStat

Table 3-28 Continue to query guest flow statistics

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

## 3.6.5 CLIENT_StopFindNumberStat

Table 3-29 Stop querying guest flow statistics

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

# 4 Callback Definition

## 4.1 fSearchDevicesCB

Table 4-1 Callback of searching devices

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

## 4.2 fSearchDevicesCBEx

Table 4-2 Callback of searching devices

| Item | Description | |
|---|---|---|
| Name | Callback of searching devices. | |
| Function | typedef void(CALLBACK * fSearchDevicesCBEx)(<br>    LLONG     lSearchHandle,<br>    DEVICE_NET_INFO_EX2   *pDevNetInfo,<br>    void*     pUserData<br>); | |
| Parameter | [out] lSearchHandle | Search Handle |
| | [out]pDevNetInfo | The searched device information. |
| | [out]pUserData | User data. |
| Return value | None. | |
| Note | None. | |

## 4.3 fDisConnect

Table 4-3 Disconnection callback

| Item | Description |
|---|---|
| Name | Disconnection callback. |
| Function | typedef void (CALLBACK *fDisConnect)(<br>    LLONG     lLoginID,<br>    char     *pchDVRIP,<br>    LONG     nDVRPort, |

| Item | Description | |
|------|-------------|---|
| | LDWORD    dwUser<br>); | |
| Parameter | [out] lLoginID | Return value of<br>CLIENT_LoginWithHighLevelSecurity. |
| | [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. | |

## 4.4 fHaveReConnect

Table 4-4 Reconnection callback

| Item | Description | |
|------|-------------|---|
| Name | Reconnection callback. | |
| Function | typedef void (CALLBACK *fHaveReConnect)(<br>    LLONG        lLoginID,<br>    char          *pchDVRIP,<br>    LONG        nDVRPort,<br>    LDWORD    dwUser<br>); | |
| Parameter | [out] lLoginID | Return value of<br>CLIENT_LoginWithHighLevelSecurity. |
| | [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. | |

## 4.5 fRealDataCallBackEx2

Table 4-5 Callback of real-time monitoring data

| Item | Description |
|------|-------------|
| Name | Callback of real-time monitoring data. |
| Function | typedef void (CALLBACK *fRealDataCallBackEx2)(<br>    LLONG        lRealHandle,<br>    DWORD      dwDataType,<br>    BYTE        *pBuffer,<br>    DWORD      dwBufSize,<br>    LLONG         param,<br>    LDWORD      dwUser |

| Item | Description | |
|---|---|---|
| | ); | |
| Parameter | [out] lRealHandle | Return value of CLIENT_RealPlayEx. |
| | [out] dwDataType | Data type:<br>● 0: Initial data.<br>● 1: Data with frame information.<br>● 2: YUV data.<br>● 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.<br>● The param is blank pointer when dwDataType is 0.<br>● The param is the pointer of tagVideoFrameParam structure when dwDataType is 1.<br>● The param is the pointer of tagCBYUVDataParam structure when dwDataType is 2.<br>● The param is the pointer of tagCBPCMDataParam structure when dwDataType is 3. |
| | [out] dwUser | User parameter of the callback. |
| Return value | None. | |
| Note | None. | |

# 4.6 fAnalyzerDataCallBack

Table 4-6 Intelligent event callback

| Item | Description | |
|---|---|---|
| Name | Intelligent event callback. | |
| Function | typedef int (CALLBACK *fAnalyzerDataCallBack)(<br>    LLONG              lAnalyzerHandle,<br>    DWORD           dwAlarmType,<br>    void*              pAlarmInfo,<br>    BYTE            *pBuffer,<br>    DWORD           dwBufSize,<br>    LDWORD         dwUser,<br>    int                nSequence,<br>    void              *reserved<br>); | |
| Parameter | [out]lAnalyzerHandle | Return value of CLIENT_RealLoadPictureEx. |
| | [out]dwAlarmType | Type of intelligent event. |
| | [out]pAlarmInfo | Cache of event information. |

| Item | Description | |
|------|-------------|---|
| | [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 | |

# 4.7 fVideoStatSumCallBack

Table 4-7 Guest flow statistics callback

| Item | Description | |
|------|-------------|---|
| Name | Guest flow statistics callback. | |
| Function | Typedef    void (CALLBACK *fVideoStatSumCallBack) (<br>      LLONG                                    lAttachHandle,<br>      NET_VIDEOSTAT_SUMMARY*           pBuf,<br>      DWORD                                  dwBufLen,<br>      LDWORD                                 dwUser<br>); | |
| Parameter | [in] lAttachHandle | Subscription handle. |
| | [out] pBuf | Callback data. |
| | [out] dwBufLen | Callback data length. |
| | [out] dwUser | User parameter of the callback. |
| Return value | None. | |
| Note | None. | |

# Appendix 1 Cybersecurity Recommendations

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

**Mandatory actions to be taken for basic equipment network security:**

1. **Use Strong Passwords**

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

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

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

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

1. **Physical Protection**

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

2. **Change Passwords Regularly**

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

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

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

4. **Enable Account Lock**

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

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

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

6. **Enable HTTPS**

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

7. **Enable Whitelist**

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

8. **MAC Address Binding**

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

9. **Assign Accounts and Privileges Reasonably**

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

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

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

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

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

11. **Audio and Video Encrypted Transmission**

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

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

12. **Secure Auditing**

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

13. **Network Log**

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

14. **Construct a Safe Network Environment**

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

- Disable the port mapping function of the router to avoid direct access to the intranet devices from external network.
- The network should be partitioned and isolated according to the actual network needs. If there are no communication requirements between two sub networks, it is

suggested to use VLAN, network GAP and other technologies to partition the network, so as to achieve the network isolation effect.

● Establish the 802.1x access authentication system to reduce the risk of unauthorized access to private networks.

● It is recommended that you enable your device's firewall or blacklist and whitelist feature to reduce the risk that your device might be attacked.