NetSDK (Intelligent Traffic)

Programming Manual

V1.0.7

Foreword

Purpose

Welcome to use NetSDK (hereinafter referred to as "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 Traffic Camera (ITC), Intelligent Traffic System (ITSE), and IPMECK. 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.

Readers

- SDK software development engineers
- Project managers
- Product managers

Safety Instruction

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
V1.0.7	● Added ANPR vehicle flow historical data search, intelligent event subscription and video and image search, playback and download. ● Added device configuration, including auto registration, device logs, getting remote device information and importing and exporting configuration information. ● Modified interface functions of 3.2.5 "Subsribing to Intelligent Event", 3.2.6 "Intelligent Traffic" 3.2.7 "Searching for and Downloading Intelligent Event Videos or Images" and 3.4 "Device Configurations". ● Added Intelligent Traffic Event Macro.	August, 2021
V1.0.6	● Added Dot-matrix Display control and voice broadcast ● Deleted fisheye dewarping library.	June, 2021
V1.0.5	● Deleted the dependent library of avnetsdk. ● Added the dependent library of Convertor.	March 2021
V1.0.4	Change the callback functions of login and device searching.	February 2020
V1.0.3	● Deleted"2.3.7 Parking Space Status Indicator Configuration" and"2.3.7 Parking Space Status Indicator Configuration". ● Changed the name of Parking Space Linking Barrier Control Process flowchart.	December, 2019
V1.0.2	Added "2.3 Parking Lot", "3.3 Parking Lot", "4.8 fTransFileCallBack" and "4.9 pfAudioDataCallBack".	October 2019
V1.0.1	Deleted some library files in "Table 1-1".	January 2019
V1.0.0	First release.	December, 2017

As the device user or data controller, you might collect personal data of others such as face, fingerprints, car plate number, email address, phone number, GPS and so on. You need to be in compliance with the local privacy protection laws and regulations to protect the legitimate rights and interests of other people by implementing measures include but not limited to: providing clear and visible identification to inform data subject the existence of surveillance area and providing related contact.

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

Term	Definition
ITC	Intelligent Traffic Camera, which is featured by capturing pictures of vehicles and automatically analyzing the traffic events.
ITSE	Intelligent Traffic System, also named as intelligent box, is connected with ITC to provide store the pictures and analyzed data.
IPMECK	Controls the opening and closing of barrier.
Login Handle	A kind of handle that connects with ITC, ITSE and IPMECK. If the connection is successful, the handle is not null (32-bit 4 bytes, 64-bit 8 bytes). This handle is used in most of function modules and will not be null till logged out.
Video Channel	The video of ITC or ITSE is expressed by channel ID. The single lens ITC has only one channel, and multi-lens ITC and ITSE have multiple channels.
Query Handle	A kind of handle that sends query request to ITSE. If the request is successful, the handle is not null (32-bit 4 bytes, 64-bit 8 bytes). It is used to query a particular function module and will not be null until logged out.
Media File	The picture captured by ITC and will be identified and analyzed automatically.
Intelligent Capture	The user needs to capture some scenarios manually. The device analyzes the captured pictures and sends the results to the user.
Intelligent Traffic Event	When the vehicle is passing the traffic junction or the capturing range, the ITC will capture and analyze send the pictures, and then send the results to the user.
Traffic Junction	The traffic junction where the device capture each passing vehicle. The device will analyze and identify the captured pictures and send the results to the user.
Open Barrier Gate	On the traffic junction installed with IPMECK and barrier gate, open the barrier gate to let the vehicle go through the control of IPMECK.
Close Barrier Gate	On the traffic junction installed with IPMECK and barrier gate, close the barrier gate to let the vehicle go through the control of IPMECK.

Table of Contents

Foreword ..... **I**
Glossary ..... **IV**
1 Overview **1**
1.1 Introduction 1
1.2 Applicability 2
1.3 Application 2
2 Function Modules **5**
2.1 General 5
2.1.1 SDK Initialization 5
2.1.2 Device Initialization 7
2.1.3 Device Login 12
2.1.4 Real-time Monitoring 15
2.2 Traffic Junction 20
2.2.1 Download of Media File 20
2.2.2 Manual Capture 25
2.2.3 Upload of Intelligent Traffic Event 28
2.2.4 Vehicle Flow Statistics 31
2.2.5 Searching for Historcial Traffic Flow Data 33
2.2.6 Subscribing to Intelligent Event 36
2.2.7 Video and Image Search/Playback/Download 38
2.3 Parking Lot 44
2.3.1 Barrier Control 44
2.3.2 Importing/Exporting Blocklist/Allowlist 51
2.3.3 Voice talk 54
2.3.4 Dot-matrix Display Content Control and Broadcast 59
2.3.5 Dot-matrix Display Character Control 62
2.3.6 Parking Space Indicator Configuration 64
2.3.7 Parking Space Status Indicator Configuration 67
2.4 Device Configuration 69
2.4.1 Auto registration 69
2.4.2 Device Logs 71
2.4.3 Get Remote Device Information 75
2.4.4 Importing and Exporting Configuration Information 78
3 Interface Definition **91**
3.1 General Interfaces 91
3.1.1 SDK Initialization 91
3.1.2 Device Initialization 92
3.1.3 Device Login 96
3.1.4 Real-time Monitoring 97
3.2 Traffic Junction 100
3.2.1 Download of Medial File 100
3.2.2 Manual Capture 103
3.2.3 Upload of Intelligent Traffic Event 104
3.2.4 Vehicle Flow Statistics 107

3.2.5 Intelligent Traffic 107
3.2.6 Searching for and Downloading Intelligent Event Videos or Images 110
3.3 Parking Lot 114
3.3.1 Barrier Control 114
3.3.2 Importing/Exporting Allowlist/Blocklist: CLIENT_FileTransmit 117
3.3.3 Voice Talk 118
3.3.4 Dot-matrix Display Content Control and Broadcast 121
3.3.5 Dot-matrix Display Character Control 121
3.3.6 Parking Space Indicator Configuration 122
3.3.7 Parking Space Status Indicator Configuration 124
3.4 Device Configuration 124
3.4.1 Auto Registration 124
3.4.2 Viewing Device Information 128
3.4.3 Importing and Exporting Configuration Information 131
4 Callback Definition 133
4.1 fSearchDevicesCB 133
4.2 fSearchDevicesCBEx 133
4.3 fDisConnect 133
4.4 fHaveReConnect 134
4.5 fRealDataCallBackEx2 134
4.6 fDownLoadPosCallBack 135
4.7 fAnalyzerDataCallBack 135
4.8 fFluxStatDataCallBack 136
4.9 fTransFileCallBack 137
4.10 pfAudioDataCallBack 137
4.11 fDataCallBack 138
4.12 fTimeDownLoadPosCallBack 139
4.13 fDownLoadPosCallBack 139
4.14 fCameraStateCallBack 140
5 Intelligent Traffic Event Macro 141
Appendix 1 Cybersecurity Recommendations 144

1 Overview

1.1 Introduction

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, download of intelligent images, manual capture, report of intelligent traffic event, vehicle flow statistics, and barrier control.

The development kit might be different dependent on the environment.

Table 1-1 Files included in Windows development kit

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

Table 1-2 files included in Linux development kit

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

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

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

1.2 Applicability

- Recommended memory: No less than 512 M.
- System supported by SDK:
  - ◇ Windows
    Windows 10/Windows 8.1/Windows 7 and Windows Server 2008/2003
  - ◇ Linux
    The common Linux systems such as Red Hat/SUSE
- Access ANPR cameras and other traffic devices:
  ITSE1604-GN5A-D Series, ITSE0400-GN5A-B Series, ITSE0804-GN5B-D Series
- Devices in parking lots:
  Access ANPR camera: ITC215-PW4I Series, ITC215-PW5H Series
  Access ANPR kit: IPMECS-2201D Series, IPMECS-2001B Series
  Parking space detection camera: ITCXX4-PH Series

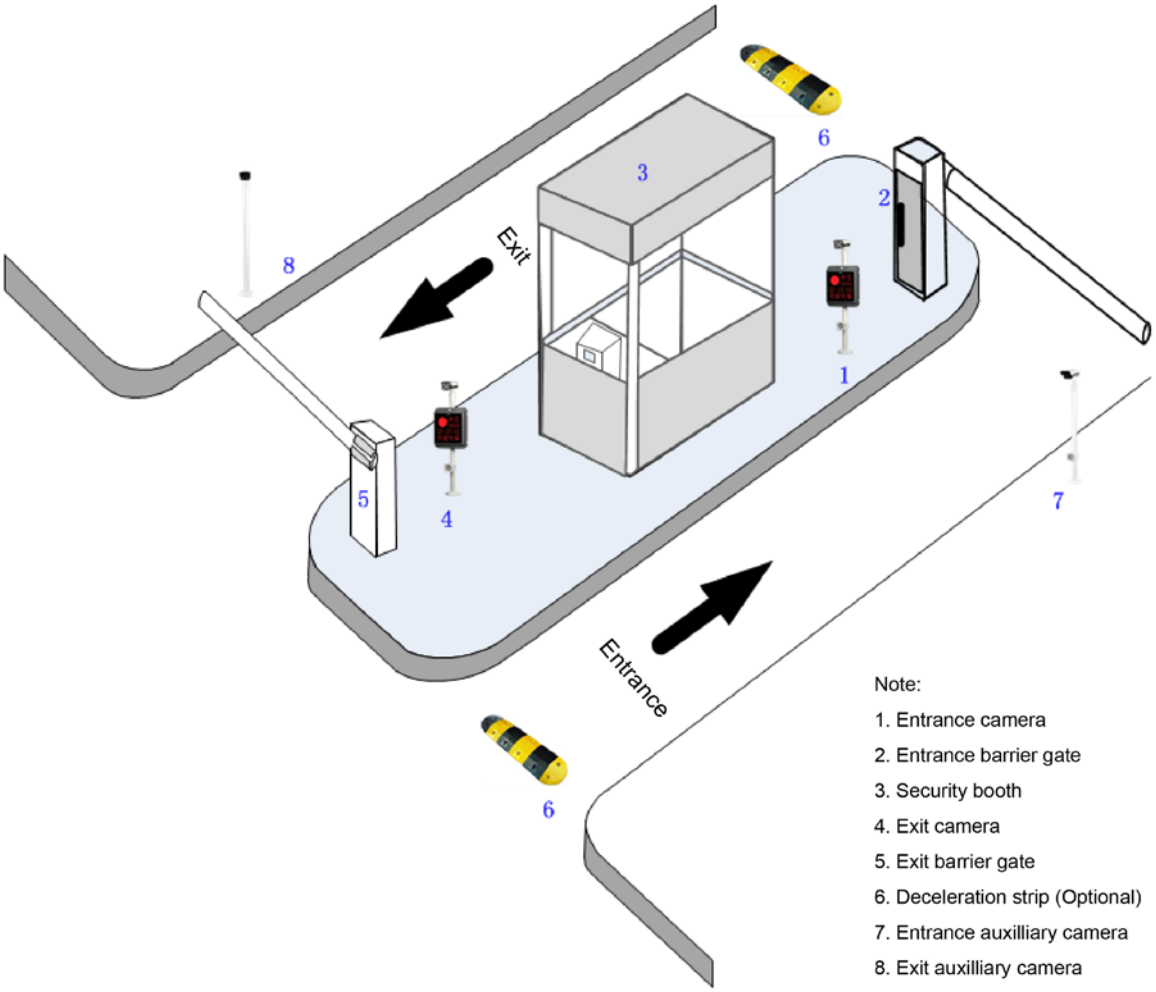
1.3 Application

- ITC and ITSET are installed at the traffic junction, to capture the traffic violations and count the vehicle flow.

Figure 1-1 Application (1)

- ITC, ITSE and IPMECK are installed at the access of parking lot, to control the entrance and exit of the vehicles and monitor the availability of parking space.

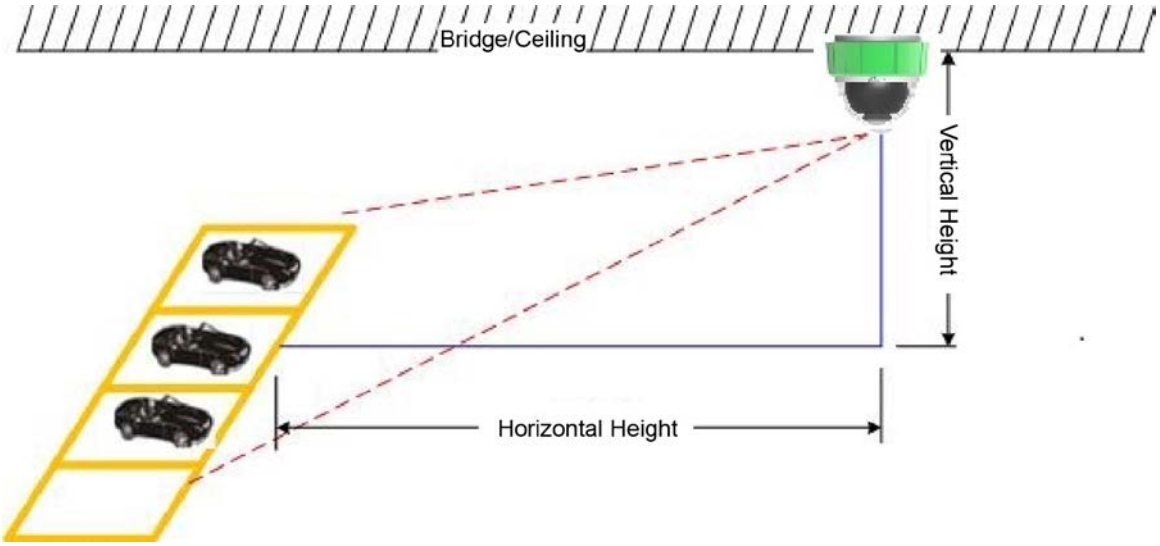
Figure 1-2 Application (2)



The access cameras with the functions of ITC and IPMECK are used to take snapshot and control barrier gates.

- ITC, ITSE and IPMECK are installed in parking lot, to capture and monitor the vehicles, and display the current status of parking spaces.

Figure 1-3 Application (3)



.

2 Function Modules

2.1 General

2.1.1 SDK Initialization

2.1.1.1 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.

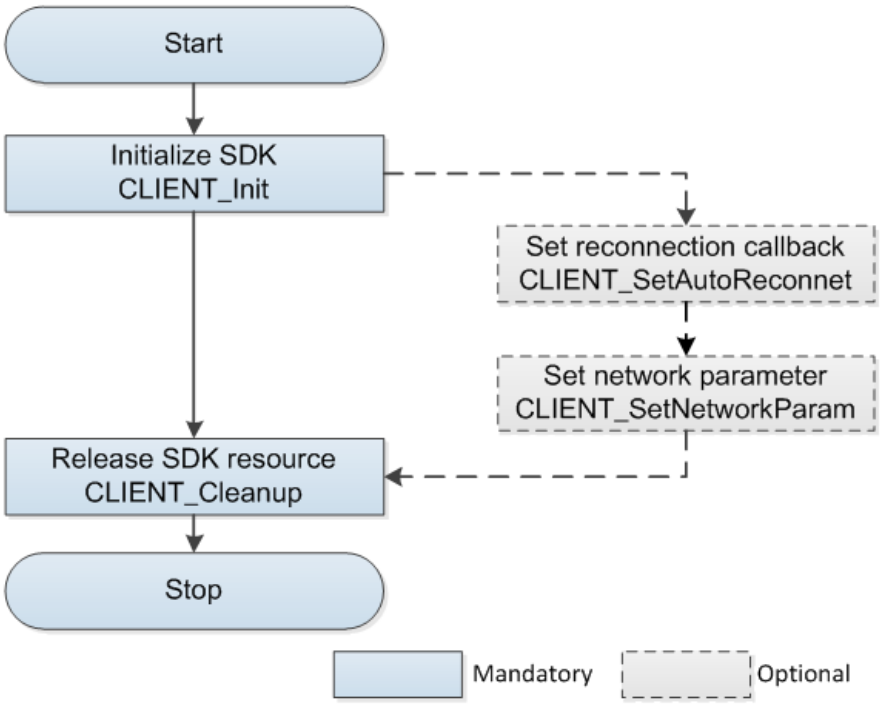
2.1.1.2 Interface Overview

Table 2-1 SDK initialization interfaces

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

2.1.1.3 Process

Figure 2-1 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

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

2.1.1.4 Example Code

```
// Set this callback through CLIENT_Init. When the device is disconnected, SDK informs the user through the 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();
```

2.1.2 Device Initialization

2.1.2.1 Introduction

The device is uninitialized by default. Initialize the device before using it.

- You can not log in to the uninitialized device.
- A password will be set for the default admin account during initialization.
- You can reset the password if you forgot it.

2.1.2.2 Interface Overview

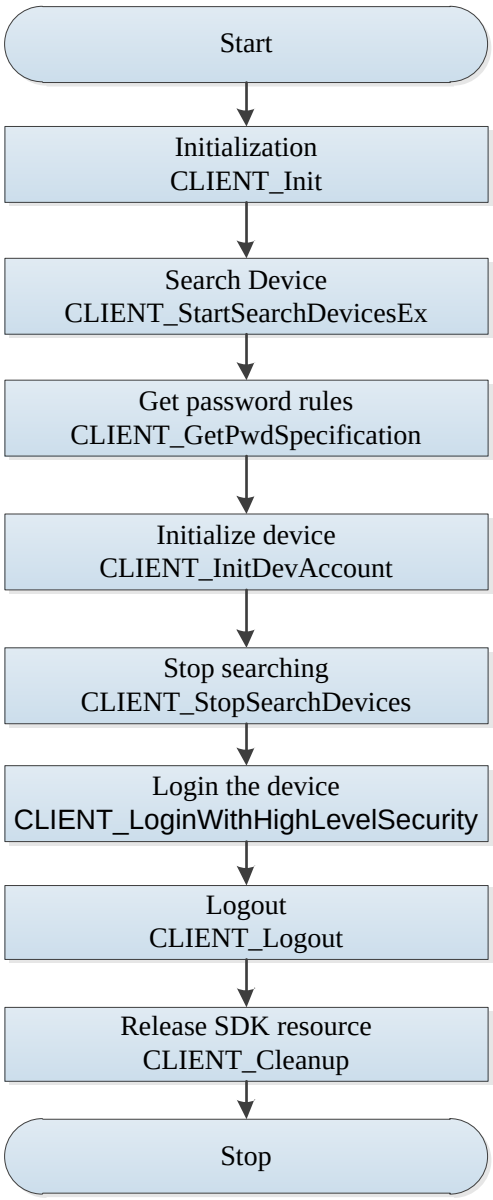
Table 2-2 Device initialization interfaces

Interface	Description
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.

2.1.2.3 Process

2.1.2.3.1 Device Initialization

Figure 2-2 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_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.

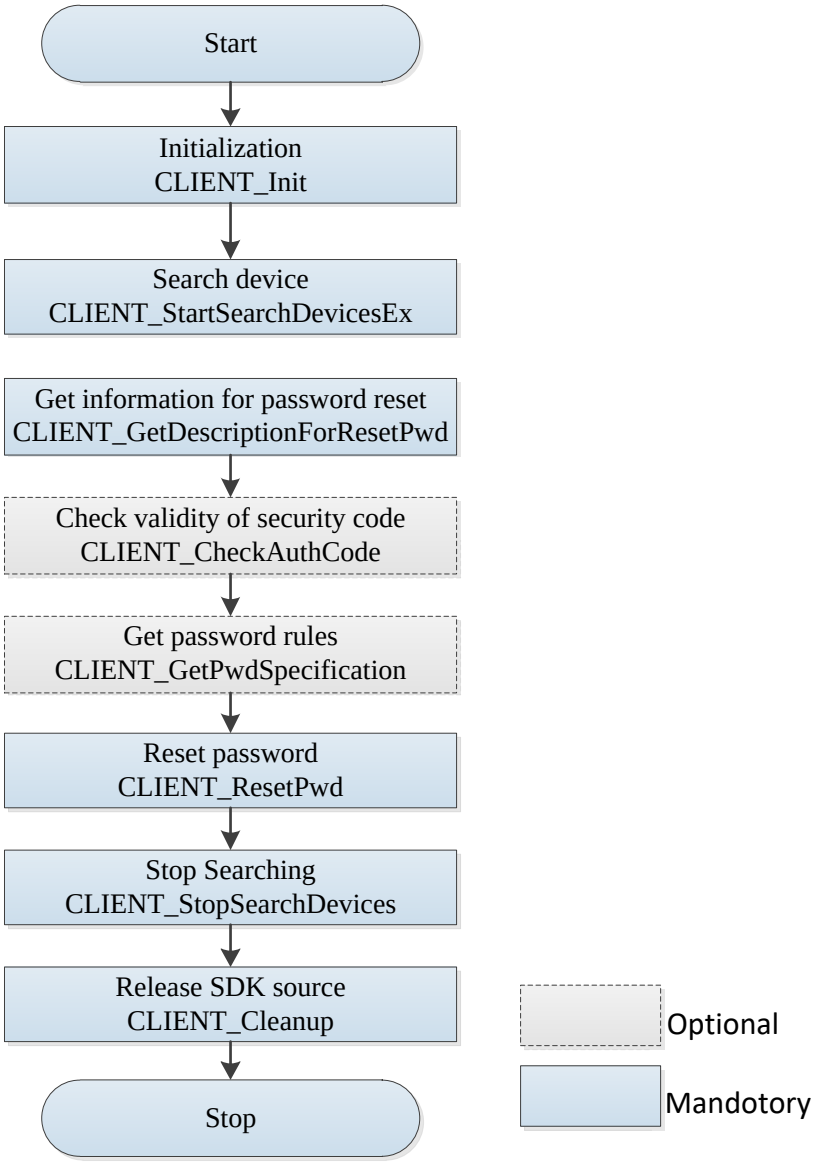
Step 8 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

Notes for Process

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

2.1.2.3.2 Password Resetting

Figure 2-3 Password resetting



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_GetDescriptionForResetPwd to get the information for password reset.

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

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

Step 6 Call CLIENT_ResetPwd to reset the password.

Step 7 Call CLIENT_StopSearchDevices to stop searching.

Step 8 Call CLIENT_LoginWithHighLevelSecurity and log in to the admin account with the configured password.

Step 9 After using the function module, call CLIENT_Logout to log out of 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.

2.1.2.4 Example Code

2.1.2.4.1 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);
```

2.1.2.4.2 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_StartSearchDevicesEx, and CLIENT_SearchDevicesByIPs)
NET_OUT_DESCRIPTION_FOR_RESET_PWD stOut = {sizeof(stOut)};
char szTemp[360];
stOut.pQrCode = szTemp;
CLIENT_GetDescriptionForResetPwd(&stIn, &stOut, 3000, NULL);//In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter. After successful connection, stout will output a QR code with address of stOut.pQrCode. Scan this QR code to get the security code for password reset. This security code will be sent to the reserved mobile phone or email box.
//(Optional) Check the security code
NET_IN_CHECK_AUTHCODE stIn1 = {sizeof(stIn1)};
strncpy(stIn1.szMac, szMac, sizeof(stIn1.szMac) - 1); //Set mac value
strncpy(stIn1.szSecurity, szSecu, sizeof(stIn1.szSecurity) - 1); // szSecu is the security code sent to the reserved mobile phone or email box
NET_OUT_CHECK_AUTHCODE stOut1 = {sizeof(stOut1)};
bRet = CLIENT_CheckAuthCode(&stIn1, &stOut1, 3000, NULL); //In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter
//Get password rules
NET_IN_PWD_SPECI stIn2 = {sizeof(stIn2)};
strncpy(stIn2.szMac, szMac, sizeof(stIn2.szMac) - 1); //Set mac value
NET_OUT_PWD_SPECI stOut2 = {sizeof(stOut2)};
CLIENT_GetPwdSpecification(&stIn2, &stOut2, 3000, NULL);// In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter. Set the password according to the rules which are used for preventing user from setting the passwords that are not supported by the device
//Reset password
NET_IN_RESET_PWD stIn3 = {sizeof(stIn3)};
strncpy(stIn3.szMac, szMac, sizeof(stIn3.szMac) - 1); //Set mac value
strncpy(stIn3.szUserName, szUserName, sizeof(stIn3.szUserName) - 1); //Set user name
strncpy(stIn3.szPwd, szPassWd, sizeof(stIn3.szPwd) - 1); //szPassWd is the password reset according to the rules
strncpy(stIn3.szSecurity, szSecu, sizeof(stIn1.szSecurity) - 1); //szSecu is the security code sent to the reserved mobile phone or email box
stIn3.byInitStaus = bStstus; //bStstus is the value of return field byInitStatus of device search interface (Callback of CLIENT_SearchDevices and CLIENT_StartSearchDevice, and CLIENT_SearchDevicesByIPs)
```

```
stIn3.byPwdResetWay = bPwdResetWay; // bPwdResetWay is the value of return field byPwdResetWay of device search interface (Callback of CLIENT_SearchDevices and CLIENT_StartSearchDevice, and CLIENT_SearchDevicesByIPs)
NET_OUT_RESET_PWD stOut3 = {sizeof(stOut3)};
CLIENT_ResetPwd(&stIn3, &stOut3, 3000, NULL);//In the case of single network card, the last parameter can be left unfilled; in the case of multiple network card, enter the host PC IP for the last parameter.
```

2.1.3 Device Login

2.1.3.1 Introduction

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

You will 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.

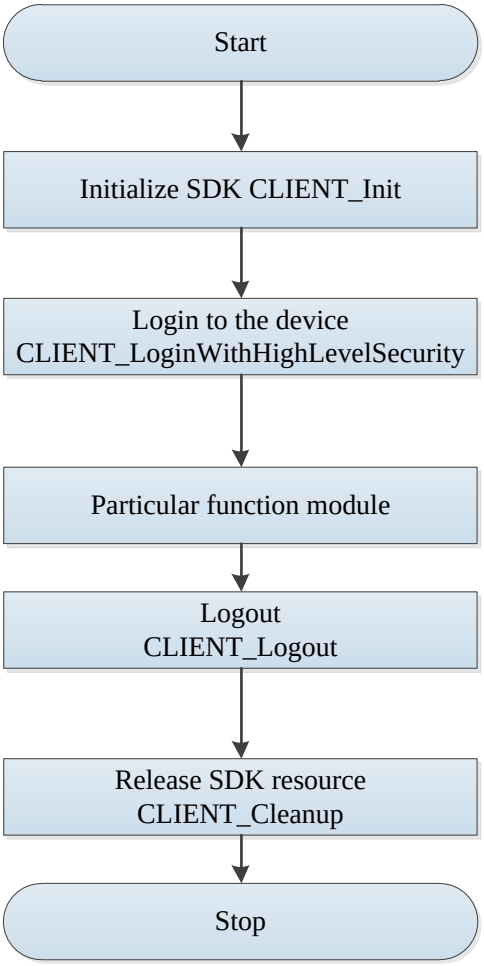
2.1.3.2 Interface Overview

Table 2-3 Device login interfaces

Interface	Description
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.

2.1.3.3 Process

Figure 2-4 Device login



Process Description

Step 1 Call CLIENT_Init to initialize SDK.
Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
Step 3 After successful login, you can realize the required function module.
Step 4 After using the function module, call CLIENT_Logout to log out of 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	Description
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.

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

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

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

2.1.3.4 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);
```

2.1.4 Real-time Monitoring

2.1.4.1 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 for you to perform independent treatment.
- Supports saving the real-time record to the specific file though saving the callback stream or calling the SDK interface.

2.1.4.2 Interface Overview

Table 2-5 Real-time monitoring interfaces

Interface	Description
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.

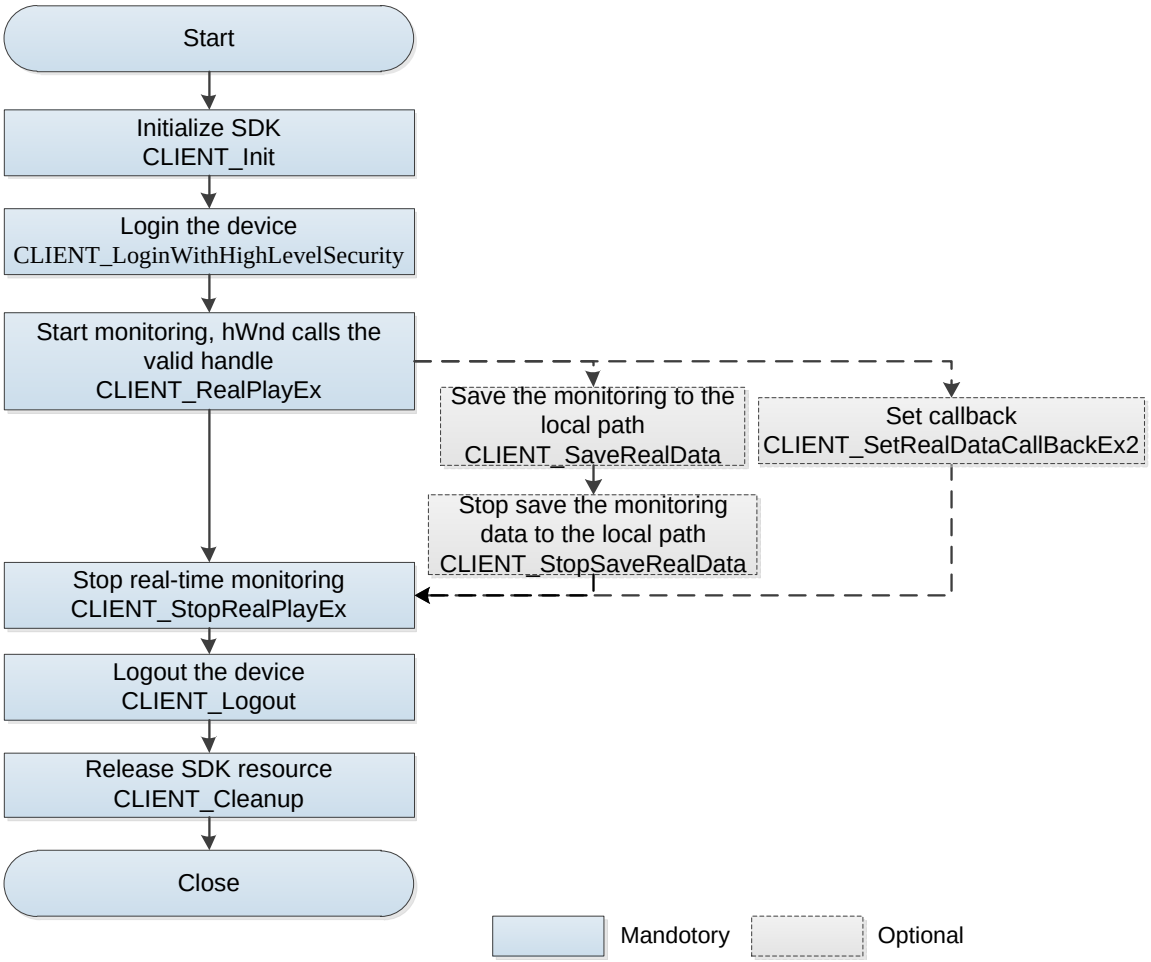
2.1.4.3 Process

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

2.1.4.3.1 SDK Decoding Play

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

Figure 2-5 Playing by SDK decoding library



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

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

Step 4 (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 log out of 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**.

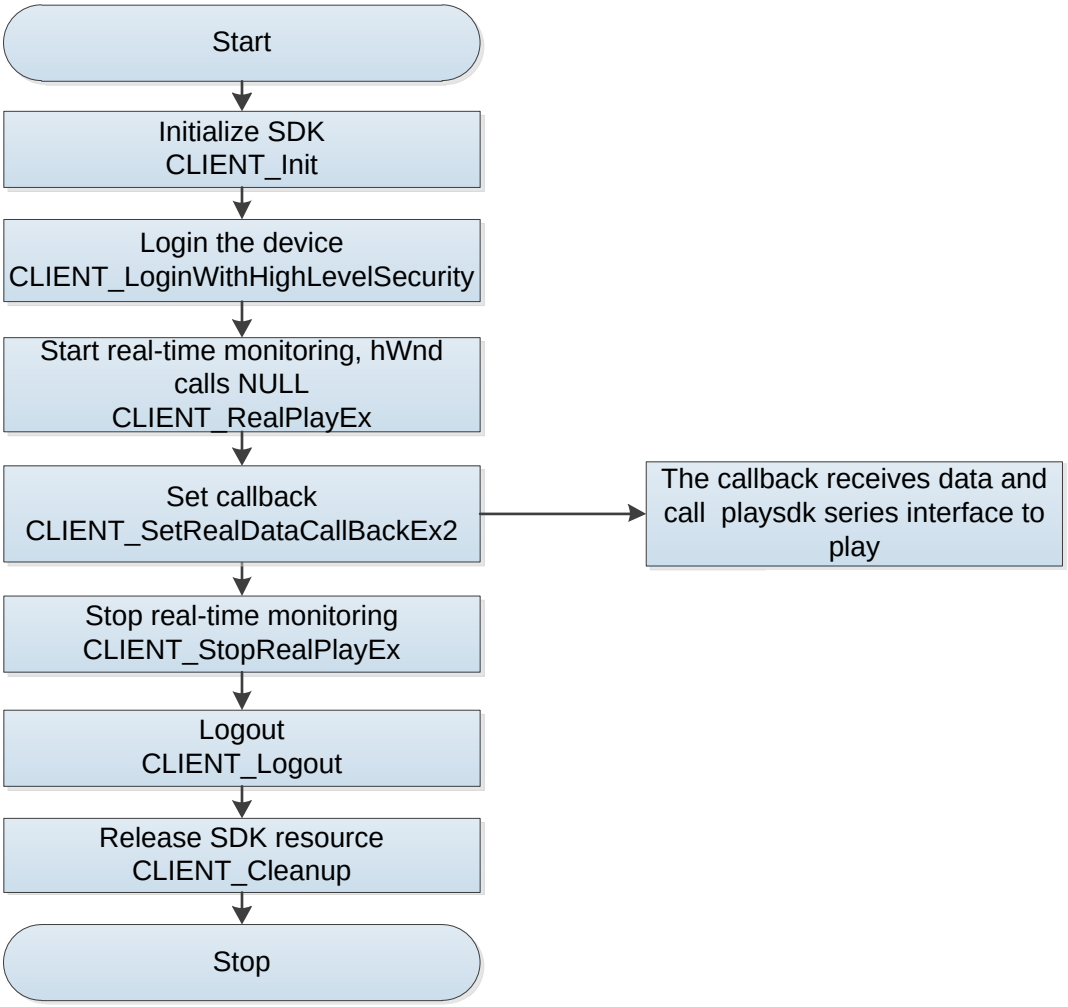
```
NET_PARAM stuNetParam = {0};
stuNetParam. nGetConnInfoTime = 5000; // unit ms
CLIENT_SetNetworkParam (&stuNetParam);
```

- CLIENT_SetNetworkParam (&stuNetParam);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 have 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 "2.1.4.3.2 Call Third-party Library."

2.1.4.3.2 Call Third-party Library

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

Figure 2-6 Calling the third-party library



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

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

Step 4 Call CLIENT_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 log out of 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.

2.1.4.4 Example Code

2.1.4.4.1 SDK Decoding Play

```
// 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);
}
```

2.1.4.4.2 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);
}
```

2.2 Traffic Junction

2.2.1 Download of Media File

2.2.1.1 Introduction

You can get the decoded pictures from ITSE through SDK and saves into the local path for further use.

To download the media files, the SDK connects to the device firstly. It sends query command per the query condition of media file and sends the download command after getting the query result to the device, and then the device will send the media files and decoded data to you.

2.2.1.2 Interface Overview

Table 2-6 Downloading media file interfaces

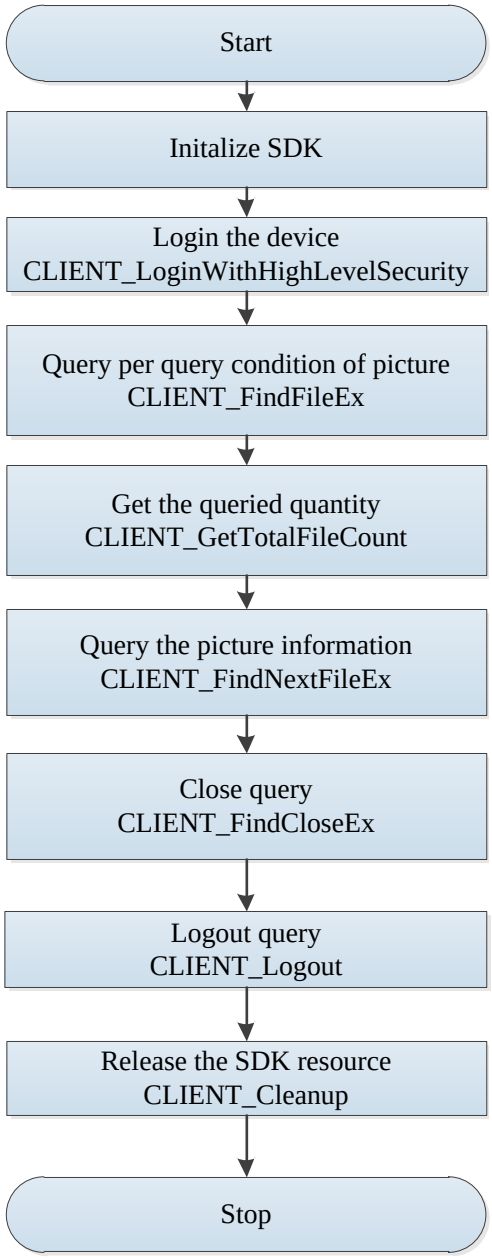
Interface	Implication
CLIENT_FindFileEx	Query per the query condition of file.
CLIENT_GetTotalFileCount	Get the queried quantity.
CLIENT_FindNextFileEx	Query the information of media file.
CLIENT_FindCloseEx	Close the query.
CLIENT_DownloadMediaFile	Download the media file.
CLIENT_StopDownloadMediaFile	Stop the download.

2.2.1.3 Process

The process of this function module is consisted of querying and downloading the media file.

2.2.1.3.1 Query of Media File

Figure 2-7 Querying the media file



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_FindFileEx to query per the query condition of media file.

Step 4 Call CLIENT_GetTotalFileCount to get the queried total number.

Step 5 Call CLIENT_FindNextFileExCall to review information of all the files.

Step 6 Call CLIENT_FindCloseEx to close query.

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

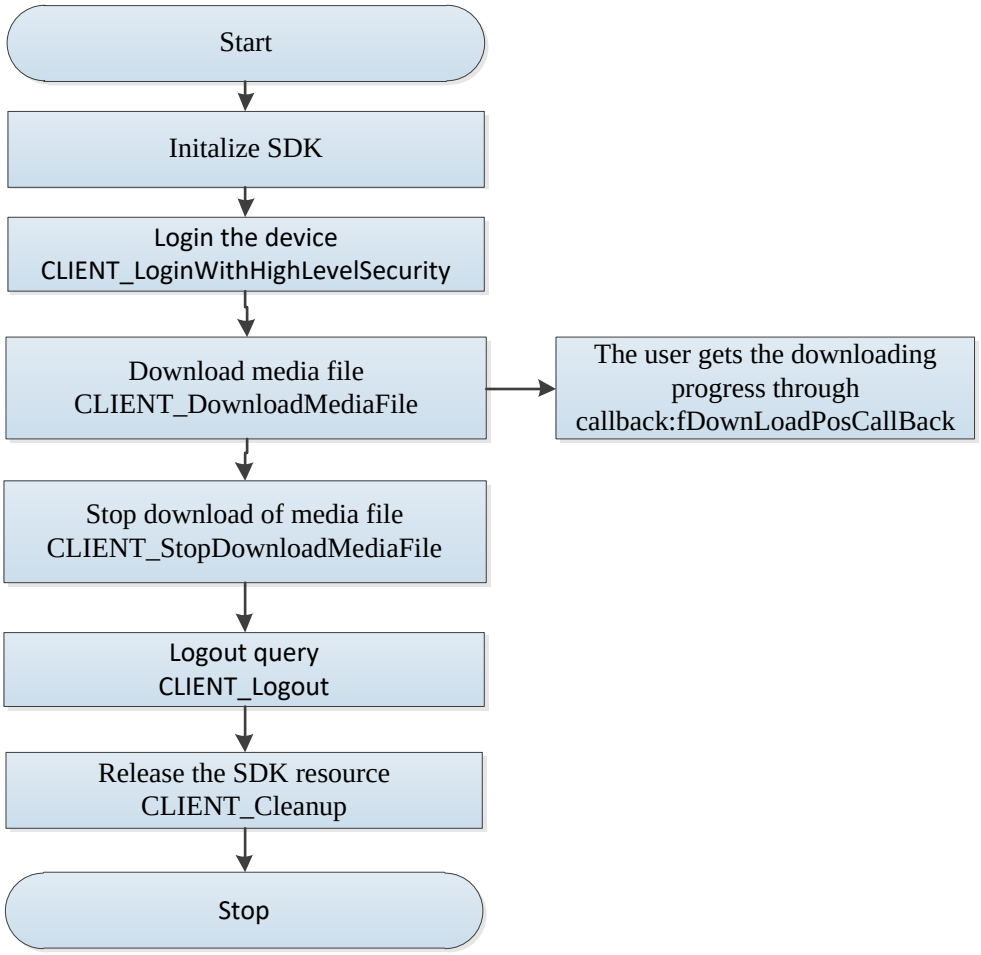
Step 8 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

Notes for Process

- Applicable device
  This process applies to ITSE devices. Please be noted that ITC only captures and identifies pictures and it is not capable of storing the data
- Parameters
  Use DH_FILE_QUERY_TRAFFICCAR_EX for parameter emType in CLIENT_FindFileEx, and the corresponding structure is MEDIA_QUERY_TRAFFICCAR_PARAM_EX. Use the corresponding structure MEDIAFILE_TRAFFICCAR_INFO_EX for interface CLIENT_FindNextFileEx.

2.2.1.3.2 Download of Media File

Figure 2-8 Downloading the media file



Process Description

Step 1 Call CLIENT_Init to initialize SDK.
Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
Step 3 Call CLIENT_DownloadMediaFile to download the media file.
Step 4 Call CLIENT_StopDownloadMediaFile to close the download.
Step 5 After using the function module, call CLIENT_Logout to log out of the device.
Step 6 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

Notes for Process

- Applicable device
  ITSE device. Please be noted that ITC only captures and identifies pictures and it is not capable of storing the data.
- Parameters
  Use only DH_FILE_QUERY_TRAFFICCAR for parameter emType in CLIENT_DownloadMediaFile, and the DH_FILE_QUERY_TRAFFICCAR_EX is not supported. The parameter lpMediaFileInfo is obtained through querying the media file.

2.2.1.4 Example Code

2.2.1.4.1 Query of Media File

```
int main()
{
        ............
        //Query condition of media file
MEDIA_QUERY_TRAFFICCAR_PARAM_EX stuCondition = {0};
stuCondition.dwSize = sizeof(MEDIA_QUERY_TRAFFICCAR_PARAM_EX);
stuCondition.stuParam.nMediaType = 1;
        ............
        //Query the media file
LLONG lFindHandle = CLIENT_FindFileEx(lLoginHandle, DH_FILE_QUERY_TRAFFICCAR_EX,
(void*)&stuCondition, NULL);
if(NULL == lFindHandle)
{
                printf("CLIENT_FindFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
                return -1;
}
int nCount = 0;
//Gets the quantity of queried media files
BOOL bRet = CLIENT_GetTotalFileCount(lFindHandle,&nCount,NULL);
if(FLASE == bRet)
{
                printf("CLIENT_GetTotalFileCount: failed! Error code: %x.\n", CLIENT_GetLastError());
                return -2;
}
//Review one queried media file per one time
int nMaxConut = 1;
do
```

```
{
MEDIAFILE_TRAFFICCAR_INFO_EX mediaFileInfo = {0};
mediaFileInfo.dwSize = sizeof(MEDIAFILE_TRAFFICCAR_INFO_EX);
//Query a single media file
bRet = CLIENT_FindNextFileEx(lFindHandle, nMaxConut, (void*)&mediaFileInfo,
sizeof(MEDIAFILE_TRAFFICCAR_INFO_EX), NULL);
if(FALSE == bRet)
{
                printf("CLIENT_FindNextFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
}
While ((nCount -= nMaxConut) > 0);
//Close query
bRet = CLIENT_FindCloseEx(lFindHandle);
if(FALSE == bRet)
{
        printf("CLIENT_FindCloseEx: failed! Error code: %x.\n", CLIENT_GetLastError());
        return -3;
}
}
```

2.2.1.4.2 Download of Media File

```
int main()
{
        ............
//Query the obtained media file
        MEDIAFILE_TRAFFICCAR_INFO info = mediaFileInfo.stuInfo;
        //Download the media file
LLONG lDownloadHandle = CLIENT_DownloadMediaFile(lLoginHandle, DH_FILE_QUERY_TRAFFICCAR,
(void*)&info, szFileName, DownLoadPosCallBack, NULL, NULL);
if(NULL == lDownloadHandle)
{
printf("CLIENT_DownloadMediaFile: failed! Error code: %x.\n", CLIENT_GetLastError());
}
Sleep(5000);
//Close download
BOOL bRet = CLIENT_StopDownloadMediaFile(lDownloadHandle);
if(FALSE == bRet)
{
        printf("CLIENT_StopDownloadMediaFile: failed! Error code: %x.\n", CLIENT_GetLastError());
```

```
}
}
//Download progress callback
void CALLBACK DownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize,
LDWORD dwUser)
{
        if (dwDownLoadSize == -1) //Download finished
        {
            printf("lPlayHandle: %p Download end!\n", lPlayHandle);
}
}
```

2.2.2 Manual Capture

2.2.2.1 Introduction

You can send the command through SDK to ITC or ITSE to capture pictures. The device will automatically analyze the pictures and report to you.

This function mainly applies to analyze the vehicles, detect if the vehicles have broken any regulations, and save the vehicles information.

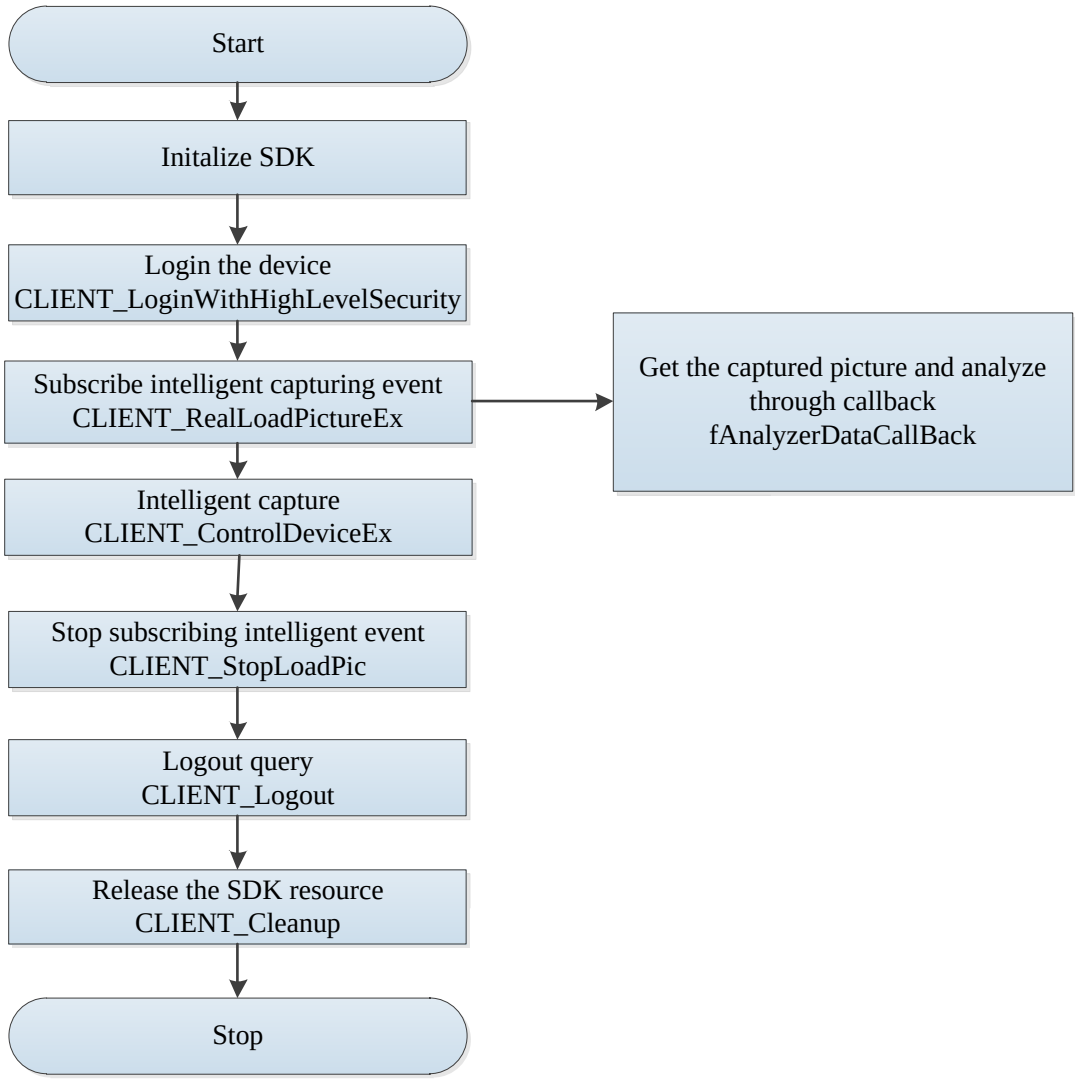
2.2.2.2 Interface Overview

Table 2-7 Manual capture interfaces

Interface	Implication
CLIENT_RealLoadPictureEx	Subscribe intelligent traffic event.
CLIENT_ControlDeviceEx	Manual capture.
CLIENT_StopLoadPic	Stop subscribing intelligent traffic event.

2.2.2.3 Process

Figure 2-9 Manual capture



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_RealLoadPictureEx to start subscribing intelligent traffic event.

Step 4 Call CLIENT_ControlDeviceEx to trigger intelligent capturing. Set parameter emType as DH_MANUAL_SNAP.

Step 5 Inform you of manual capturing event through the callback fAnalyzerDataCallBack.

Step 6 Call CLIENT_StopLoadPic to stop subscribing intelligent event.

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

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

Notes for Process

Setting of cache for receiving pictures:

Because SDK default cache is 2M, when the data is over 2M, call **CLIENT_SetNetworkParam** to set the receiving cache; otherwise the data pack will be lost.

2.2.2.4 Example Code

```
int main()
{
        ............
        //Subscribe intelligent capturing event
        LLONG lAnalyerHandle = CLIENT_RealLoadPictureEx(lLoginHandle, 0, (DWORD)EVENT_IVS_ALL, TRUE,
AnalyzerDataCallBack, NULL, NULL);
        if(NULL == lAnalyerHandle)
        {
                printf("CLIENT_RealLoadPictureEx: failed! Error code %x.\n", CLIENT_GetLastError());
                return -1;
        }
        MANUAL_SNAP_PARAMETER stuManualSnap = {0};
        stuManualSnap.nChannel = 0;
        sprintf((char*)stuManualSnap.bySequence,"abc");
        //Intelligent capturing
        BOOL bRet = CLIENT_ControlDeviceEx(lLoginHandle,DH_MANUAL_SNAP,&stuManualSnap);
        if(FALSE == bRet)
        {
                printf("CLIENT_ControlDeviceEx: failed! Error code %x.\n", CLIENT_GetLastError());
                return -2;
        }
Sleep(5000);
        //Stop subscribing intelligent capturing event
        BOOL bRet = CLIENT_StopLoadPic(lAnalyerHandle);
        if(FALSE == bRet)
        {
                printf("CLIENT_StopLoadPic: failed! Error code %x.\n", CLIENT_GetLastError());
                return -3;
        }
        return 0;
}
//Callback of intelligent capturing
int CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE
*pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
        switch(dwAlarmType)
        {
                case EVENT_IVS_TRAFFIC_MANUALSNAP:
```

```
{
                    DEV_EVENT_TRAFFIC_MANUALSNAP_INFO*                    pInfo                    =
                    (DEV_EVENT_TRAFFIC_MANUALSNAP_INFO*)pAlarmInfo;
                    printf("ManualSnapNo: %s", (char*)pInfo-> szManualSnapNo);
                        ............
                  break;
              }
          default:
              break;
      }
      return 0;
}
```

2.2.3 Upload of Intelligent Traffic Event

2.2.3.1 Introduction

The device decodes the real-time stream and sends the detected intelligent traffic event to you. The event includes the situations such as traffic violation, availability of parking space.

To upload the event, SDK connects to the device and subscribe the intelligent event. The device will send the event to SDK once such event has been detected.

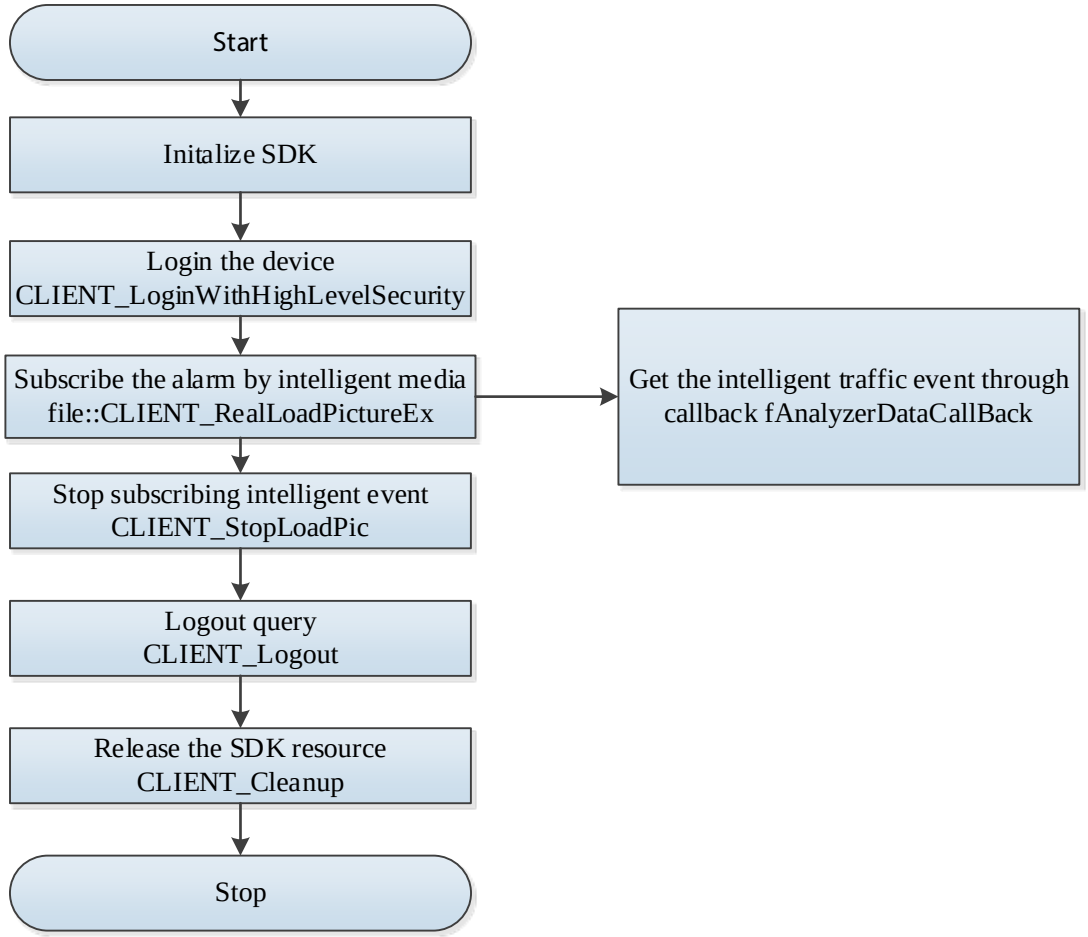
2.2.3.2 Interface Overview

Table 2-8 Intelligent traffic event uploading interfaces

Interface	Description
CLIENT_RealLoadPictureEx	Subscribe intelligent traffic event.
CLIENT_StopLoadPic	Stop subscribing intelligent traffic event.

2.2.3.3 Process

Figure 2-10 Uploading intelligent traffic event



Process Description

Step 1 Call CLIENT_Init to initialize SDK.
Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
Step 3 Call CLIENT_ RealLoadPictureEx to subscribe the intelligent traffic event.
Step 4 Get the uploaded event through callback fAnalyzerDataCallBack and send to you.
Step 5 Call CLIENT_StopLoadPic to stop subscribing event.
Step 6 After using the function module, call CLIENT_Logout to log out of the device.
Step 7 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

Notes for Process

- Subscribed event type: Support subscribing all event type (EVENT_IVS_ALL) at the same time or subscribing a single event type.
- Setting of cache for receiving pictures: Because SDK default cache is 2M, when the data is over 2M, call CLIENT_SetNetworkParam to set the receiving cache, otherwise the data pack will be lost.
- Setting of whether to receive pictures: Because some devices have 3G or 4G network, when SDK is connecting to the device, if it does not need to receive picture, set the parameter bNeedPicFile as False in interface CLIENT_ RealLoadPictureEx to only receive the intelligent event without picture.

2.2.3.4 Example Code

```
int main()
{
    ……
    //Subscribe the upload of intelligent traffic event
    LLONG lAnalyerHandle = CLIENT_RealLoadPictureEx(lLoginHandle, 0, (DWORD)EVENT_IVS_ALL, TRUE, AnalyzerDataCallBack, NULL, NULL);
    if(NULL == lAnalyerHandle)
    {
        printf("CLIENT_RealLoadPictureEx: failed! Error code %x.\n", CLIENT_GetLastError());
        return -1;
    }
    Sleep(5000);
    //Stop subscribing the upload of intelligent traffic event
    BOOL bRet = CLIENT_StopLoadPic(lAnalyerHandle);
    if(FALSE == bRet)
    {
        printf("CLIENT_StopLoadPic: failed! Error code %x.\n", CLIENT_GetLastError());
        return -2;
    }
    return 0;
}
//Callback for upload of intelligent traffic event
int CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    switch(dwAlarmType)
    {
        ……
        case EVENT_IVS_TRAFFIC_RUNREDLIGHT: // Event of running the red light
            {
                DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO*  pInfo  =  (DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO*)pAlarmInfo;
                ……
              break;
            }
        ……
        default:
            break;
```

```
    }
    return 0;
}
```

2.2.4 Vehicle Flow Statistics

2.2.4.1 Introduction

ITC device counts on all the passing vehicles to analyze the traffic status and directly send the result to you or to ITSE that sends to you.

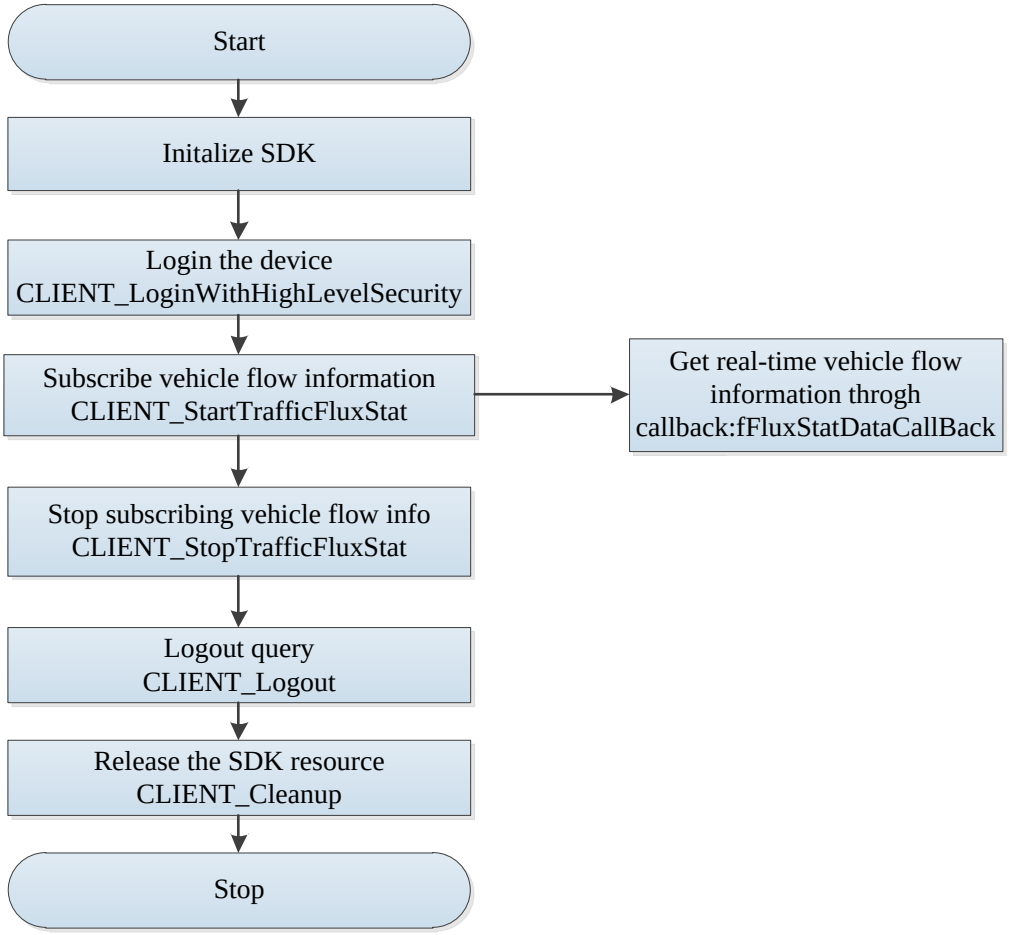
2.2.4.2 Interface Overview

Table 2-9 Vehicle flow statistics interfaces

Interface	Description
CLIENT_StartTrafficFluxStat	Subscribe intelligent traffic event
CLIENT_StopTrafficFluxStat	Stop subscribing intelligent traffic event

2.2.4.3 Process

Figure 2-11 Vehicle flow statistics



Process Description

Step 1 Call CLIENT_Init to initialize SDK.
Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
Step 3 Call CLIENT_StartTrafficFluxStat to subscribe the vehicle flow information.
Step 4 Get the vehicles information uploaded by ITC or ITSE through callback fFluxStatDataCallBack and inform you.
Step 5 Call CLIENT_StopTrafficFluxStat to stop subscribing the vehicle flow information.
Step 6 After using the function module, call CLIENT_Logout to log out of the device.
Step 7 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

Notes for Process

Callback data type: The parameter pEventInfo corresponds to structure of DEV_EVENT_TRAFFIC_FLOWSTAT_INFO.

2.2.4.4 Example Code

```
int main()
{
    ............
    NET_IN_TRAFFICFLUXSTAT stuIn = {0};
    stuIn.dwSize = sizeof(NET_IN_TRAFFICFLUXSTAT);
    stuIn.cbData = FluxStatDataCallBack;
    NET_OUT_TRAFFICFLUXSTAT stuOut = {0};
    stuOut.dwSize = sizeof(NET_OUT_TRAFFICFLUXSTAT);
    //Subscribe the vehicle flow statistics
    LLONG lFluxStatHandle = CLIENT_StartTrafficFluxStat(lLoginHandle, &stuIn, &stuOut);
    if(NULL == lFluxStatHandle)
    {
            printf("CLIENT_StartTrafficFluxStat: failed! Error code %x.\n", CLIENT_GetLastError());
        return -1;
    }
    Sleep(5000);
    //Stop subscribing the vehicle flow statistics
    BOOL bRet = CLIENT_StopTrafficFluxStat(lFluxStatHandle);
    if(FALSE == bRet)
    {
        printf("CLIENT_StopTrafficFluxStat: failed! Error code %x.\n", CLIENT_GetLastError());
        return -2;
    }
    return 0;
```

```
}
//Callback of vehicle flow statistics
int CALLBACK FluxStatDataCallBack (LLONG lFluxStatHandle, DWORD dwEventType, void* pEventInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
        DEV_EVENT_TRAFFIC_FLOWSTAT_INFO*  pInfo  =
    (DEV_EVENT_TRAFFIC_FLOWSTAT_INFO*)pEventInfo;
    ............
    return 0;
}
```

2.2.5 Searching for Historcial Traffic Flow Data

2.2.5.1 Introduction

To search for the historical data of traffic flow.

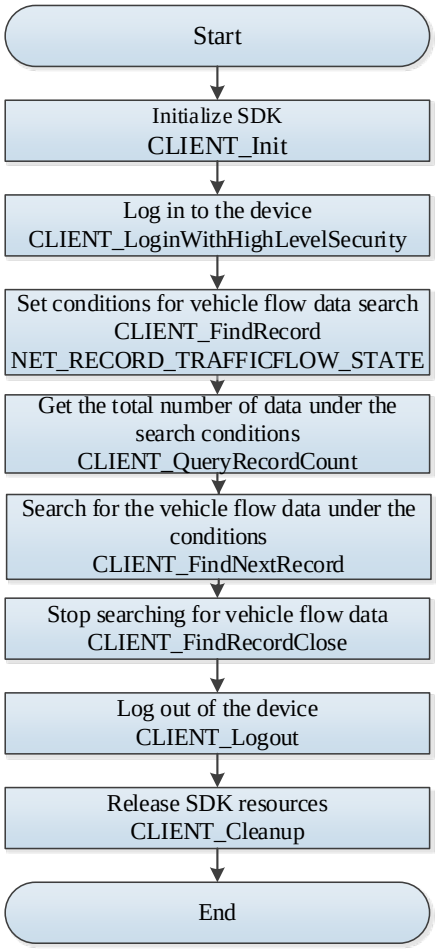
2.2.5.2 Interface Overview

Table 2-10 Searching for traffic flow historical data

Interface	Description
CLIENT_FindRecord	Set search conditions
CLIENT_QueryRecordCount	Get search number
CLIENT_FindNextRecord	Search data under the search conditions
CLIENT_FindRecordClose	Stop searching

2.2.5.3 Process

Figure 2-12 Searching for traffic flow historical data



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_OperateTrafficList, and select NET_TRAFFIC_LIST_INSERT for enumeration type to add blocklist and allowlist.

Step 4 Call CLIENT_FindRecord to set conditions for blocklist and allowlist search and select NET_RECORD_TRAFFICREDLIST (Allowlist) and NET_RECORD_TRAFFICBLACKLIST (Blocklist) for enumeration type

Step 5 Call CLIENT_QueryRecordCount to get the total number of data searched under the search conditions.

Step 6 Call CLIENT_FindNextRecord to search for certain number of blocklist and allowlist under the search conditions.

Step 7 Call CLIENT_OperateTrafficList, select NET_TRAFFIC_LIST_UPDATE (to modify) and NET_TRAFFIC_LIST_REMOVE (to delete) to respectively modify or delete the blocklist and allowlist acquired.

Step 8 Call CLIENT_FindRecordClose to clean search resources.

Step 9 Call CLIENT_Logout to log out of the device.

Step 10 Call CLIENT_Cleanup to release SDK resource.

Notes for Process

- To search for the traffic flow, there muat be traffic flow data in the process. The device shall be equipped with an SD card to save traffic flow data.
- In the traffic flow, when the average speed displays as－1, there is no vehicle passing through in the search period; When the average speed displays as or exceeds 0, it is the average speed of the vehicles passing through in the period.

2.2.5.4 Example Code

```
// Start searching and set search conditions
FIND_RECORD_TRAFFICFLOW_CONDITION stTrafficFlow =
{sizeof(FIND_RECORD_TRAFFICFLOW_CONDITION)};
stTrafficFlow.abChannelId =TRUE;
stTrafficFlow.nChannelId = 0;
stTrafficFlow.abLane = FALSE;
stTrafficFlow.bStartTime= TRUE;
stTrafficFlow.bEndTime= TRUE;
stTrafficFlow.stStartTime = startTime;
stTrafficFlow.stEndTime = endTime;
stTrafficFlow.bStatisticsTime = TRUE;
NET_IN_FIND_RECORD_PARAM stuFindInParam = {sizeof(NET_IN_FIND_RECORD_PARAM)};
stuFindInParam.emType = NET_RECORD_TRAFFICFLOW_STATE;
stuFindInParam.pQueryCondition = &stTrafficFlow;
NET_OUT_FIND_RECORD_PARAM stuFindOutParam = {sizeof(NET_OUT_FIND_RECORD_PARAM)};
bool bRet = CLIENT_FindRecord(m_lLoginHandle, &stuFindInParam, &stuFindOutParam,
MAX_TIMEOUT);
if (!bRet)
{
return;
}
// The total numbe of searches
NET_IN_QUEYT_RECORD_COUNT_PARAM inQueryCountParam =
{ sizeof(NET_IN_QUEYT_RECORD_COUNT_PARAM)};
inQueryCountParam.lFindeHandle = stuFindOutParam.lFindeHandle;
NET_OUT_QUEYT_RECORD_COUNT_PARAM outQueryCountParam =
{ sizeof(NET_OUT_QUEYT_RECORD_COUNT_PARAM) };
bRet = CLIENT_QueryRecordCount(&inQueryCountParam, &outQueryCountParam , MAX_TIMEOUT);
if (!bRet)
{
MessageBox(ConvertString("Query record count failed!"), ConvertString("Prompt"));
return;
}
// Search for 100 records
int nQueryCount = 100;
NET_RECORD_TRAFFIC_FLOW_STATE* pRecordList = new
```

```
NET_RECORD_TRAFFIC_FLOW_STATE[nQueryCount];
memset(pRecordList, 0, sizeof(NET_RECORD_TRAFFIC_FLOW_STATE) * nQueryCount);
for (int unIndex = 0; unIndex < nQueryCount; ++unIndex)
{
pRecordList[unIndex].dwSize = sizeof(NET_RECORD_TRAFFIC_FLOW_STATE);
}
NET_IN_FIND_NEXT_RECORD_PARAM stuFindNextInParam =
{sizeof(NET_IN_FIND_NEXT_RECORD_PARAM)};
stuFindNextInParam.lFindeHandle = stuFindOutParam.lFindeHandle;
stuFindNextInParam.nFileCount = nQueryCount;
NET_OUT_FIND_NEXT_RECORD_PARAM stuFindNextOutParam =
{sizeof(NET_OUT_FIND_NEXT_RECORD_PARAM)};
stuFindNextOutParam.pRecordList = pRecordList;
stuFindNextOutParam.nMaxRecordNum = nQueryCount;
bRet = CLIENT_FindNextRecord(&stuFindNextInParam, &stuFindNextOutParam, MAX_TIMEOUT);
if (!bRet)
{
MessageBox(ConvertString("Query record count failed!"), ConvertString("Prompt"));
}
// Stop searching
CLIENT_FindRecordClose(stuFindOutParam.lFindeHandle);
delete[] pRecordList;
```

2.2.6 Subscribing to Intelligent Event

2.2.6.1 Introduction

Intelligent event subscription: Cameras and storage devices analyze real-time streams based on the intelligent algorithm and report events and relevant information to users when detecting subscribed intelligent events.

Intelligent event referred in the manual includes: Intelligent video surveillance intelligent events (tripwire, intrusion, etc), face detection, face recognition, human detection and intelligent traffic intelligent events (ANPR, overspeed, underspeed, traffic jam and more).

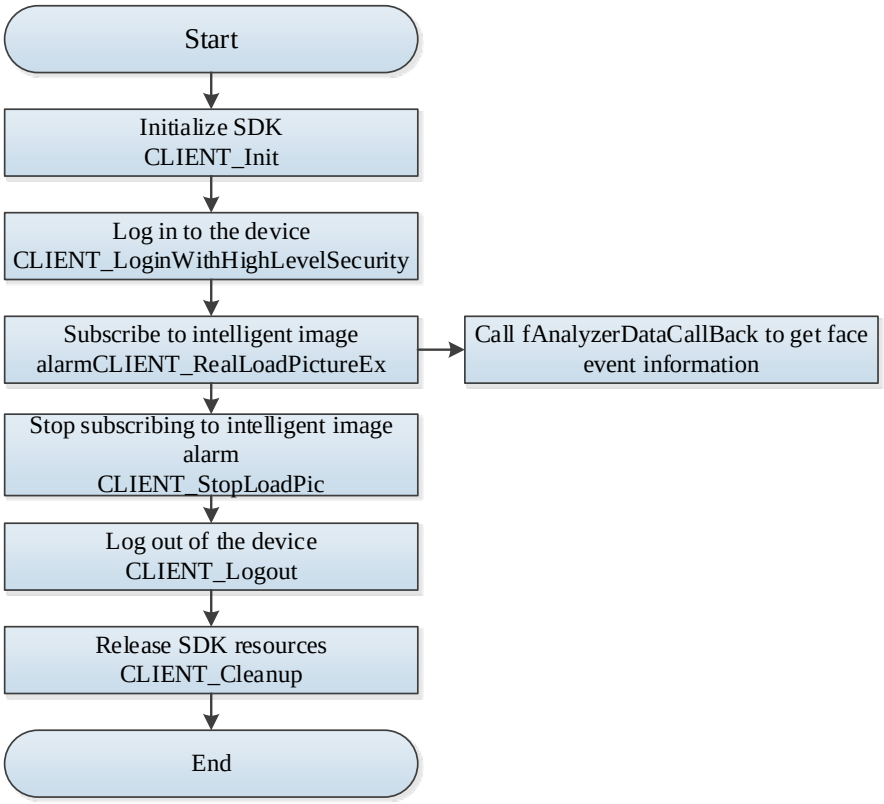
2.2.6.2 Interface Overview

Table 2-11 Subscribing to intelligent event

Interface	Description
CLIENT_RealLoadPictureEx	Subscribe to intelligent event
CLIENT_StopLoadPic	Stop subscribing to intelligent event

2.2.6.3 Process

Figure 2-13 Reporting face event



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_RealLoadPictureEx to subscribe to intelligent event from the device.

Step 4 After subscription, the device will report the intelligent event through fAnalyzerDataCallBack. With the function, users can easily filter the intelligent alarm events that they need based on alarm type.

Step 5 Call CLIENT_StopLoadPic to stop subscribing to intelligent event.

Step 6 Call CLIENT_Logout to log out of the device.

Step 7 Call CLIENT_Cleanup to release SDK resources

Notes for Process

- Subscription event type: Users can either subscribe to all intelligent events or a single intelligent event.
- Set whether to receive images: Some devices are set in 3G or 4G network environment. If images are not needed, users may choose to only receive face event information through configuring bNeedPicFile as False in CLIENT_RealLoadPictureEx interface when SDK is connected to the device.

2.2.6.4 Example Code

```
// Callback function of intelligent event report
int CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void*
pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
switch(dwAlarmType)
{
// Filter the intelligent event that you need
……
case EVENT_IVS_TRAFFIC_VEHICLE_RACE: // Car racing
……
default:
break;
}
}
// Subscribe to intelligent event report
LLONG lAnalyerHandle = CLIENT_RealLoadPictureEx(lLoginHandle, 0, (DWORD)EVENT_IVS_ALL, TRUE,
AnalyzerDataCallBack, NULL, NULL);
if(NULL == lAnalyerHandle)
{
printf("CLIENT_RealLoadPictureEx: failed! Error code %x.\n", CLIENT_GetLastError());
return -1;
}
// Stop subscribing to intelligent event report
CLIENT_StopLoadPic(lAnalyerHandle);
```

2.2.7 Video and Image Search/Playback/Download

2.2.7.1 Introduction

If intelligent algorithm of the device detects some subscribed intelligent event while analyzing the real-time streams, video recording and image capturing are triggered and corresponding data are saved. Users can search for, download and playback the videos and images of the intelligent event saved in the device.

2.2.7.2 Interface Overview

Table 2-12 Video and image search/playback/download interfaces

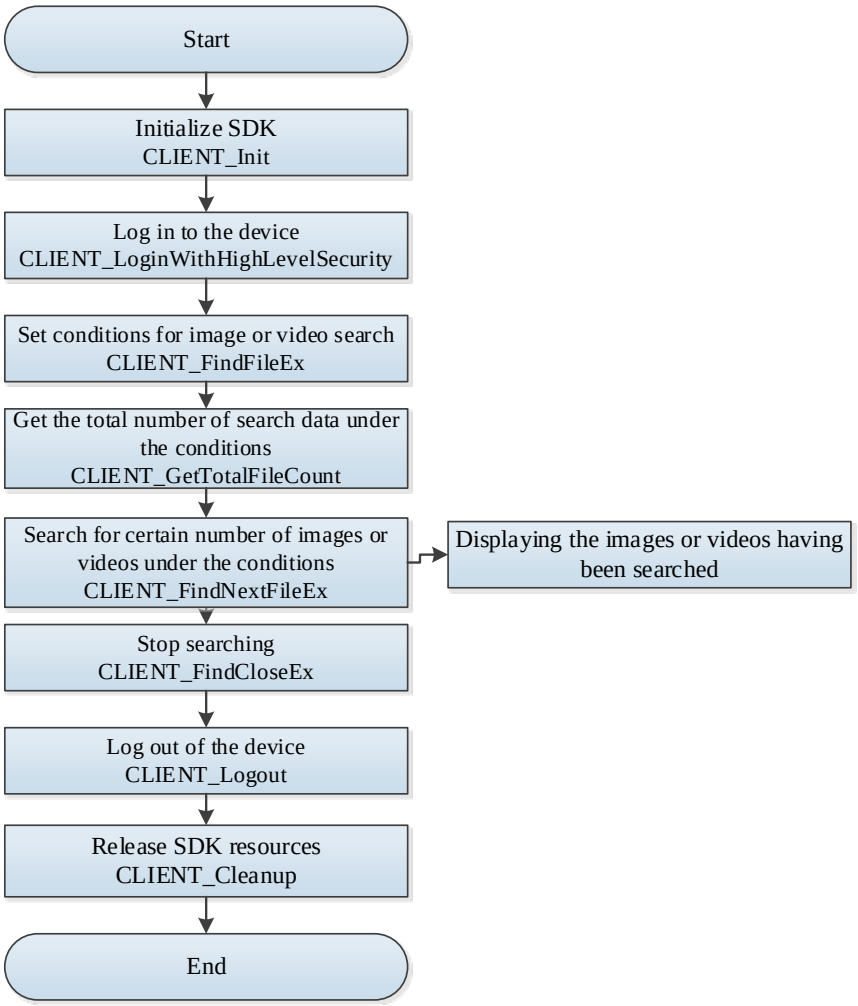
Interface	Description
CLIENT_FindFileEx	Set conditions for video or image search.
CLIENT_GetTotalFileCount	Get the total number of videos or images searched under the search conditions
CLIENT_FindNextFileEx	Search for certain number of videos or images

Interface	Description
CLIENT_FindCloseEx	Stop searching
CLIENT_PlayBackByTimeEx2	Playback the videos by time
CLIENT_StopPlayBack	Stop video playback
CLIENT_DownloadByTimeEx	Download videos
CLIENT_StopDownload	Stop downloading videos
CLIENT_DownloadRemoteFile	Download images

2.2.7.3 Process

2.2.7.3.1 Searching for Videos or Images

Figure 2-14 Searching for videos or images



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_FindFileEx to set search conditions, and return to the search handle after setting searching conditions. Judge the search type according to the emType.

Step 4 Call CLIENT_GetTotalFileCount to get the total number of videos or images having been searched under the search conditions.

Step 5 Call CLIENT_FindNextFileEx to search for certain number of videos or images and save them for playback or download afterward.

Step 6 Call CLIENT_FindCloseEx to stop searching.

Step 7 Call CLIENT_Logout to log out of the device.

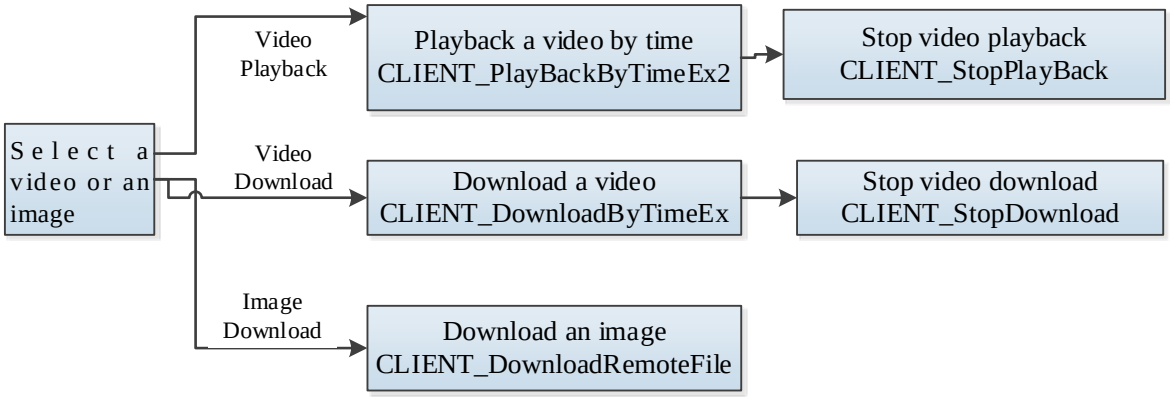
Step 8 Call CLIENT_Cleanup to release SDK resources.

Notes for Process

- PQueryCondition in CLIENT_FindFileEx is applied and released by the user and the type of it is determined by the emType.
- If the search succeeded, CLIENT_FindFileEx will return to the search handle and continue to search specific videos or images using the handle as parameter. To close the search handle, users must call CLIENT_FindCloseEx.
- Call CLIENT_FindNextFileEx to set search number. If the number exceeds 1, pMediaFileInfo must be a data pointer which exceeds or equals the search number.

2.2.7.3.2 Videos Playback and Download/Images download

Figure 2-15 Playback and download videos /Download images



Process Description

Download or playback a video or an image searched from CLIENT_FindNextFileEx.

- Video Playback
  Call CLIENT_PlayBackByTimeEx2 to playback a video according to the start time and end time of the video. Call CLIENT_StopPlayBack to stop video playback during or at the end of the playback.
- Video Download
  Call CLIENT_DownloadByTimeEx to download a video according to the start time and the end time of the video. Call CLIENT_StopDownload to stop downloading the video.
- Image Download
  Call CLIENT_DownloadRemoteFile to download an image using the file name and image type of the search results.

Notes for Process

Video playback and download as well as image download are based on the search results.

2.2.7.4 Example Code

2.2.7.4.1 Searching for Videos or Images

```
// Search conditions
MEDIAFILE_FACE_DETECTION_PARAM param;
memset(&param, 0, sizeof(param));
param.dwSize = sizeof(param);
param.stuDetail.dwSize = sizeof(MEDIAFILE_FACE_DETECTION_DETAIL_PARAM);
param.nChannelID = -1;
param.stuStartTime = startTime;
param.stuEndTime = endTime
param.emPicType = NET_FACEPIC_TYPE_SMALL;    // Small face images
param.bDetailEnable = FALSE;
param.emSex = EM_DEV_EVENT_FACEDETECT_SEX_TYPE_MAN;
param.bAgeEnable = FALSE;
param.nEmotionValidNum = 0;
param.emGlasses = EM_FACEDETECT_WITH_GLASSES;
// Search for small images of face detection
LLONG lFindFileHandle = CLIENT_FindFileEx(g_lLoginHandle, DH_FILE_QUERY_FACE_DETECTION,
&param, NULL,5000);
if (lFindFileHandle == 0)
{
printf("CLIENT_FindFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
return;
}
// The number of faces having been searched
BOOL nRet = CLIENT_GetTotalFileCount(lFindFileHandle,&nCount,NULL);
if (!nRet)
{
printf("CLIENT_GetTotalFileCount: failed! Error code: %x.\n", CLIENT_GetLastError());
return;
}
// The number of searches
int nMaxConut = 10;
MEDIAFILE_FACE_DETECTION_INFO*  pMediaFileInfo  =  NEW
MEDIAFILE_FACE_DETECTION_INFO[nMaxConut];
memset(pMediaFileInfo, 0, sizeof(MEDIAFILE_FACE_DETECTION_INFO) * nMaxConut);
for (int i = 0; i < nMaxConut; i++)
{
    pMediaFileInfo[i].dwSize = sizeof(MEDIAFILE_FACE_DETECTION_INFO);
}
// Start searching
int nRet = CLIENT_FindNextFileEx(lFindFileHandle, nMaxConut, (void*)pMediaFileInfo, nMaxConut *
sizeof(MEDIAFILE_FACE_DETECTION_INFO), NULL,3000);
if (nRet < 0)
{
```

```
printf("CLIENT_FindNextFileEx: failed! Error code: %x.\n", CLIENT_GetLastError());
return;
}
// Stop searching
CLIENT_FindCloseEx(lFindFileHandle);
```

2.2.7.4.2 Video Playback

```
// Set the stream type for video playback. Set to main stream.
int nStreamType = 0; // 0-Main and sub stream, 1-Main stream, 2-Sub stream
CLIENT_SetDeviceMode(lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);
// Set video file type for playback. Set to all video types.
NET_RECORD_TYPE emFileType = NET_RECORD_TYPE_ALL; // All video types
CLIENT_SetDeviceMode(lLoginHandle, DH_RECORD_TYPE, &emFileType);
// Startvideo playback
int nChannelID = 0; // Channel No.
NET_IN_PLAY_BACK_BY_TIME_INFO stIn = {0};
NET_OUT_PLAY_BACK_BY_TIME_INFO stOut = {0};
memcpy(&stIn.stStartTime, &stuStartTime, sizeof(stuStartTime));
memcpy(&stIn.stStopTime, &stuStopTime, sizeof(stuStopTime));
stIn.hWnd = hWnd;
stIn.fDownLoadDataCallBack = DataCallBack;
stIn.dwDataUser = NULL;
stIn.cbDownLoadPos = NULL;
stIn.dwPosUser = NULL;
stIn.nPlayDirection = emDirection;
stIn.nWaittime = 10000;
LLONG lPlayHandle = CLIENT_PlayBackByTimeEx2(lLoginHandle, nChannelID, &stIn, &stOut);
if (0 == lPlayHandle)
{
printf("CLIENT_PlayBackByTimeEx2: failed! Error code: %x.\n", CLIENT_GetLastError());
}
if (FALSE == CLIENT_StopPlayBack(lPlayHandle))
{
printf("CLIENT_StopPlayBack Failed, lRealHandle[%x]!Last Error[%x]\n" , lPlayHandle,
CLIENT_GetLastError());
}
Video Download
//Playbck process function
void CALLBACK TimeDownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser);
// Playback/Download data callback function
int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LDWORD dwUser);
int main()
{
// Set the stream type for video playback. Set to main stream.
int nStreamType = 0; // 0-Main and sub stream, 1- Main stream, 2- Sub stream
```

```
CLIENT_SetDeviceMode(lLoginHandle, DH_RECORD_STREAM_TYPE, &nStreamType);
// Set the start time and end time of download
int nChannelID = 0; // Channel No.
NET_TIME stuStartTime = {0};
stuStartTime.dwYear = 2018;
stuStartTime.dwMonth = 9;
stuStartTime.dwDay = 17;
NET_TIME stuStopTime = {0};
stuStopTime.dwYear = 2018;
stuStopTime.dwMonth = 9;
stuStopTime.dwDay = 18;
// Start video download
// One of the function variables of sSavedFileName and fDownLoadDataCallBack must be valid or the
parameter input is wrong.
lDownloadHandle = CLIENT_DownloadByTimeEx(lLoginHandle, nChannelID, EM_RECORD_TYPE_ALL,
&stuStartTime, &stuStopTime, "test.dav", TimeDownLoadPosCallBack, NULL, DataCallBack, NULL);
if (lDownloadHandle == 0)
{
printf("CLIENT_DownloadByTimeEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
// Stop downloading. Call either during or at the end of the download.
if (0 != lDownloadHandle)
{
if (!CLIENT_StopDownload(lDownloadHandle))
{
printf("CLIENT_StopDownload Failed, lDownloadHandle[%x]!Last Error[%x]\n" ,
lDownloadHandle, CLIENT_GetLastError());
}
}
}
void CALLBACK TimeDownLoadPosCallBack(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser)
{
// Users manage the process callback
}
int CALLBACK DataCallBack(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD
dwBufSize, LDWORD dwUser)
{
switch(dwDataType)
{
case 0:
//Original data
// Users save stream data and decode or forward the data after closing the call function.
break;
case 1://Standard video data
break;
case 2:    //yuv data
```

```
break;
case 3://pcm audio data
break;
default:
break;
}
return 0;
}
```

2.2.7.4.3 Image Download

```
DH_IN_DOWNLOAD_REMOTE_FILE stuRemoteFileParm;
memset(&stuRemoteFileParm, 0, sizeof(DH_IN_DOWNLOAD_REMOTE_FILE));
stuRemoteFileParm.dwSize = sizeof(DH_IN_DOWNLOAD_REMOTE_FILE);
stuRemoteFileParm.pszFileName = pInfo->stObjectPic.szFilePath ;
stuRemoteFileParm.pszFileDst = szFileName;
DH_OUT_DOWNLOAD_REMOTE_FILE *fileinfo = NEW DH_OUT_DOWNLOAD_REMOTE_FILE;
fileinfo->dwSize = sizeof(DH_OUT_DOWNLOAD_REMOTE_FILE);
if (!CLIENT_DownloadRemoteFile(g_lLoginHandle, &stuRemoteFileParm, fileinfo))
{
printf("CLIENT_DownloadRemoteFile Failed,Last Error[%x]\n" , CLIENT_GetLastError());
}
```

2.3 Parking Lot

2.3.1 Barrier Control

2.3.1.1 Introduction

IPMECK device can control the opening and closing operations of road barrier. You can send the command through SDK to IPMECK for the manual barrier control. For example:

- Issue the configuration to IPMECK through SDK, to set the barrier normal open or normal close, and set the period.
- Barrier will opened in case of vehicle location event (dominant) or traffic junction event to link opening barrier gate.

The barrier control mainly applies to the places such as parking lot, toll gate, and gate of district.

Applicable device: IPMECK device.

2.3.1.2 Interface Overview

Table 2-13 Barrier control interfaces

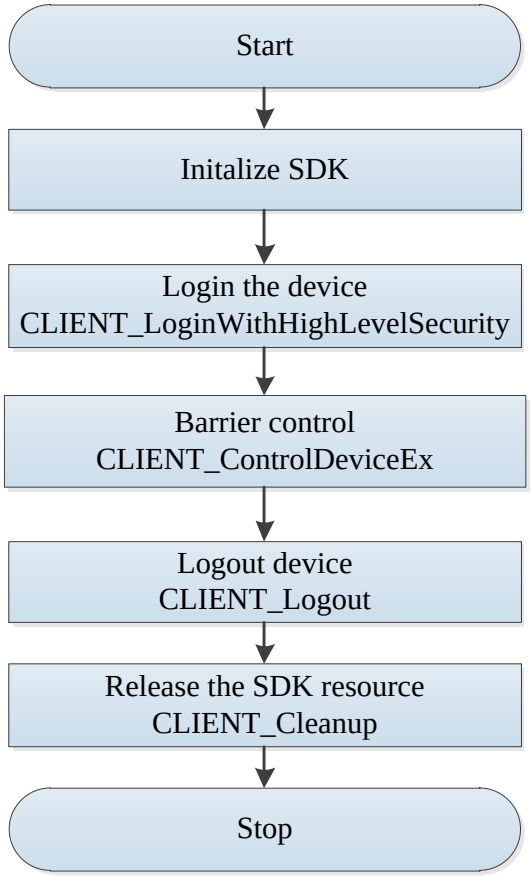
Interface	Description
CLIENT_ControlDeviceEx	Barrier control.

Interface	Description
CLIENT_GetConfig	Get the configuration of barrier gate.
CLIENT_SetConfig	Isuee the configuration of barrier gate.
CLIENT_SetDVRMessCallBack	Set alarm callback function.
CLIENT_StartListenEx	Subscribe vehicle location event.
CLIENT_StopListen	Stop subscribing vehicle location event.
CLIENT_RealLoadPictureEx	Subscribe traffic junction event.
CLIENT_StopLoadPic	Stop subscribing traffic event.

2.3.1.3 Process

2.3.1.3.1 Manual Barrier Control

Figure 2-16 Manual barrier control



Process Description

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

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

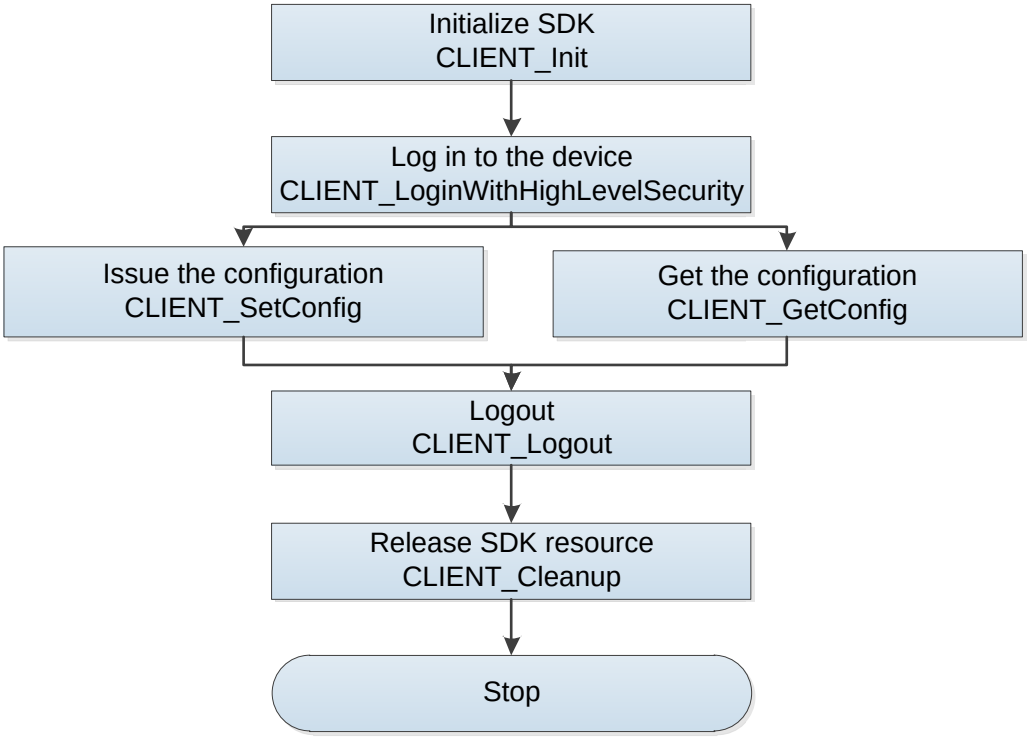
Step 3 Call **CLIENT_ControlDeviceEx** to open or close the barrier.

Step 4 After using the function module, call **CLIENT_Logout** to log out of the device.

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

2.3.1.3.2 Barrier Control Configuration

Figure 2-17 Barrier control configuration



Process Description

Setting

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_SetConfig to set the period of barrier normally open enable or barrier normally open mode.

Step 4 Call CLIENT_Logout to log out of the device.

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

Getting

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

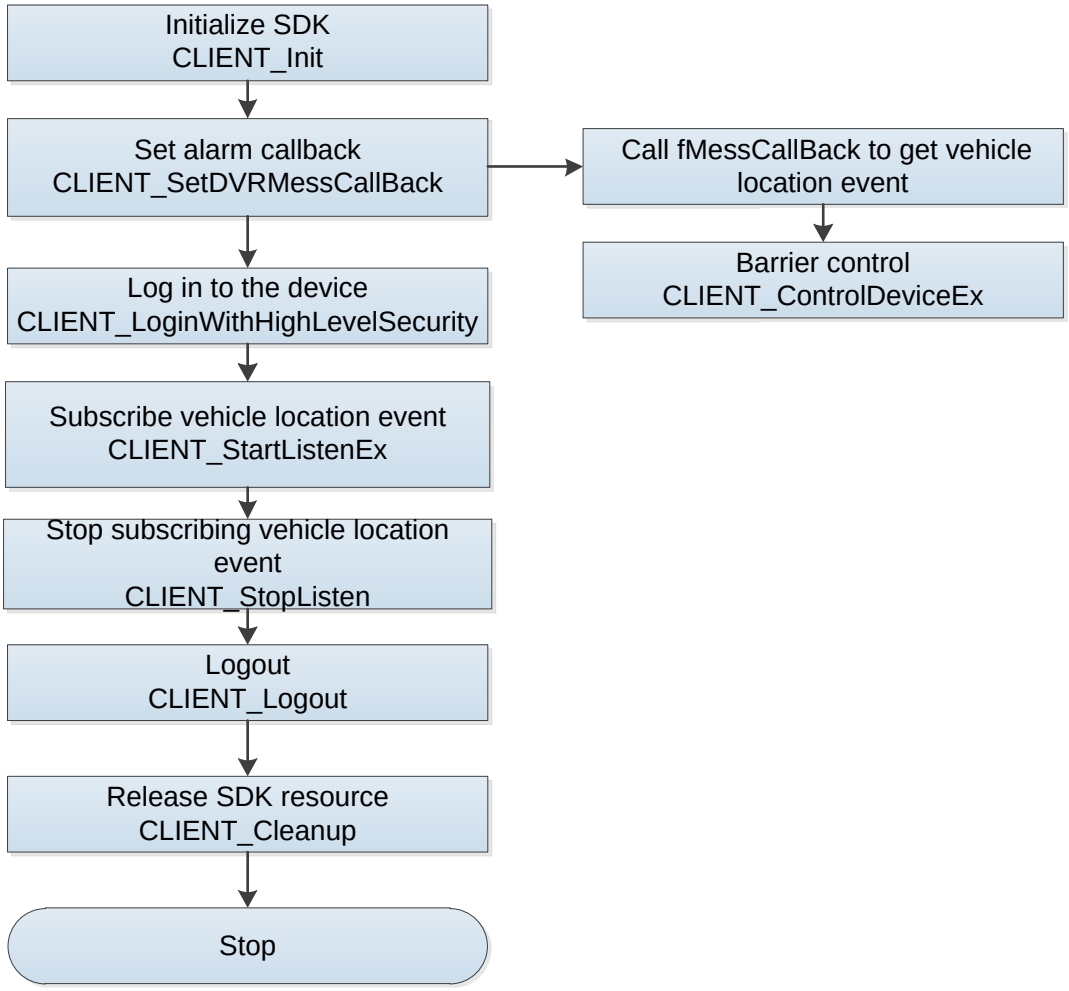
Step 3 Call CLIENT_GetConfig to get the configuration of the period of barrier normally open enable or barrier normally open mode.

Step 4 Call CLIENT_Logout to log out of the device.

Step 5 Call CLIENT_Cleanup to release SDK resource.

2.3.1.3.3 Vehicle Location Event links Barrier Control

Figure 2-18 Vehicle location event linking barrier control



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_SetDVRMessCallBack to set alarm callback function. When vehicle location comes, call CLIENT_ControlDeviceEx to open barrier gate.

Step 3 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 4 Call CLIENT_StartListenEx to subscribe vehicle location event.

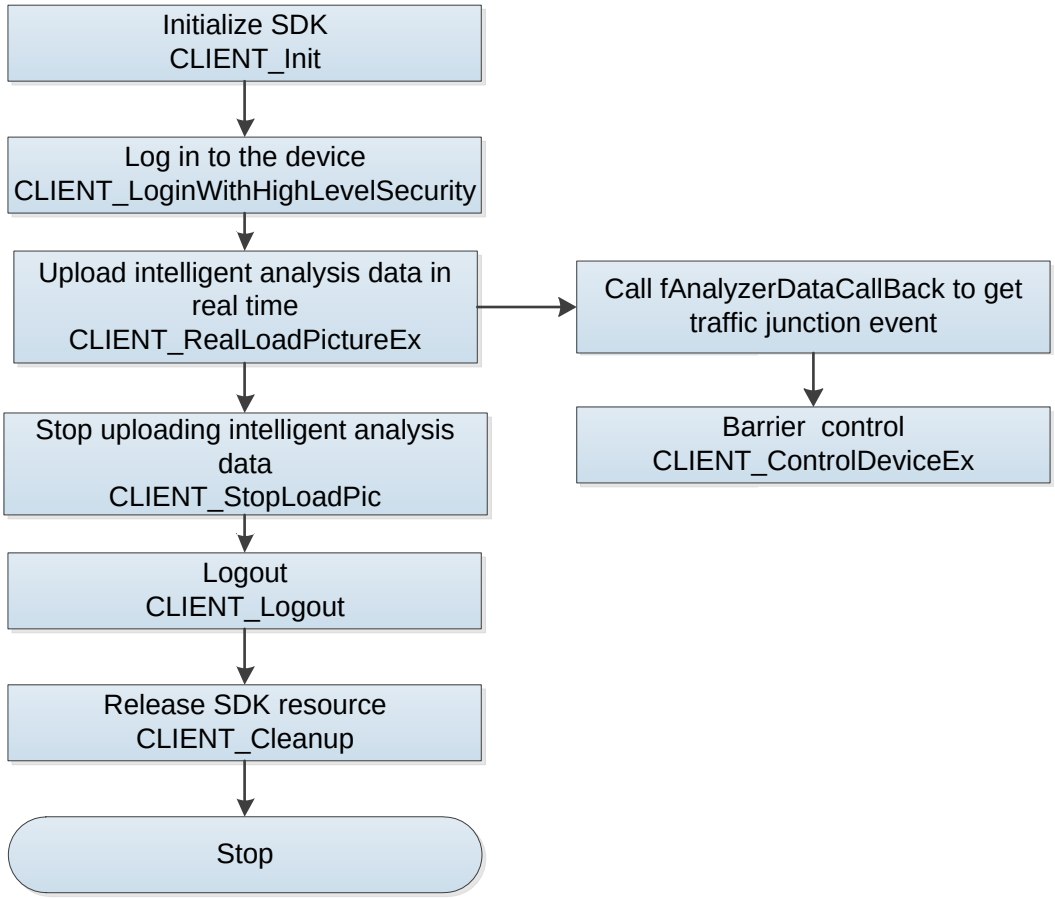
Step 5 Call CLIENT_StopListen to stop subscribing vehicle location event.

Step 6 Call CLIENT_Logout to log out of the device.

Step 7 Call CLIENT_Cleanup to release SDK resource.

2.3.1.3.4 Traffic Junction Event links Barrier Control

Figure 2-19 Traffic junction event linking barrier control



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_RealLoadPictureEx to subscribe traffic junction event. When an event is triggeried fAnalyzerDataCallBack calls CLIENT_ControlDeviceEx to open the barrier gate.

Step 4 Call CLIENT_StopLoadPic to stop subscribing traffic junction event.

Step 5 Call CLIENT_Logout to log out of the device.

Step 6 Call CLIENT_Cleanup to release SDK resource.

2.3.1.4 Example Code

2.3.1.4.1 Manual Barrier Control

```
int main()
{
    ............
    NET_CTRL_OPEN_STROBE stuOpenStrobe = {0};
    stuOpenStrobe.dwSize = sizeof(NET_CTRL_OPEN_STROBE);
    stuOpenStrobe.nChannelId = 0;
```

```
    sprintf(stuOpenStrobe.szPlateNumber,"123456");
    //Open the barrier gate
    BOOL bRet = CLIENT_ControlDeviceEx(lLoginHandle,DH_CTRL_OPEN_STROBE,&stuOpenStrobe);
    if(FALSE == bRet)
    {
        printf("CLIENT_ControlDeviceEx: Open strobe failed! Error code %x.\n", CLIENT_GetLastError());
        return -1;
    }
    NET_CTRL_CLOSE_STROBE stuCloseStrobe = {0};
    stuCloseStrobe.dwSize = sizeof(NET_CTRL_CLOSE_STROBE);
    stuCloseStrobe.nChannelId = 0;
    //Close the barrier gate
    bRet = CLIENT_ControlDeviceEx(lLoginHandle,DH_CTRL_CLOSE_STROBE,&stuCloseStrobe);
    if(FALSE == bRet)
    {
        printf("CLIENT_ControlDeviceEx: Close strobe failed! Error code %x.\n", CLIENT_GetLastError());
        return -2;
    }
    return 0;
}
```

2.3.1.4.2 Barrier Control Configuration

```
//Get barrier gate configuration
NET_CFG_TRAFFICSTROBE_INFO m_stuTrafficstrobeInfo = {sizeof(m_ stuTrafficstrobeInfo)};
BOOL bRet = CLIENT_GetConfig(m_LoginID, NET_EM_CFG_TRAFFICSTROBE,m_nChannel,&m_
stuTrafficstrobeInfo,sizeof(m_ stuTrafficstrobeInfo), 5000);
if ( !    bRet)
{
    //Failed
}
//Set barrier gate configuration
NET_CFG_TRAFFICSTROBE_INFO m_ stuTrafficstrobeInfo = {sizeof(m_ stuTrafficstrobeInfo)};
…….
BOOL bRet = CLIENT_SetConfig(m_LoginID, NET_EM_CFG_TRAFFICSTROBE,m_nChannel,&m_
stuTrafficstrobeInfo,sizeof(m_ stuTrafficstrobeInfo), 5000);
if ( !    bRet)
{
    //Failed
}
```

2.3.1.4.3 Vehicle location Event links Opening Barrier

```
//Event callback
int CALLBACK    afMessCallBack(LONG lCommand, LLONG lLinID, char *pBuf, DWORD dwBufLen,
char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    if(lCommand == DH_ALARM_TRAFFIC_VEHICLE_POSITION) //Vehicle location event
    {
        ALARM_TRAFFIC_VEHICLE_POSITION* pstAccessInfo =
                                        (ALARM_TRAFFIC_VEHICLE_POSITION*)pBuf;
         //Then control the barrier gate through the pstAccessInfo information.
    }
}
//Set event callback
CLIENT_SetDVRMessCallBack(afMessCallBack,0);
//Subscribe vehicle location event
CLIENT_StartListenEx(lLoginHandle);
//Stop subscribing vehicle location event
CLIENT_StopListen(lLoginHandle);
```

2.3.1.4.4 Traffic Junction Event links Opening Barrier

```
int main()
{
    ............
    //Subscribe traffic junction event
    LLONG lAnalyerHandle = CLIENT_RealLoadPictureEx(lLoginHandle, 0, (DWORD)EVENT_IVS_ALL, TRUE,
AnalyzerDataCallBack, NULL, NULL);
    if(NULL == lAnalyerHandle)
    {
        printf("CLIENT_RealLoadPictureEx: failed! Error code %x.\n", CLIENT_GetLastError());
        return -1;
    }
    // Stop subscribing traffic junction event
    BOOL bRet = CLIENT_StopLoadPic(lAnalyerHandle);
    if(FALSE == bRet)
    {
        printf("CLIENT_StopLoadPic: failed! Error code %x.\n", CLIENT_GetLastError());
        return -3;
    }
    return 0;
}
```

```
// traffic junction callback
int CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    switch(dwAlarmType)
    {
        case EVENT_IVS_TRAFFICJUNCTION:
            {
                DEV_EVENT_TRAFFICJUNCTION_INFO*  pInfo  =
                (DEV_EVENT_TRAFFICJUNCTION_INFO*)pAlarmInfo;
                //Control the barrier gate according to the pInfo information.
                break;
            }
        default:
            break;
    }
    return 0;
}
```

2.3.2 Importing/Exporting Blocklist/Allowlist

2.3.2.1 Introduction

Importing or exporting the blocklist or allowlist is applicable to quick configuration of the camera. You can use the imported list only when you have configured the camera.

Applicable device: IPMECK device

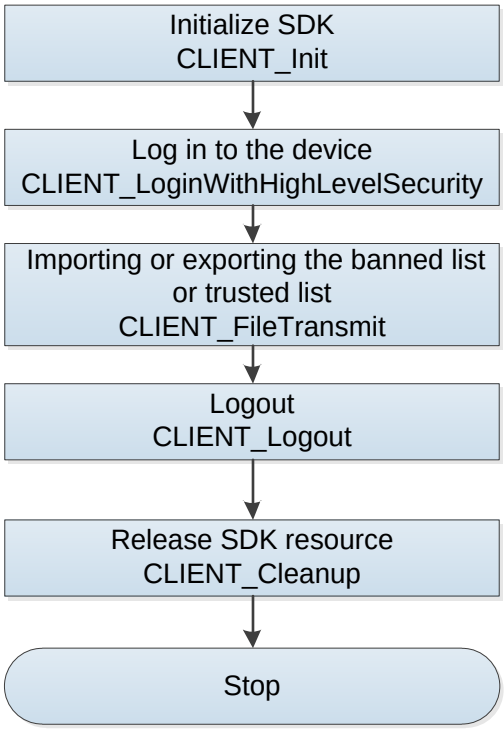
2.3.2.2 Interface Overview

Table 2-14 Importing/exporting the blocklist/allowlist interfaces

Interface	Description
CLIENT_FileTransmit	Import or export the blocklist or allowlist.

2.3.2.3 Process

Figure 2-20 Importing/exporting the blocklist/allowlist



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_FileTransmit to control importing or exporting the blocklist or allowlist.

Step 4 Call CLIENT_Logout to log out of the device.

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

Notes for Process

Keep the table head of the imported table consistent with the camera template; otherwise, the query will fail.

2.3.2.4 Example Code

```
//File transmission progress callback
void CALLBACK    bfTransFileCallBack(LLONG lHandle, int nTransType, int nState, int nSendSize, int nTotalSize, LDWORD dwUser)
{
    if (nTransType == DH_DEV_BLACKWHITE_LOAD)
    {
        if (nState == 0)
        {
```

```
                //After calling stopLoadFileTransmit, export the blocklist or the allowlist
            }
        }
        else if(nTransType == DH_DEV_BLACKWHITETRANS_SEND)
        {
            if (nState == 0)
            {
                //After calling stopSendFileTransmit, send the blocklist or allowlist
            }
        }
        //Display file transmission progress
}
//Stop exporting the blocklist or allowlist
Void stopLoadFileTransmit(LLONG lHandle )
{
LLONG nRet =
CLIENT_FileTransmit(m_lLoginHandle,DH_DEV_BLACKWHITE_LOAD_STOP,(char*)&lHandle,sizeof(LLONG),NULL,NULL,5000);
}
// Stop sending the blocklist or allowlist
void CBWListDlg::stopSendFileTransmit(LLONG lHandle )
{
        LLONG nRet =
CLIENT_FileTransmit(m_lLoginHandle,DH_DEV_BLACKWHITETRANS_STOP,(char*)&lHandle,sizeof(LLONG),NULL,NULL,5000);
}
int main()
{
        //Export the blocklist or allowlist
        DHDEV_LOAD_BLACKWHITE_LIST_INFO stulistinfo;
        CString strPath = "C:\\1\\3.CSV";
        strncpy(stulistinfo.szFile, strPath.GetBuffer(), sizeof(stulistinfo.szFile)-1);
        stulistinfo.byFileType = 1;
        LLONG nRet =
CLIENT_FileTransmit(m_lLoginHandle,DH_DEV_BLACKWHITE_LOAD,(char*)&stulistinfo,sizeof(DHDEV_LOAD_BLACKWHITE_LIST_INFO),bfTransFileCallBack,(LDWORD)this,5000);
        if (nRet <= 0)
        {
            //Failed
        }
```

```
    //Send the blocklist or allowlist
    DHDEV_BLACKWHITE_LIST_INFO stulistinfo;
    CString strPath = "C:\\1\\3.CSV";
    strncpy(stulistinfo.szFile, strPath.GetBuffer(), sizeof(stulistinfo.szFile)-1);
    stulistinfo.byFileType = 1;
    stulistinfo.byAction = 0;
    LLONG nHandle =
CLIENT_FileTransmit(m_lLoginHandle,DH_DEV_BLACKWHITETRANS_START,(char*)&stulistinfo,sizeof(DHDEV_B
LACKWHITE_LIST_INFO),bfTransFileCallBack,(LDWORD)this,5000);
    if (nHandle > 0)
    {
        LLONG nRet =
CLIENT_FileTransmit(m_lLoginHandle,DH_DEV_BLACKWHITETRANS_SEND,(char*)&nHandle,sizeof(LLONG),bfT
ransFileCallBack,(LDWORD)this,5000);
        if (nRet <= 0)
        {
            //Failed
        }
    }
    else
    {
        //Failed
    }
    return 0;
}
```

2.3.3 Voice talk

2.3.3.1 Introduction

Voice talk is used to realize the intercom between local platform and the scene where cameras installed. For example: In unattended solution, customers want to communicate the barrier abnormality with the center platform.

This section introduces how to realize voice talk between the platform and device through SDK.

2.3.3.2 Interface Overview

Table 2-15 Voice talk interfaces

Interface	Description
CLIENT_StartTalkEx	Extension interface of satrting vioce talk.
CLIENT_StopTalkEx	Extension interface of stopping vioce talk

Interface	Description
CLIENT_RecordStartEx	Extension interface of satrting client sound reording (It is valid only when the device connects to Windows paltform).
CLIENT_RecordStopEx	Extension interface of stopping client sound reording. (It is valid only when the device connects to Windows paltform).
CLIENT_TalkSendData	Send sound reording data to devices.
CLIENT_AudioDecEx	Extension interface of decoding sound reording data (It is valid only when the device is working with Windows paltform).
CLIENT_SetDeviceMode	Set voice talk woring mode of the device.
CLIENT_SetDVRMessCallBack	Set the callback of ITC requesting the platform to start voice talk event.
CLIENT_StartListenEx	Subscribe ITC requesting the platform to start voice talk event.
CLIENT_StopListen	Stop subscribing ITC requesting the platform to start voice talk event.

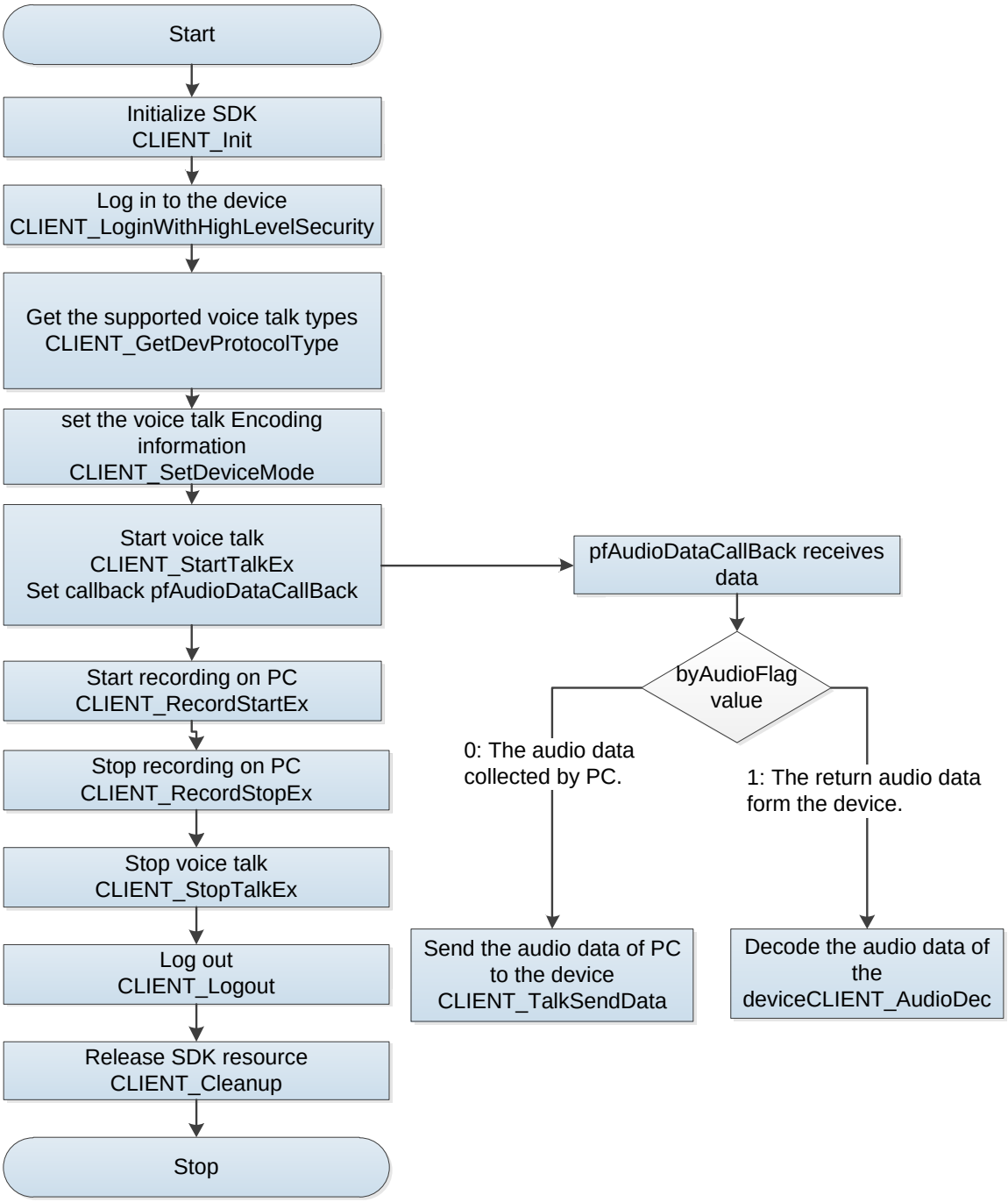
2.3.3.3 Process

2.3.3.3.1 Voice Talk Process

When SDK collects the audio data from local audio card, or SDK receives the audio data from the camera, it calls audio data callback. You can call SDK interface when calling the function to send the collected audio data to the camera, and also can call SDK interface to decode the received audio data from the camera.

- This model is valid only when working with Windows platform.
- There are voice talk (generation II) and voice talk (generation III) at present. You can call CLIENT_GetDevProtocolType to get the supported voice talk types of the device. Voice talk (generation II) and voice talk (generation III) have the same voice talk process, and different parameter configurations of CLIENT_SetDeviceMode.

Figure 2-21 voice talk (generation II)



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_GetDevProtocolType to get the supported voice talk types (generation II or generation III).

Step 4 Call CLIENT_SetDeviceMode to set the voice talk parameters.

If voice talk (generation II) is supported: Set coding mode, client mode, and speak mode. Set emType to be DH_TALK_ENCODE_TYPE, DH_TALK_CLIENT_MODE and DH_TALK_SPEAK_PARAM.

If voice talk (generation III) is supported: Set coding mode, client mode, and the parameters of voice talk (generation III). Set emType to be DH_TALK_ENCODE_TYPE, DH_TALK_CLIENT_MODE, and DH_TALK_MODE3.

Step 5 Call CLIENT_StartTalkEx to set callback and start voice talk. When call back function, call CLIENT_AudioDec to decode the audio data from the device; call CLIENT_TalkSendData to send audio data of PC to the device.

Step 6 Call CLIENT_StartTalkEx to start sound recording on PC. After calling the interface, voice talk callback of CLIENT_StartTalkEx will receive the local audio data.

Step 7 After using voice talk function, call CLIENT_RecordStopEx to stop PC sound recording.

Step 8 Call CLIENT_StopTalkEx to stop voice talk.

Step 9 Call CLIENT_Logout to log out of the device.

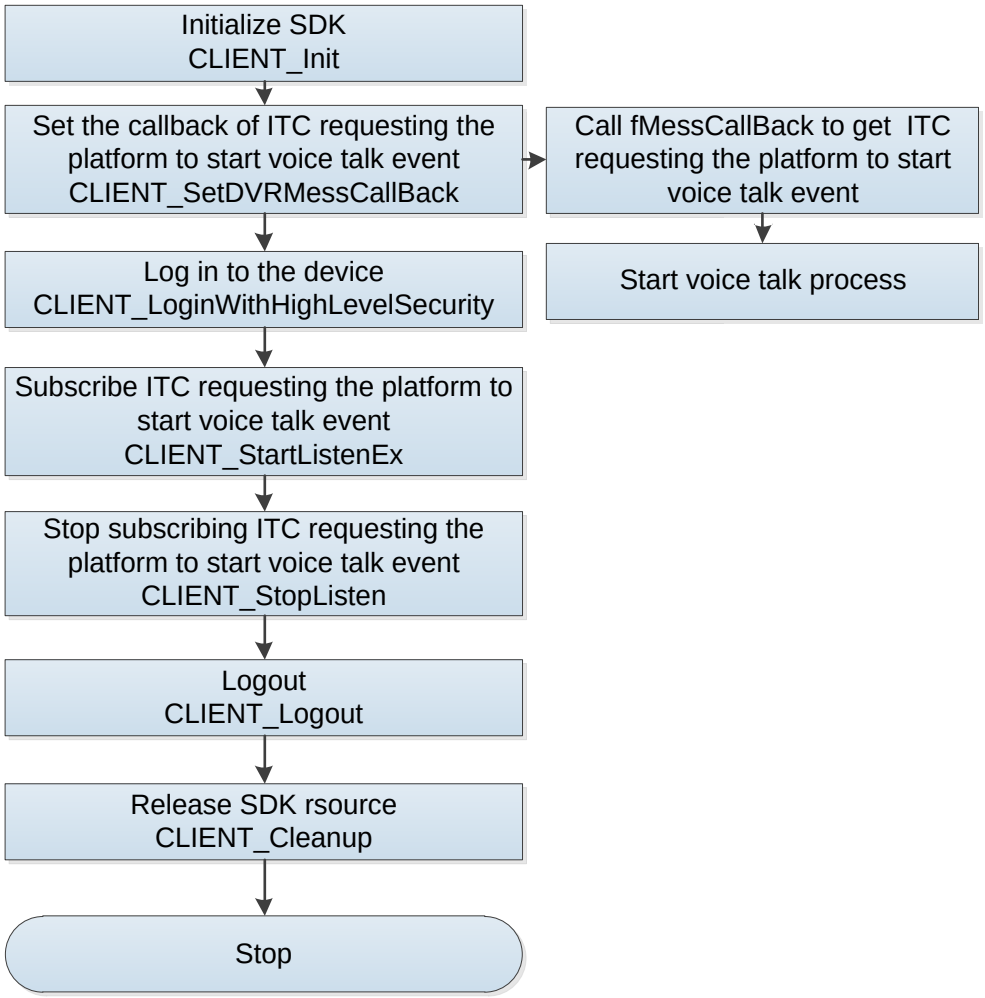
Step 10 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

Notes for Process

- Audio encoding format: The example adopts the common format PCM, SDK supports getting the supported voice talk encoding format. For the source code, see the release package on the official website. If the default PCM can meet the user's demand, no need to get the supported voice talk encoding format.
- Device has no sound: Collect audio data from audio collection devices such as microphone. Check whether the device connects to an audio collection device, and whether CLIENT_RecordStartEx interface returns.

2.3.3.3.2 ITC Requesting the Platform to Start Voice Talk Event

Figure 2-22 ITC requesting the platform to start voice talk event



Process Description

Step 1 Call CLIENT_Init to initialize SDK.
Step 2 Call CLIENT_SetDVRMessCallBack to set alarm callback. When there is requesting voice talk event, call voice talk precess.
Step 3 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
Step 4 Call CLIENT_StartListenEx to subscribe requesting voice talk event.
Step 5 Call CLIENT_StopListen to stop subscribing requesting voice talk event.
Step 6 Call CLIENT_Logout to log out of the device.
Step 7 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

2.3.3.4 Example Code

2.3.3.4.1 Voice Talk

```
//Get the supported voice talk type (generation II or generation III)
EM_DEV_PROTOCOL_TYPE emTpye = EM_DEV_PROTOCOL_UNKNOWN;
CLIENT_GetDevProtocolType(g_lLoginHandle, &emTpye);
DHDEV_TALKDECODE_INFO curTalkMode = {0};
curTalkMode.encodeType = DH_TALK_PCM;
curTalkMode.nAudioBit = 16;
curTalkMode.dwSampleRate = 8000;
curTalkMode.nPacketPeriod = 25;
CLIENT_SetDeviceMode(lLoginHandle, DH_TALK_ENCODE_TYPE, &curTalkMode); //Set voice talk encoding format
CLIENT_SetDeviceMode(lLoginHandle, DH_TALK_CLIENT_MODE, NULL);//Set client voice talk
//Set parameters according to the supported voice talk type
if (emTpye == EM_DEV_PROTOCOL_V3) // Voice talk (generation III) requests this setting, and voice talk (generation II) does not request this setting
{
    NET_TALK_EX stuTalk = {sizeof(stuTalk)};
        stuTalk.nAudioPort = RECEIVER_AUDIO_PORT;    //Custom receiving port        stuTalk.nChannel    = 0;
        stuTalk.nWaitTime  = 5000;
        CLIENT_SetDeviceMode(m_lLoginHandle, DH_TALK_MODE3, &stuTalk)
}
//Start voice talk
lTalkHandle = CLIENT_StartTalkEx(lLoginHandle, AudioDataCallBack, (LDWORD)NULL);
//Start local sound recording
CLIENT_RecordStartEx(lLoginHandle);
//Stop local sound recording
CLIENT_RecordStopEx(lLoginHandle)
//Stop voice talk
```

```
CLIENT_StopTalkEx(lTalkHandle);
//Voice talk callback data processing
void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE
byAudioFlag, LDWORD dwUser)
{
if(0 == byAudioFlag)
    {
        //Send the audio card data detected by PC to the device
        CLIENT_TalkSendData(lTalkHandle, pDataBuf, dwBufSize);
    }
else if(1 == byAudioFlag)
    {
        //Send the audio data from the device to SDK for decoding play
        CLIENT_AudioDec(pDataBuf, dwBufSize);
    }
}
```

2.3.3.4.2 ITC Requesting the Platform to Start Voice Talk Event

```
// Call ITC requesting the platform to start voice talk event
int CALLBACK    afMessCallBack(LONG lCommand, LLONG lLinID, char *pBuf, DWORD dwBufLen,
char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    if(lCommand == DH_ALARM_TALKING_INVITE) // ITC requesting the platform to start voice talk event
    {
        //Callback voice talk process 2.3.3.4.1
    }
}
// Call ITC requesting the platform to start voice talk event
CLIENT_SetDVRMessCallBack(afMessCallBack,0);
// Subscribe ITC requesting the platform to start voice talk event
CLIENT_StartListenEx(lLoginHandle);
// Stop subscribing ITC requesting the platform to start voice talk event
CLIENT_StopListen(lLoginHandle);
```

2.3.4 Dot-matrix Display Content Control and Broadcast

2.3.4.1 Overview

Dot-matrix display has 2 categories:

- Products before September 2020 only supports character control, including sending vehicle passing characters, screen filling characters and displaying according to character content. For details, see "2.3.5 Dot-matrix Display Character Control".
- Products released in 2020 (QR code available) support complete screen and audio entrusting. You can fully control the display content and audio broadcast through SDK interfaces.

- Make sure that the entrusting mode is supported and enabled on devices. You can configure on web interface or LED screen.

2.3.4.2 Interface

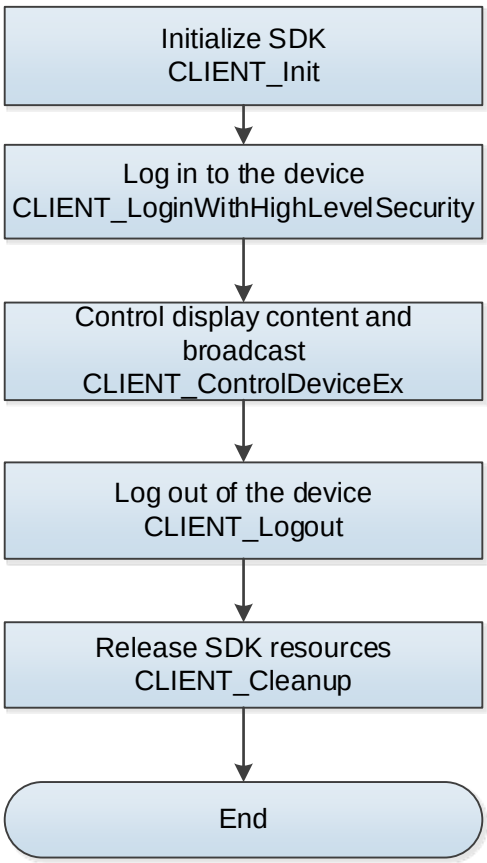
Table 2-16 Interface Information

Port	Description
CLIENT_ControlDeviceEx	Dot-matrix Display Content Control and Broadcast

2.3.4.3 Process

For the process of display content control and broadcast, See Figure 2-19.

Figure 2-23 Process of display content control and broadcast



Process Description

Step 1 Complete SDK initialization.

Step 2 After successful initialization, call CLIENT_LoginWithHighLevelSecurity to log in to the device.
Step 3 Call CLIENT_ControlDeviceEx to control display content and broadcast.
Step 4 Call CLIENT_Logout to log out of the device.
Step 5 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

Notes

None.

2.3.4.4 Example Code

```
// display content. Plate number + parking duration + parking fee + local time
UINT nScreenShowCount = 4;
stuIn.nScreenShowInfoNum = __min(nScreenShowCount, _countof(stuIn.stuScreenShowInfo));
stuIn.stuScreenShowInfo[0].nScreenNo = 0; // first row
stuIn.stuScreenShowInfo[0].emTextType = EM_SCREEN_TEXT_TYPE_ORDINARY;
stuIn.stuScreenShowInfo[0].emTextColor = EM_SCREEN_TEXT_COLOR_GREEN;
stuIn.stuScreenShowInfo[0].emTextRollMode = EM_SCREEN_TEXT_ROLL_MODE_NO;
stuIn.stuScreenShowInfo[0].nRollSpeed = 1;
std::string strText1 = "ZA8888";
memcpy(stuIn.stuScreenShowInfo[0].szText, strText1.c_str(), strText1.length());
stuIn.stuScreenShowInfo[1].nScreenNo = 1; // second row
stuIn.stuScreenShowInfo[1].emTextType = EM_SCREEN_TEXT_TYPE_ORDINARY;
stuIn.stuScreenShowInfo[1].emTextColor = EM_SCREEN_TEXT_COLOR_GREEN;
stuIn.stuScreenShowInfo[1].emTextRollMode = EM_SCREEN_TEXT_ROLL_MODE_NO;
stuIn.stuScreenShowInfo[1].nRollSpeed = 1;
std::string strText2 = "Parking for 30 minutes";
memcpy(stuIn.stuScreenShowInfo[1].szText, strText2.c_str(), strText2.length());
stuIn.stuScreenShowInfo[2].nScreenNo = 2; // third row
stuIn.stuScreenShowInfo[2].emTextType = EM_SCREEN_TEXT_TYPE_ORDINARY;
stuIn.stuScreenShowInfo[2].emTextColor = EM_SCREEN_TEXT_COLOR_GREEN;
stuIn.stuScreenShowInfo[2].emTextRollMode = EM_SCREEN_TEXT_ROLL_MODE_NO;
stuIn.stuScreenShowInfo[2].nRollSpeed = 1;
std::string strText3 = "Charging 10 yuan";
memcpy(stuIn.stuScreenShowInfo[2].szText, strText3.c_str(), strText3.length());
stuIn.stuScreenShowInfo[3].nScreenNo = 3; // fourth row
stuIn.stuScreenShowInfo[3].emTextType = EM_SCREEN_TEXT_TYPE_LOCAL_TIME;
stuIn.stuScreenShowInfo[3].emTextColor = EM_SCREEN_TEXT_COLOR_GREEN;
stuIn.stuScreenShowInfo[3].emTextRollMode = EM_SCREEN_TEXT_ROLL_MODE_NO;
stuIn.stuScreenShowInfo[3].nRollSpeed = 1;
std::string strText4 = "%Y-%M-%D %H:%m:%S";
```

```
memcpy(stuIn.stuScreenShowInfo[3].szText, strText4.c_str(), strText4.length());
// audio broadcast
// example code. Plate number + parking duration + parking fee
UINT nBroadCastCount = 3;
stuIn.nBroadcastInfoNum = __min(nBroadCastCount,_countof(stuIn.stuScreenShowInfo));
stuIn.stuBroadcastInfo[0].emTextType = EM_BROADCAST_TEXT_TYPE_PLATE_NUMBER;
std::string strVoice = “ZA8888";
memcpy(stuIn.stuBroadcastInfo[0].szText, strVoice.c_str(), strVoice.length());
stuIn.stuBroadcastInfo[1].emTextType = EM_BROADCAST_TEXT_TYPE_TIME;
std::string strVoice1 = "Parking for 30 minutes";
memcpy(stuIn.stuBroadcastInfo[1].szText, strVoice1.c_str(), strVoice1.length());
stuIn.stuBroadcastInfo[2].emTextType = EM_BROADCAST_TEXT_TYPE_NUMBER_STRING;
std::string strVoice2 = "Charging 10 yuan";
memcpy(stuIn.stuBroadcastInfo[2].szText, strVoice2.c_str(), strVoice2.length());
BOOL bRet = CLIENT_ControlDeviceEx(m_lLoginID, DH_CTRL_SET_PARK_CONTROL_INFO, &stuIn, &stuOut,
3000);
if (!bRet) {
    printf("Failed to set parking control info. Error Code 0x%x.\n", CLIENT_GetLastError());
}
```

2.3.5 Dot-matrix Display Character Control

2.3.5.1 Introduction

There are two statuses of dot-matrix display: Car pass status and normal status.

- Car pass status: When a car passes the access, the camera captures it, which triggers the event, and the car pass status will be activated, and it lasts a certain period (the period can be set). When the car is passing, the dot-matrix display displays plate number, parking card validity and custom data, and it broadcasts the displayed data automatically.
- Normal status: The status appears after car pass status, and dot-matrix display displays the available space information.

- When the mornal status display information is issues in car pass status, it cannot be displayed immediately. The information will be displayed after the display enters normal status.
- When the car pass status display information is issues in normal status, it cannot be displayed immediately. The information will be displayed after the display enters car pass status.

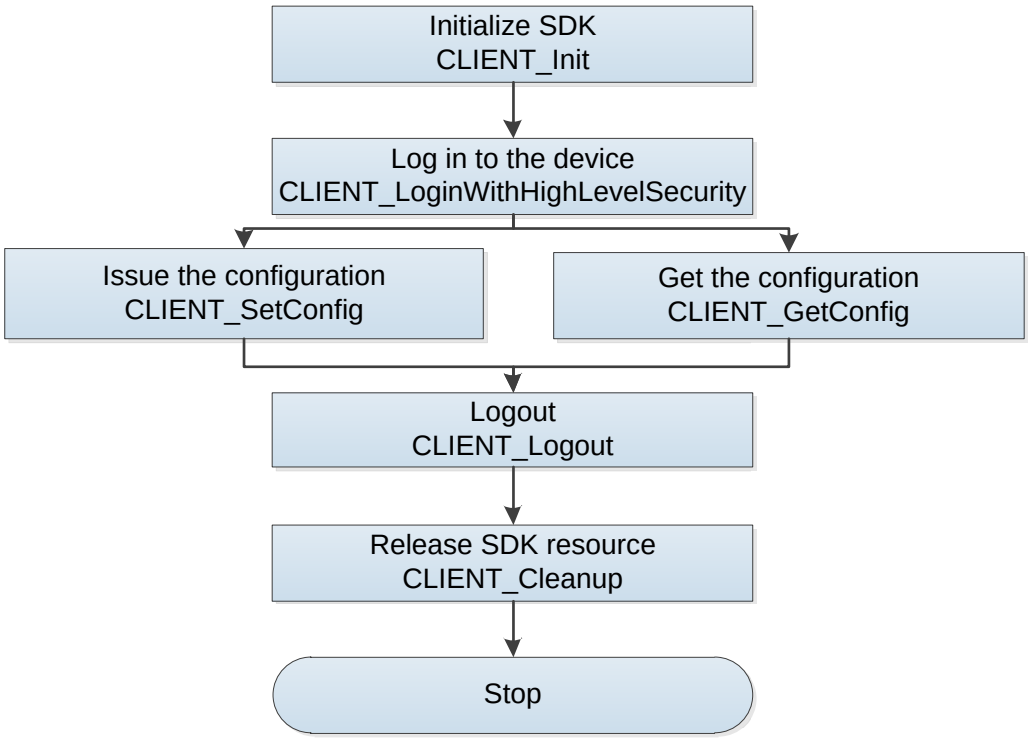
2.3.5.2 Interface Overview

Table 2-17 Dot-matrix display interfaces

Interface	Description
CLIENT_GetConfig	Get the LED Lattice screen display configuration
CLIENT_SetConfig	Set the LED Lattice screen display configuration

2.3.5.3 Process

Figure 2-24 Dot-matrix display character control



Process Description

Setting

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_SetConfig to set the LED Lattice screen display configuration.

Step 4 Call CLIENT_Logout to log out of the device.

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

Getting

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_GetConfig to get the LED Lattice screen display configuration.

Step 4 Call CLIENT_Logout to log out of the device.

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

2.3.5.4 Example Code

```
//Get the LED Lattice screen display configuration
NET_CFG_TRAFFIC_LATTICE_SCREEN_INFO m_stuTrafficscreenInfo= {sizeof(m_ stuTrafficscreenInfo)};
```

```
BOOL bRet = CLIENT_GetConfig(m_LoginID, NET_EM_CFG_TRAFFIC_LATTICE_SCREEN,m_nChannel,&m_
stuTrafficscreenInfo,sizeof(m_ stuTrafficscreenInfo), 5000);
if ( !    bRet)
{
        //Failed
}
//Set the LED Lattice screen display configuration
NET_CFG_TRAFFIC_LATTICE_SCREEN_INFO m_ stuTrafficscreenInfo= {sizeof(m_ stuTrafficscreenInfo)};
……
BOOL bRet = CLIENT_SetConfig(m_LoginID, NET_EM_CFG_TRAFFIC_LATTICE_SCREEN,m_nChannel,&m_
stuTrafficscreenInfo,sizeof(m_ stuTrafficscreenInfo), 5000);
if ( !    bRet)
{
        //Failed
}
```

2.3.6 Parking Space Indicator Configuration

2.3.6.1 Introduction

Get the supervision status of the indicator group.

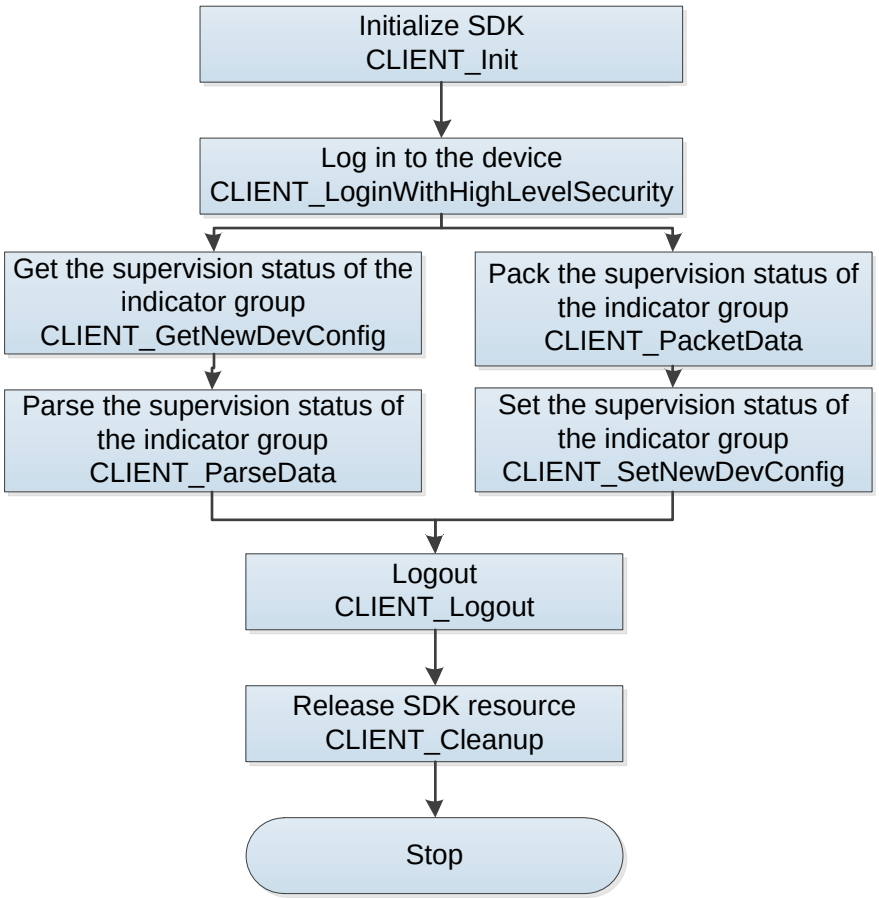
2.3.6.2 Interface Overview

Table 2-18 Parking space indicator configuration interfaces

Interface	Description
CLIENT_SetNewDevConfig	Set the supervision status of the indicator group
CLIENT_GetNewDevConfig	Get the supervision status of the indicator group
CLIENT_ParseData	Parse the supervision status of the indicator group
CLIENT_PacketData	Pack the supervision status of the indicator group

2.3.6.3 Process

Figure 2-25 Parking space indicator configuration



Process Description

Getting

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_GetNewDevConfig to get the parking space indicator configuration.

Step 4 Call CLIENT_ParseData to parse the parking space indicator light configuration.

Step 5 Call CLIENT_Logout to log out of the device.

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

Setting

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_PackatData to pack the parking space indicator configuration.

Step 4 Call CLIENT_SetNewDevConfig to set the parking space indicator light configuration.

Step 5 Call CLIENT_Logout to log out of the device.

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

2.3.6.4 Example Code

```
//Set parking space indicator configuration
```

```
CFG_PARKING_SPACE_LIGHT_GROUP_INFO_ALL stuInfo = {0};
stuInfo. nCfgNum= m_ nCfgNum;
for (int i = 0;i<m_ nCfgNum;i++)
{
	stuInfo.stuLightGroupInfo.bEnable = TRUE;
	......
}
BOOL bRet = CLIENT_PacketData(CFG_CMD_PARKING_SPACE_LIGHT_GROUP,(LPVOID)&stuInfo, sizeof(stuInfo),
szJsonBuf, sizeof(szJsonBuf));
if (bRet)
{
	int nerror = 0;
	int nrestart = 0;
	int nChannelID = -1;
	bRet = CLIENT_SetNewDevConfig(m_iLoginID, CFG_CMD_PARKING_SPACE_LIGHT_GROUP, nChannelID,
szJsonBuf, 512*40, &nerror, &nrestart, 3000);
}
//Get parking space indicator configuration
char szJsonBuf[1024 * 40] = {0};
int nerror = 0;
int nChannel = -1;
BOOL ret = CLIENT_GetNewDevConfig(m_iLoginID,
CFG_CMD_PARKING_SPACE_LIGHT_GROUP,nChannel,szJsonBuf,1024*40,&nerror,3000);
if (0 != ret)
{
	CFG_PARKING_SPACE_LIGHT_GROUP_INFO_ALL stuInfo = {0};
	DWORD dwRetLen = 0;
	ret =
CLIENT_ParseData(CFG_CMD_PARKING_SPACE_LIGHT_GROUP,szJsonBuf,(char*)&stuInfo,sizeof(stuInfo),&dwRe
tLen);
	if (!ret)
	{
		//Failed
		return ;
	}
}
else
{
	//Failed
	return ;
```

```
}
```

2.3.7 Parking Space Status Indicator Configuration

2.3.7.1 Introduction

Configure parking space status indicator.

- Set getting the indicator color of parking space free status.
- Set getting the indicator color of parking space full status.
- Set getting the indicator color of single network port exception.
- Set getting the indicator color of dual network port exception.

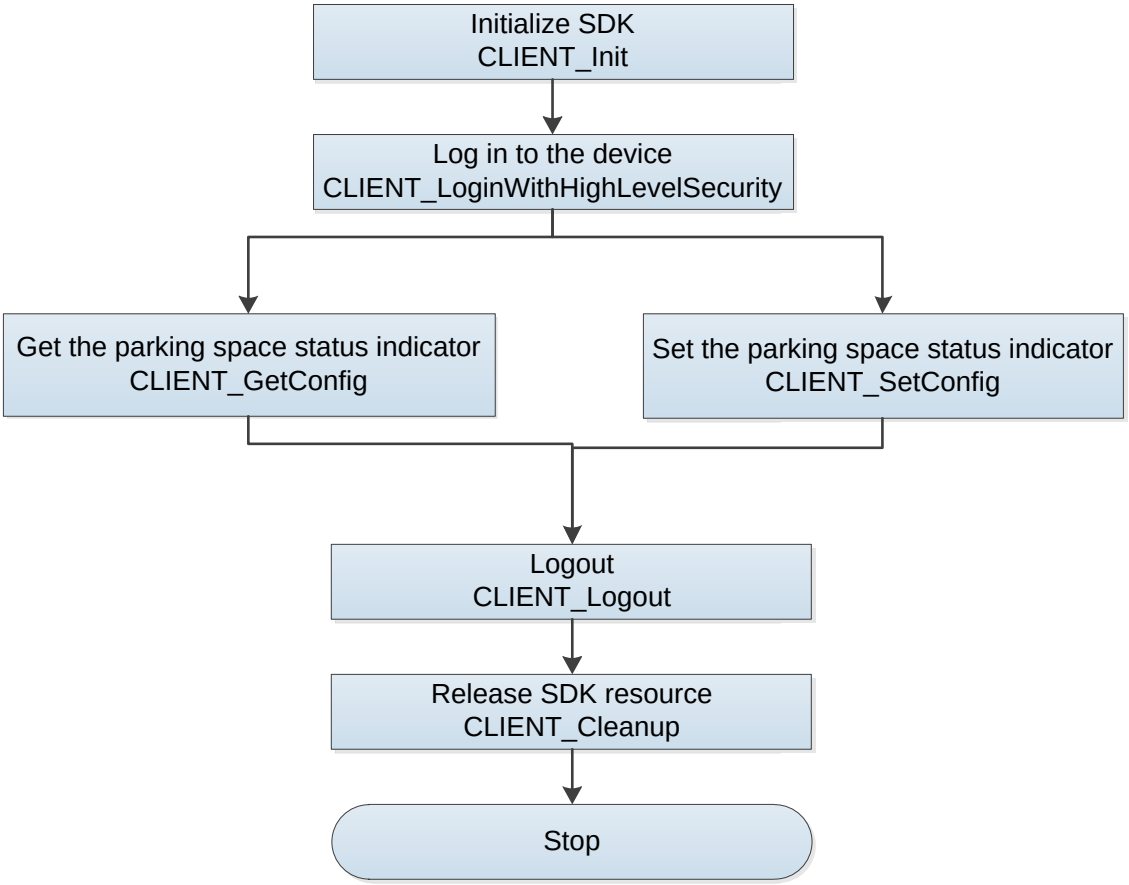
2.3.7.2 Interface Overview

Table 2-19 Parking space status indicator configuration interfaces

Interface	Description
CLIENT_SetConfig	Set the parking space status indicator
CLIENT_GetConfig	Get the parking space status indicator

2.3.7.3 Process

Figure 2-26 Parking space status indicator configuration



Process Description

Getting

Step 1 Call CLIENT_Init to initialize SDK.
Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
Step 3 Call CLIENT_GetConfig to get the parking space status indicator configuration.
Step 4 Call CLIENT_Logout to log out of the device.
Step 5 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

Setting

Step 1 Call CLIENT_Init to initialize SDK.
Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.
Step 3 Call CLIENT_SetConfig to set the parking space status indicator configuration.
Step 4 Call CLIENT_Logout to log out of the device.
Step 5 After using all SDK functions, call CLIENT_Cleanup to release SDK resource.

```
//Set the parking space status indicator configuration
NET_PARKINGSPACELIGHT_STATE_INFO stuInfo;
memset(&stuInfo, 0, sizeof(stuInfo));
stuInfo.dwSize = sizeof(stuInfo);
stuInfo.stuSpaceFreeInfo.nRed = 1;          //Set the status indicator to be red normally on for the free parking space
BOOL bRet = CLIENT_SetConfig(m_lLoginID, NET_EM_CFG_PARKINGSPACELIGHT_STATE, -1, &stuInfo, sizeof(stuInfo));
if (bRet == FALSE)
{
    //Failed
    return;
}
// Get the parking space status indicator configuration
NET_PARKINGSPACELIGHT_STATE_INFO stuInfo;
memset(&stuInfo, 0, sizeof(stuInfo));
stuInfo.dwSize = sizeof(stuInfo);
BOOL bRet = CLIENT_GetConfig(m_lLoginID, NET_EM_CFG_PARKINGSPACELIGHT_STATE, -1, &stuInfo, sizeof(stuInfo));
if (bRet == FALSE)
{
    //Failed
    return;
}
```

2.3.7.4 Example Code

```
//Set corresponding parking space indicator for parking space status
```

```
NET_PARKINGSPACELIGHT_STATE_INFO stuInfo;
memset(&stuInfo, 0, sizeof(stuInfo));
stuInfo.dwSize = sizeof(stuInfo);
stuInfo.stuSpaceFreeInfo.nRed = 1;          //Set vacant parking space toSolid Red
BOOL bRet = CLIENT_SetConfig(m_lLoginID, NET_EM_CFG_PARKINGSPACELIGHT_STATE, -1, &stuInfo, sizeof(stuInfo));
if (bRet == FALSE)
{
    //Failed to set
    return;
}
// Get corresponding parking space indicator configuration for parking space status
NET_PARKINGSPACELIGHT_STATE_INFO stuInfo;
memset(&stuInfo, 0, sizeof(stuInfo));
stuInfo.dwSize = sizeof(stuInfo);
BOOL bRet = CLIENT_GetConfig(m_lLoginID, NET_EM_CFG_PARKINGSPACELIGHT_STATE, -1, &stuInfo, sizeof(stuInfo));
if (bRet == FALSE)
{
    //Failed to get
    return;
}
```

2.4 Device Configuration

2.4.1 Auto registration

2.4.1.1 Introduction

Users can configure automatic registration information of the device including enabling automatic registration, device ID and server by calling SDK interface.

2.4.1.2 Interface Overview

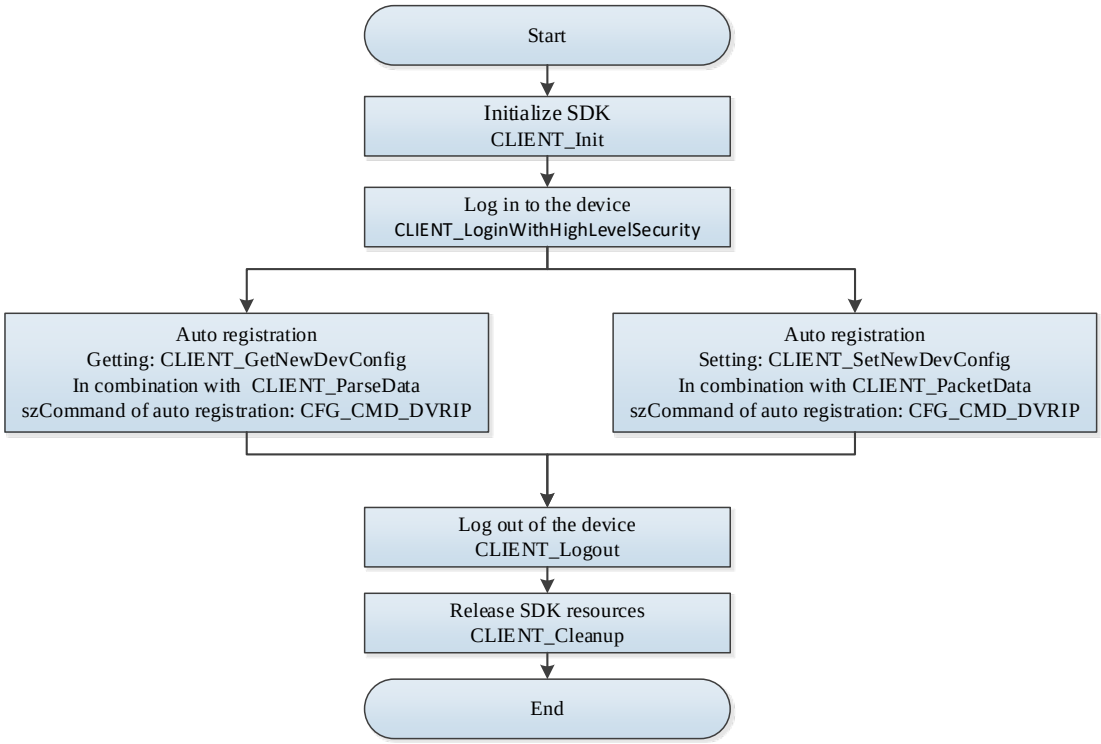
Table 2-20 Auto registration configuration interfaces

Interfaces	Description
CLIENT_GetNewDevConfig	Search for configuration information.
CLIENT_ParseData	Parse the configuration information having been searched.
CLIENT_SetNewDevConfig	Set configuration information.

Interfaces	Description
CLIENT_PacketData	Pack the configuration information to be set into the string format.

2.4.1.3 Process

Figure 2-27 Auto registration



Process Description

Step 1 Call CLIENT_Init function to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity function to log in to the device.

Step 3 Auto registration configuration

- Call CLIENT_GetNewDevConfig and CLIENT_ParseData to search for auto registration configuration.
  - szCommand: CFG_CMD_DVRIP.
  - pBuf: CFG_DVRIP_INFO.
- Call CLIENT_SetNewDevConfig and CLIENT_PacketData to set automatic registration configuration.
  - szCommand: CFG_CMD_DVRIP.
  - pBuf: CFG_DVRIP_INFO.

Step 4 Call the CLIENT_Logout function to log out of the device.

Step 5 Call CLIENT_Cleanup function to release SDK resources.

2.4.1.4 Example Code

```
// Get auto registration network configuration information
char * szOut1 = new char[1024*32];
```

```
CFG_DVRIP_INFO stOut2 = {sizeof(stOut2)};
int nError = 0;
BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_DVRIP, 0, szOut1, 1024*32,
&nError, 3000);
if(bRet){
    BOOL bRet1 = CLIENT_ParseData(CFG_CMD_DVRIP, szOut1, &stOut2, sizeof(CFG_NTP_INFO),
NULL);
}
else{
    printf("parse failed!!!");
}
// Set auto registration network configuration information
char * szOut = new char[1024*32];
stOut2.nTcpPort = 46650;
BOOL bRet0 = CLIENT_PacketData(CFG_CMD_DVRIP, (char *)&stOut2, sizeof(CFG_DVRIP_INFO), szOut,
1024*32);
if(bRet)
{
    BOOL bRet1 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_DVRIP, 0, szOut, 1024*32,
NULL, NULL, 3000);
}
```

2.4.2 Device Logs

2.4.2.1 Introduction

Users can call SDK interface to search for the operation logs of the access control device by specifying the log type or search number, or searching by pages.

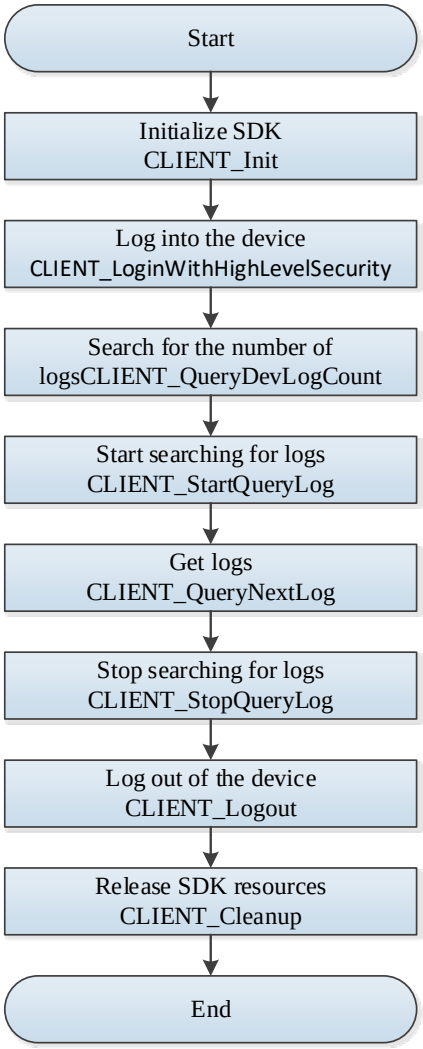
2.4.2.2 Interface Overview

Table 2-21 Device log interfaces

Interface	Description
CLIENT_QueryDevLogCount	Search for the number of device logs
CLIENT_StartQueryLog	Start searching for logs
CLIENT_QueryNextLog	Get logs
CLIENT_StopQueryLog	Stop searching for logs

2.4.2.3 Process

Figure 2-28 Device logs



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_QueryDevLogCount to set the number of logs to be searched.

Step 4 Call CLIENT_StartQueryLog to start searching for logs.

- pInParam: NET_IN_START_QUERYLOG.
- pOutParam: NET_OUT_START_QUERYLOG.

Step 5 Call CLIENT_QueryNextLog to get logs.

- pInParam: NET_IN_QUERYNEXTLOG.
- pOutParam：NET_OUT_QUERYNEXTLOG.

Step 6 Call CLIENT_StopQueryLog to stop searching for logs.

Step 7 Call CLIENT_Logout to log out of the device.

Step 8 Call CLIENT_Cleanup to release SDK resources.

2.4.2.4 Example Code

```
// Start searching for logs
NET_IN_START_QUERYLOG stuIn = {sizeof(stuIn)};
NET_OUT_START_QUERYLOG stuOut = {sizeof(stuOut)};
LLONG lLogID = CLIENT_StartQueryLog(m_lLoginId, &stuIn, &stuOut, 5000);
// Get logs
NET_IN_QUERYNEXTLOG stuIn = {sizeof(stuIn)};
stuIn.nGetCount = m_nMaxPageSize;
NET_OUT_QUERYNEXTLOG stuOut = {sizeof(stuOut)};
stuOut.nMaxCount = 60;
stuOut.pstuLogInfo = new NET_LOG_INFO[60];
if (NULL == stuOut.pstuLogInfo)
{
    return -1;
}
memset(stuOut.pstuLogInfo, 0, sizeof(NET_LOG_INFO) * m_nMaxPageSize);
for (int i = 0; i < m_nMaxPageSize; i++)
{
    stuOut.pstuLogInfo[i].dwSize = sizeof(NET_LOG_INFO);
    stuOut.pstuLogInfo[i].stuLogMsg.dwSize = sizeof(NET_LOG_MESSAGE);
}
BOOL bRet = CLIENT_QueryNextLog(m_lLogID, &stuIn, &stuOut, 5000);
// Stop searching for logs
BOOL bRet0 = CLIENT_StopQueryLog(m_lLogID);
```

2.4.2.5 Network Time Protocol (NTP) Server and Time Zone Configuration

2.4.2.5.1 Introduction

Users can get and configure NTP server and time zone by calling SDK interface.

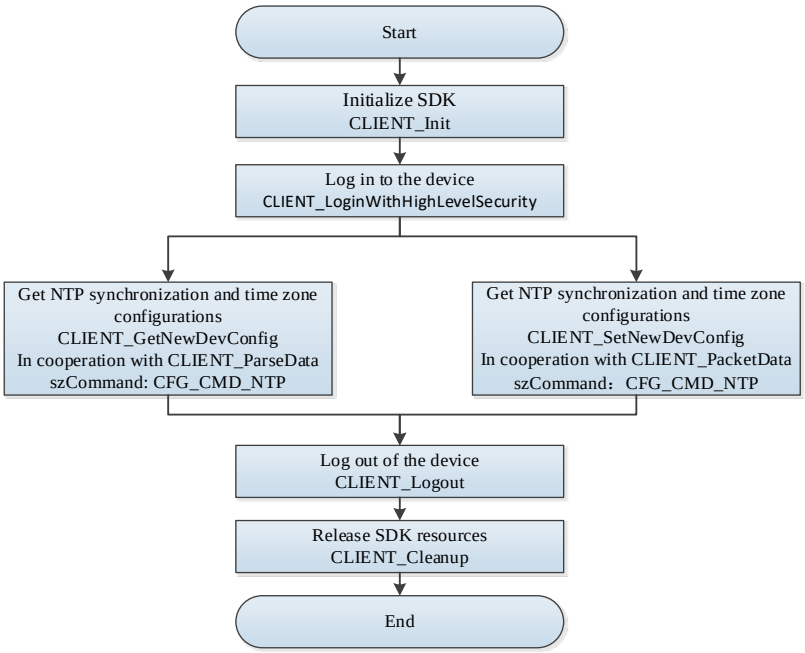
2.4.2.5.2 Interface Overview

Table 2-22 NTP server and time zone configuration interface

Interfaces	Description
CLIENT_GetNewDevConfig	Search for the configuration information.
CLIENT_ParseData	Parse the configuration information having been searched.
CLIENT_SetNewDevConfig	Set the configuration information.
CLIENT_PacketData	Pack the configuration information to be set into the string format.

2.4.2.5.3 Process

Figure 2-29 NTP time synchronization



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_GetNewDevConfig and CLIENT_ParseData to search for NTP time synchronization and time zone configuration of the access control.

- szCommand：CFG_CMD_NTP.
- pBuf：CFG_NTP_INFO.

Step 4 Call CLIENT_GetNewDevConfig and CLIENT_ParseData to search for NTP time synchronization and time zone configuration of the access control.

- szCommand：CFG_CMD_NTP.
- pBuf: CFG_NTP_INFO.

Step 5 Call CLIENT_Logout to log out of the device.

Step 6 Call CLIENT_Cleanup to release SDK resources.

2.4.2.5.4 Example Code

```
// Set NTP time synchronization and time zone configuration information.
char * szOut1 = new char[1024*32];
    CFG_NTP_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_NTP, 0, szOut1, 1024*32,
&nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_NTP, szOut1, &stOut2, sizeof(CFG_NTP_INFO),
NULL);
    }
    else{
```

```
            printf("parse failed!!!");
        }
// Set NTP time synchronization and time zone configuration information
        char * szOut = new char[1024*32];
        stOut2.bEnable = TRUE;
        BOOL bRet0 = CLIENT_PacketData(CFG_CMD_NTP, (char *)&stOut2, sizeof(CFG_NTP_INFO), szOut, 1024*32);
        if(bRet)
        {
            BOOL bRet1 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_NTP, 0, szOut, 1024*32, NULL, NULL, 3000);
        }
```

2.4.3 Get Remote Device Information

2.4.3.1 Introduction

To get information about the remote device.

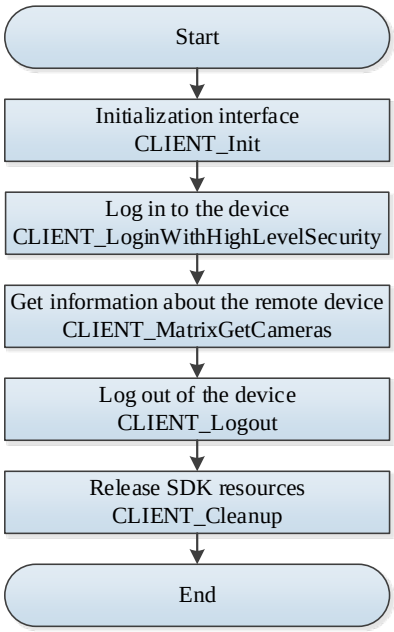
2.4.3.2 Interface Overview

Table 2-23 Interfaces for getting remote device information

Interface	Description
CLIENT_MatrixGetCameras	Get information about the remote device, including the device model, IP address and more.

2.4.3.3 Process

Figure 2-30 Get remote device information



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_MatrixGetCameras to get information about the remote device, including the device model, IP address and more.

Step 4 Call CLIENT_Logout to log out of the device.

Step 5 Call CLIENT_Cleanup to release SDK resources.

2.4.3.4 Example Code

```
// Interface No.1for getting information about the remote device
// nChanNum: The number of channels that the login interface return.
NET_IN_GET_CAMERA_STATEINFO stuInfo = { sizeof(NET_IN_GET_CAMERA_STATEINFO), TRUE };
NET_OUT_GET_CAMERA_STATEINFO stuOutInfo = { sizeof(NET_OUT_GET_CAMERA_STATEINFO) };
NET_CAMERA_STATE_INFO* pstuArrayStatInfo = new NET_CAMERA_STATE_INFO[nChanNum];
memset(pstuArrayStatInfo,0,sizeof(NET_CAMERA_STATE_INFO)*nChanNum);
stuOutInfo.nMaxNum = nChanNum;
stuOutInfo.pCameraStateInfo = pstuArrayStatInfo;
printf("State(0:UNKNOWN,1:CONNECTING,2:CONNECTED,3:UNCONNECT,4:EMPTY,5:DISABLE).\n");
BOOL bRet = CLIENT_QueryDevInfo(lLoginHandle, NET_QUERY_GET_CAMERA_STATE, &stuInfo,
&stuOutInfo, NULL, 2000);
if (bRet)
{
printf("CLIENT_QueryDevInfo NET_QUERY_GET_CAMERA_STATE success.\n");
for (int i = 0; i < stuOutInfo.nValidNum; i++)
```

```
{
printf("channel:%d,Status:%d.\n",
stuOutInfo.pCameraStateInfo[i].nChannel,stuOutInfo.pCameraStateInfo[i].emConnectionState);
}
}
else
{
printf("CLIENT_QueryDevInfo Failed, Last Error[%x]\n",
CLIENT_GetLastError());
}
// Interface No.2 for getting information about the remote device
DH_IN_MATRIX_GET_CAMERAS stuInParm = {sizeof(DH_IN_MATRIX_GET_CAMERAS)};
DH_OUT_MATRIX_GET_CAMERAS stuOutParam = {sizeof(DH_OUT_MATRIX_GET_CAMERAS)};
DH_MATRIX_CAMERA_INFO    stuAllmatrixcamerinfo[128] = {0};
stuOutParam.nMaxCameraCount = nChanNum; //Maximum number
stuOutParam.pstuCameras = stuAllmatrixcamerinfo;
for (int i=0;i< __min(stuOutParam.nMaxCameraCount,stuOutParam.nRetCameraCount);++i)
{
stuOutParam.pstuCameras[i].dwSize = sizeof(DH_MATRIX_CAMERA_INFO);
stuOutParam.pstuCameras[i].stuRemoteDevice.dwSize = sizeof(DH_REMOTE_DEVICE);
}
int iNumbers = 1;
// Get all valid display sources
BOOL bRet = CLIENT_MatrixGetCameras(lLoginHandle, &stuInParm, &stuOutParam);
printf("ALL the Device list Info Begin:\n");
if(bRet)
{
int iChannelNumbers =0;
        char szUserInput[32] = "";
memset(szUserInput, 0, sizeof(szUserInput));
printf("too many channels info:Input    your show numbers: ==>\n");
gets(szUserInput);
iChannelNumbers = atoi(szUserInput);
for ( int j=0;j<__min(stuOutParam.nRetCameraCount,iChannelNumbers);++j)
{
DH_MATRIX_CAMERA_INFO stuinfo = stuOutParam.pstuCameras[j];
if( TRUE)// Remote device or not
{
        switch (stuinfo.emChannelType)
        {
        case LOGIC_CHN_REMOTE:
                {
                        printf("This is LOGIC_CHN_REMOTE(remote channel):\n");
break;
}
case LOGIC_CHN_LOCAL:
        {
```

```
            printf("This is LOGIC_CHN_LOCAL(local channel):\n");
            break;
        }
case LOGIC_CHN_COMPOSE:
        {
            printf("This is LOGIC_CHN_COMPOSE(composite channel):\n");
            break;
        }
case LOGIC_CHN_MATRIX:
        {
printf("This is LOGIC_CHN_MATRIX(simulative matrix channel):\n");
break;
}
case LOGIC_CHN_CASCADE:
        {
            printf("This is LOGIC_CHN_CASCADE(cascade channel):\n");
            break;
        }
default:
        {
            printf("This is LOGIC_CHN_UNKNOWN(unknown channel):\n");
        }
}
printf("..........................................\n");
printf("This is the %d remote camera:\n",iNumbers++);
printf("Dev Remote ChannelID = %d,the Local
nUniqueChannel = %d.\n",stuinfo.nChannelID,stuinfo.nUniqueChannel);
printf("Dev Local szDevID = %s,
the Local szName = %s.\n",stuinfo.szDevID,stuinfo.szName);
DH_REMOTE_DEVICE stuRemoteDevice = stuinfo.stuRemoteDevice;
printf("RemoteDev IP = %s,
RemoteDev Port = %d.\n",stuRemoteDevice.szIp,stuRemoteDevice.nPort);
}
}
}
else
{
printf("CLIENT_MatrixGetCameras Failed, Last Error[%x]\n",
CLIENT_GetLastError());
}
```

2.4.4 Importing and Exporting Configuration Information

2.4.4.1 Introduction

To import or export configuration information

2.4.4.2 Interface Overview

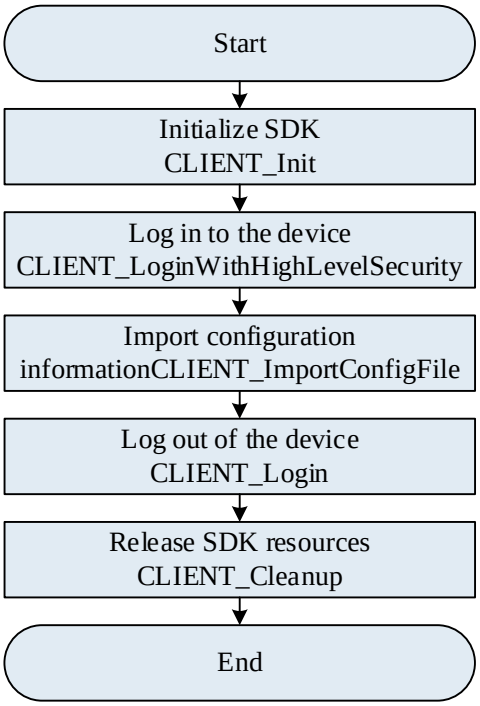
Table 2-24 Import and export configuration information interface

Interface	Description
CLIENT_ImportConfigFileJson	Import configuration information
CLIENT_ExportConfigFileJson	Export configuration information

2.4.4.3 Process Description

2.4.4.3.1 Importing Configuration Information

Figure 2-31 Import configuration information



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_ImportConfigFile to import configuration information.

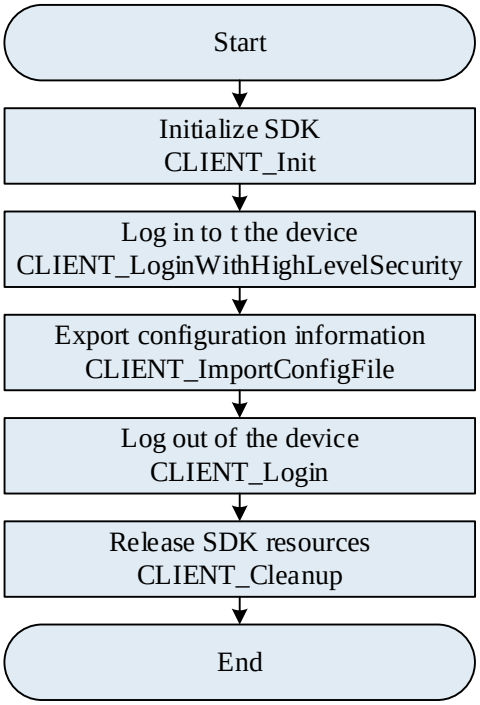
Step 4 Call CLIENT_StopImportCfgFilen to stop importing configuration information.

Step 5 Call CLIENT_Logout to log out of the device.

Step 6 Call CLIENT_Cleanup to release SDK resources.

2.4.4.3.2 Exporting Configuration Information

Figure 2-32 Export Configuration Information



Process Description

Step 1 Call CLIENT_Init to initialize SDK.

Step 2 Call CLIENT_LoginWithHighLevelSecurity to log in to the device.

Step 3 Call CLIENT_ImportConfigFile to export configuration information, including all configurations, local configuration, network configuration and user configuration.

Step 4 Call CLIENT_StopImportCfgFilen to stop importing configuration information.

Step 5 Call CLIENT_Logout to log out of the device.

Step 6 Call CLIENT_Cleanup to release SDK resources.

2.4.4.4 Example Code

2.4.4.4.1 Importing Configuration Information

```
// importConfigJson.cpp : Define the App entering point of the control panel
//
// updownloadConfig.cpp Define the App entering point of the control panel
//
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"
#pragma comment(lib , "dhnetsdk.lib")
static LLONG g_lLoginHandle = 0L;
static char g_szDevIp[32] = "172.23.12.211";
static WORD g_nPort = 37777; // tcp connecting port, conforming to the tcp port configuration of the
login device interface.
```

```
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin123";
static BOOL g_bNetSDKInitFlag = FALSE;
// Download status
double g_downloadStatus = 0;
//*******************************************************************************
// Common callback declaration
// Device disconnection callback function
// We recommend you not call SDK interface in this callback function.
// Set the callback function through CLIENT_Init. When the device is disconnected, SDK will call the function.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);
// Successful reconnection callback function
// We recommend you not call SDK interface in this callback function.
// Set the callback function through CLIENT_Init. When the device is disconnected, SDK will call the function.
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort,LDWORD dwUser);
// Download process callback function
void CALLBACK downloadPosCallback(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);
//*******************************************************************************
void InitTest()
{
    // Initialize SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // Call log interface
    LOG_SET_PRINT_INFO pstLogPrintInfo = {sizeof(LOG_SET_PRINT_INFO)};
    BOOL openLogFlag = CLIENT_LogOpen(&pstLogPrintInfo);
    if (TRUE == openLogFlag)
    {
        // Succeeded
        printf("Success call CLIENT_LogOpen\n");
    }
    else
    {
        // Failed
        printf("Fail call CLIENT_LogOpen\n");
    }
```

```
    // Get SDK version information
    // This operation is optional
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set auto reconnection callback interface. After setting auto reconnection callback function, the
SDK will automatically reconnect the device to network when the device is disconnected.
    // This operation is optional, but we recommend you to configure this.
    CLIENT_SetAutoReconnect(&HaveReConnect, 0);

    // Set login timeout duration and number of attempts
    // This operation is optional.
    int nWaitTime = 5000; // Set the timeout duration of response to login request to 5 seconds
    int nTryTimes = 3; // Set the login attempts to 3.
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // Set more network parameters. The timeout duration and number of attempts of nWaittime and
nConnectTryNum in NET_PARAM are identical with those of CLIENT_SetConnectTime.
    // This operation is optional.
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // timeout duration of login attempts
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam =
{sizeof(NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY)};
    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam =
{sizeof(NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY)};
    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;
    while(0 == g_lLoginHandle)
    {
        // Log in to the device
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);
        if (0 == g_lLoginHandle)
        {
            // Find corresponding explanation from dhnetsdk.h based on the error code. Note the
transfer between the hexadecimal format in printing and the decimal format in header file.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23) // The current SDK does not support this
function. The corresponding error code is 0x80000017, or 0x17 in hexadecimal format.
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" ,
g_szDevIp , g_nPort , CLIENT_GetLastError());
        }
```

```
            else
            {
                printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp, g_nPort);
            }
            // When users first log in to the device, the device needs to initialize some data before functions can be realized. We recommend you wait for a while after loging in. The actual waiting depends on the device.
            Sleep(1000);
            printf("\n");
        }
}
void RunTest()
{
        if (0 == g_lLoginHandle)
        {
            printf("Logining client is failed.\n");
            return ;
        }
        char *pathPtr = "./config.txt";
        FILE *fp = fopen(pathPtr, "rb+");
        if (NULL != fp)
        {
            printf("Success open file\n");
            /*
             * Read files
             */
            // Get file length
            fseek(fp, 0, SEEK_END);
            int fileLength = ftell(fp);
            rewind(fp);
            // Read the file and then close.
            char *configBuffer = new char[1024 * 1024];
            memset(configBuffer, 0, 1024 * 1024);
            fread(configBuffer, sizeof(char), fileLength, fp);
            printf("Success read file\n");
            fclose(fp);
            /*
             * Import the configuration information
             */
            BOOL importStatus = CLIENT_ImportConfigFileJson(g_lLoginHandle, configBuffer, fileLength);
            if (TRUE == importStatus)
            {
                printf("Success import config.\n");
            }
            else
            {
```

```
                printf("Fail import config. Last error[%x]\n", CLIENT_GetLastError());
            }

        }
        else
        {
            printf("Fail open file. Fail write json\n");
            return ;
        }
        // char *pOutBuffer = new char[1024 * 1024 * 1024];
        // memset(pOutBuffer, 0, 1024 * 1024 * 1024);
        // int maxlen = 1024 * 1024 * 1024;
        // printf("maxlen = %d\n", maxlen);
        // // Actual length
        // int nRetlen = 0;
        // BOOL exportStatus = CLIENT_ExportConfigFileJson(g_lLoginHandle, pOutBuffer, maxlen,
&nRetlen);
        // if (TRUE == exportStatus)
        // {
        //     // Import succeeded.
        //     printf("json:\n");
        //     printf("%s\n", pOutBuffer);

        // }
        // else
        // {
        //     // Failed to import
        //     printf("Fail to CLIENT_ExportConfigFileJson. Last error[%x]\n", CLIENT_GetLastError());
        // }
}
void EndTest()
{
        printf("input any key to quit!\n");
        getchar();
        // Log out of the device
        if (0 != g_lLoginHandle)
        {
            if (FALSE == CLIENT_Logout(g_lLoginHandle))
            {
                printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
            }
            else
            {
                g_lLoginHandle = 0;
            }
        }
        BOOL closeLogFlag = CLIENT_LogClose();
```

```
    if (0 == closeLogFlag)
    {
        // Succeeded
        printf("Success call CLIENT_LogClose\n");
    }
    else
    {
        // Failed
        printf("Fail call CLIENT_LogClose\n");
    }
    // Clean initialization resources
    if (TRUE == g_bNetSDKInitFlag)
    {
        CLIENT_Cleanup();
        g_bNetSDKInitFlag = FALSE;
    }
    return;
}
int main()
{
    // Initialize and log in to the device
    InitTest();
    // Realize corresponding functions: import configurations
    RunTest();
    // Log out of the device and clean the initialization resources
    EndTest();
    return 0;
}
//*******************************************************************************
// Common callback declarations
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser)
{
    printf("Call DisConnectFunc\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
    {
        printf("pchDVRIP[%s]\n", pchDVRIP);
    }
    printf("nDVRPort[%d]\n", nDVRPort);
    printf("dwUser[%p]\n", dwUser);
    printf("\n");
}
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    printf("Call HaveReConnect\n");
    printf("lLoginID[0x%x]", lLoginID);
    if (NULL != pchDVRIP)
```

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

2.4.4.4.2 Exporting Configuration information

```
// updownloadConfig.cpp : Define the entering point of control panel application
//
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include "dhnetsdk.h"
#pragma comment(lib , "dhnetsdk.lib")
static LLONG g_lLoginHandle = 0L;
static char g_szDevIp[32] = "172.23.12.211";
static WORD g_nPort = 37777; // tcp connecting port, conforming to the tcp port configuration of the
login device interface.
static char g_szUserName[64] = "admin";
static char g_szPasswd[64] = "admin123";
static BOOL g_bNetSDKInitFlag = FALSE;
// Download status
double g_downloadStatus = 0;
//*******************************************************************************
// Common callback declaration
// Device disconnection callback function
// We recommend you not call SDK interface in this callback function.
// Set the callback function through CLIENT_Init. When the device is disconnected, SDK will call the
function.
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser);
// Successful reconnection callback function
// We recommend you not call SDK interface in this callback function.
// Set the callback function through CLIENT_Init. When the device is disconnected, SDK will call the
function.
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort,LDWORD dwUser);
// Download process callback function
void CALLBACK downloadPosCallback(LLONG lPlayHandle, DWORD dwTotalSize, DWORD
dwDownLoadSize, LDWORD dwUser);
//*******************************************************************************
void InitTest()
{
    // Initialize SDK
    g_bNetSDKInitFlag = CLIENT_Init(DisConnectFunc, 0);
    if (FALSE == g_bNetSDKInitFlag)
    {
```

```
        printf("Initialize client SDK fail; \n");
        return;
    }
    else
    {
        printf("Initialize client SDK done; \n");
    }
    // Call log interface
    LOG_SET_PRINT_INFO pstLogPrintInfo = {sizeof(LOG_SET_PRINT_INFO)};
    BOOL openLogFlag = CLIENT_LogOpen(&pstLogPrintInfo);
    if (TRUE == openLogFlag)
    {
        // Succeeded
        printf("Success call CLIENT_LogOpen\n");
    }
    else
    {
        // Failed
        printf("Fail call CLIENT_LogOpen\n");
    }
    // Get SDK version information
    // This operation is optional.
    DWORD dwNetSdkVersion = CLIENT_GetSDKVersion();
    printf("NetSDK version is [%d]\n", dwNetSdkVersion);

    // Set the reconnection callback interface. After the reconnection callback is successfully set, when the device is disconnected, the SDK will automatically reconnect it.
    // This operation is optional, but we recommend you set it.

    // Set login timeout duration and number of attempts
    // This operation is optional
    int nWaitTime = 5000; // Set the timeout duration of response to login request to 5 seconds
    int nTryTimes = 3; // Set the login attempts to 3
    CLIENT_SetConnectTime(nWaitTime, nTryTimes);

    // Set more network parameters. The timeout duration and number of attempts of nWaittime and nConnectTryNum in NET_PARAM are identical with those of CLIENT_SetConnectTime.
    // This operation is optional
    NET_PARAM stuNetParm = {0};
    stuNetParm.nConnectTime = 3000; // timeout duration of login attempts
    CLIENT_SetNetworkParam(&stuNetParm);

    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stInparam = {sizeof(NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY)};
    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stOutparam = {sizeof(NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY)};
    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;
    while(0 == g_lLoginHandle)
    {
        // Log in to the device
        g_lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stInparam, &stOutparam);
        if (0 == g_lLoginHandle)
        {
            // Find corresponding explanation from dhnetsdk.h based on the error code. Note the
transfer between the hexadecimal format in printing and the decimal format in header file.
            // For example:
            // #define NET_NOT_SUPPORTED_EC(23) // The current SDK does not support this
function. The corresponding error code is 0x80000017, or 0x17 in hexadecimal format.
    printf("CLIENT_LoginWithHighLevelSecurity %s[%d]Failed!Last Error[%x]\n" , g_szDevIp ,
g_nPort , CLIENT_GetLastError());
        }
        else
        {
            printf("CLIENT_LoginWithHighLevelSecurity %s[%d] Success\n" , g_szDevIp, g_nPort);
        }
        // When users first log in to the device, the device needs to initialize some data before
functions can be realized. We recommend you wait for a while after logging in. The actual waiting time
depends on the device.
        Sleep(1000);
        printf("\n");
    }
}
void RunTest()
{
    if (0 == g_lLoginHandle)
    {
        printf("Logining client is failed.\n");
        return ;
    }
    char *pOutBuffer = new char[1024 * 1024 * 1024];
    memset(pOutBuffer, 0, 1024 * 1024 * 1024);
    int maxlen = 1024 * 1024 * 1024;
    printf("maxlen = %d\n", maxlen);
    // Actual length
    int nRetlen = 0;
    BOOL exportStatus = CLIENT_ExportConfigFileJson(g_lLoginHandle, pOutBuffer, maxlen,
&nRetlen);
    if (TRUE == exportStatus)
    {
```

```
            // Export succeeded
            printf("json:\n");
            printf("%s\n", pOutBuffer);
            char *pathPtr = "./config.txt";
            FILE *fp = fopen(pathPtr, "wb+");
            if (NULL != fp)
            {
                printf("Success open file\n");
                fwrite(pOutBuffer, sizeof(char), nRetlen, fp);
                printf("Success write file\n");
                fclose(fp);
            }
            else
            {
                printf("Fail open file. Fail write json\n");
            }
        }
        else
        {
            // Export failed
            printf("Fail to CLIENT_ExportConfigFileJson. Last error[%x]\n", CLIENT_GetLastError());
        }
}
void EndTest()
{
        printf("input any key to quit!\n");
        getchar();
        // Log out of the device
        if (0 != g_lLoginHandle)
        {
            if (FALSE == CLIENT_Logout(g_lLoginHandle))
            {
                printf("CLIENT_Logout Failed!Last Error[%x]\n", CLIENT_GetLastError());
            }
            else
            {
                g_lLoginHandle = 0;
            }
        }
        BOOL closeLogFlag = CLIENT_LogClose();
        if (0 == closeLogFlag)
        {
            // Succeeded
            printf("Success call CLIENT_LogClose\n");
        }
        else
        {
```

```
            // Failed
            printf("Fail call CLIENT_LogClose\n");
        }
        // Clean initialization resources
        if (TRUE == g_bNetSDKInitFlag)
        {
            CLIENT_Cleanup();
            g_bNetSDKInitFlag = FALSE;
        }
        return;
}
int main()
{
        // Initialize and log in to the device
        InitTest();
        // Realize corresponding functions: import configurations
        RunTest();
        // Log out of the device and clean the initialization resources
        EndTest();
        return 0;
}
//*******************************************************************************
// Common callback definitio
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser)
{
        printf("Call DisConnectFunc\n");
        printf("lLoginID[0x%x]", lLoginID);
        if (NULL != pchDVRIP)
        {
            printf("pchDVRIP[%s]\n", pchDVRIP);
        }
        printf("nDVRPort[%d]\n", nDVRPort);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
}
void CALLBACK HaveReConnect(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
        printf("Call HaveReConnect\n");
        printf("lLoginID[0x%x]", lLoginID);
        if (NULL != pchDVRIP)
        {
            printf("pchDVRIP[%s]\n", pchDVRIP);
        }
        printf("nDVRPort[%d]\n", nDVRPort);
        printf("dwUser[%p]\n", dwUser);
        printf("\n");
}
```

3 Interface Definition

3.1 General Interfaces

3.1.1 SDK Initialization

3.1.1.1 SDK CLIENT_Init

Table 3-1 Initialize SDK

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

3.1.1.2 CLIENT_Cleanup

Table 3-2 Clean up SDK

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

3.1.1.3 CLIENT_SetAutoReconnect

Table 3-3 Set reconnection callback

Item	Description
Description	Set auto reconnection callback.

Item	Description	
Function	void CLIENT_SetAutoReconnect(fHaveReConnect cbAutoConnect, LDWORD dwUser);	
Parameter	[in]cbAutoConnect	Reconnection callback.
	[in]dwUser	User parameter of disconnection callback.
Return value	None.	
Note	Set the reconnection callback interface. If the callback is set as NULL, it will not connect automatically.	

3.1.1.4 CLIENT_SetNetworkParam

Table 3-4 Set network parameter

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

3.1.2 Device Initialization

3.1.2.1 CLIENT_StartSearchDevicesEx

Table 3-5 Search for device

Item	Description	
Description	Search the device.	
Function	LLONG CLIENT_StartSearchDevicesEx (NET_IN_STARTSERACH_DEVICE* pInBuf, NET_OUT_STARTSERACH_DEVICE* pOutBuf);	
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.	

3.1.2.2 CLIENT_InitDevAccount

Table 3-6 Initialize device

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

3.1.2.3 CLIENT_GetDescriptionForResetPwd

Table 3-7 Get information for password reset

Item	Description	
Description	Get information for password reset.	
Function	BOOL CLIENT_GetDescriptionForResetPwd(const NET_IN_DESCRIPTION_FOR_RESET_PWD *pDescriptionIn, NET_OUT_DESCRIPTION_FOR_RESET_PWD *pDescriptionOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pDescriptionIn	Corresponds to structure of NET_IN_DESCRIPTION_FOR_RESET_PWD.
	[out]pDescriptionOut	Corresponds to structure of NET_OUT_DESCRIPTION_FOR_RESET_PWD.
	[in]dwWaitTime	Timeout.
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.

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

3.1.2.4 CLIENT_CheckAuthCode

Table 3-8 Check the validity of security code

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

3.1.2.5 CLIENT_ResetPwd

Table 3-9 Reset the password

Item	Description	
Description	Reset the password.	
Function	BOOL CLIENT_ResetPwd(const NET_IN_RESET_PWD *pResetPwdIn, NET_OUT_RESET_PWD *pResetPwdOut, DWORD dwWaitTime, char *szLocalIp);	
Parameter	[in]pResetPwdIn	Corresponds to structure of NET_IN_RESET_PWD.
	[out]pResetPwdOut	Corresponds to structure of NET_OUT_RESET_PWD.
	[in]dwWaitTime	Timeout.

Item	Description	
	[in]szLocalIp	● In case of single network card, the last parameter is not required to be filled. ● In case of multiple network card, enter the IP of the host PC for the last parameter.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.1.2.6 CLIENT_GetPwdSpecification

Table 3-10 Get password rules

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

3.1.2.7 CLIENT_StopSearchDevices

Table 3-11 Stop searching device

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

3.1.3 Device Login

3.1.3.1 CLIENT_LoginWithHighLevelSecurity

Table 3-12 Log in with high level security

Item	Description	
Description	Login the device with high level security.	
Function	LLONG CLIENT_LoginWithHighLevelSecurity (NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY* pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY* pstOutParam);	
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. ● Failure: 0.	
Note	Login 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 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.

3.1.3.2 CLIENT_Logout

Table 3-14 Log out

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

3.1.4 Real-time Monitoring

3.1.4.1 CLIENT_RealPlayEx

Table 3-15 Start the real-time monitoring

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

Table 3-16 Live view type and meaning

Preview type	Meaning
DH_RType_Realplay	Real-time preview.
DH_RType_Multiplay	Multi-picture preview.
DH_RType_Realplay_0	Real-time monitoring—main stream, equivalent to DH_RType_Realplay.
DH_RType_Realplay_1	Real-time monitoring—sub stream 1.

Preview type	Meaning
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.

3.1.4.2 CLIENT_StopRealPlayEx

Table 3-17 Stop the real-time monitoring

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

3.1.4.3 CLIENT_SaveRealData

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

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

3.1.4.4 CLIENT_StopSaveRealData

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

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

3.1.4.5 CLIENT_SetRealDataCallBackEx2

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

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

Table 3-21 dwFlag type and parameter

dwFlag	Description
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.

3.2 Traffic Junction

3.2.1 Download of Medial File

3.2.1.1 CLIENT_FindFileEx

Table 3-22 Query the media file per query condition

Item	Description	
Description	Query the media file per query condition.	
Function	LLONG CLIENT_FindFileEx(LLONG lLoginID, EM_FILE_QUERY_TYPE emType, void* pQueryCondition, void* reserved, int waittime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] emType	Query information type of media file, see Table 3-23.
	[in] pQueryCondition	Query condition.
	[in]reserved	Reserved parameter, not valid.
	[in] waittime	Timeout.
Return value	● Success: Not 0. ● Failure: 0.	
Note	When querying the media file, use DH_FILE_QUERY_TRAFFICCAR_EX for parameter emType. The parameter pQueryCondition corresponds to structure MEDIA_QUERY_TRAFFICCAR_PARAM_EX.	

Table 3-23 emType and meaning

emType enumeration definition	Meaning	Corresponding structure of pQueryCondition
DH_FILE_QUERY_TRAFFICCAR	Traffic vehicles information	MEDIA_QUERY_TRAFFICCAR_PARAM
DH_FILE_QUERY_FACE	Face information	MEDIAFILE_FACERECOGNITION_PARAM
DH_FILE_QUERY_FILE	File information	NET_IN_MEDIA_QUERY_FILE
DH_FILE_QUERY_TRAFFICCAR_EX	Traffic vehicles information (extension)	MEDIA_QUERY_TRAFFICCAR_PARAM_EX
DH_FILE_QUERY_FACE_DETECTION	Face detection information	MEDIAFILE_FACE_DETECTION_PARAM

3.2.1.2 CLIENT_GetTotalFileCount

Table 3-24 Get the total number of queried files

Item	Description	
Description	Get the total number of queried files.	
Function	BOOL CLIENT_GetTotalFileCount(LLONG lFindHandle, int* pTotalCount, void* reserved, int waittime);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindFileEx.
	[out] pTotalCount	The total number of queried information.
	[in]reserved	Reserved parameter, not valid.
	[in] waittime	Timeout.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.1.3 CLIENT_FindNextFileEx

Table 3-25 Query the media file

Item	Description	
Description	Query the media file.	
Function	int CLIENT_FindNextFileEx(LLONG lFindHandle, int nFilecount, void* pMediaFileInfo, int maxlen, void* reserved, int waittime);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindFileEx.
	[in] nFilecount	Query number.
	[out] pMediaFileInfo	Output cache of media file information.
	[in] maxlen	Value of maximum cache area.
	[in]reserved	Reserved parameter, not valid.
	[in] waittime	Timeout.
Return value	Returns the total number of queried media files. The query is called finished if the return value is smaller than the query number.	
Note	None.	

3.2.1.4 CLIENT_FindCloseEx

Table 3-26 Stop querying the media file

Item	Description
Description	Stop querying the media file.
Function	BOOL CLIENT_FindCloseEx(

Item	Description	
	LLONG lFindHandle);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindFileEx.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.2.1.5 CLIENT_DownloadMediaFile

Table 3-27 Download the media file

Item	Description	
Description	Download the media file.	
Function	LLONG CLIENT_DownloadMediaFile(LLONG lLoginID, EM_FILE_QUERY_TYPE emType, void* lpMediaFileInfo, char* sSavedFileName, fDownLoadPosCallBack cbDownLoadPos, LDWORD dwUserData, void* reserved);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] emType	Media file type, seeTable 3-23. .
	[in] lpMediaFileInfo	Media file information.
	[in] sSavedFileName	Save path.
	[in] cbDownLoadPos	Callback of download progress: fDownLoadPosCallBack.
	[in] dwUserData	Corresponding user number of callback.
	[in]reserved	Reserved parameter, not valid.
Return value	● Success: Not 0. ● Failure: 0.	
Note	When downloading vehicles pictures, the parameter emType only supports DH_FILE_QUERY_TRAFFICCAR.	

3.2.1.6 CLIENT_StopDownloadMediaFile

Table 3-28 Stop downloading the media file

Item	Description	
Description	Stop downloading the media file.	
Function	BOOL CLIENT_StopDownloadMediaFile(LLONG lFileHandle);	
Parameter	[in] lFindHandle	Return value of CLIENT_DownloadMediaFile.
Return value	● Success: TRUE. ● Failure: FALSE.	

Item	Description
Note	None.

3.2.2 Manual Capture

3.2.2.1 CLIENT_RealLoadPictureEx

Table 3-29 Subscribe intelligent event

Item	Description	
Description	Subscribe intelligent event.	
Function	LLONG CLIENT_RealLoadPictureEx(LLONG lLoginID, int nChannelID, DWORD dwAlarmType, BOOL bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] nChannelID	Device channel number.
	[in] dwAlarmType	Type of intelligent traffic event, see Table 3-30 and Table 3-34.
	[in] bNeedPicFile	Whether picture is needed.
	[in] cbAnalyzerData	Callback of intelligent event: fAnalyzerDataCallBack.
	[in] dwUser	Corresponding user data of callback.
	[in]Reserved	Reserved parameter, not valid.
Return value	● Success: Not 0. ● Failure: 0.	
Note	● Call this interface in advance for manual capturing to receive the captured pictures. ● Call this interface in advance for event upload to receive the event information and pictures.	

Table 3-30 dwAlarmType and meaning

dwAlarmType macro definition	Value of macro definition	Meaning	Call the corresponding structure of pAlarmInfo
EVENT_IVS_TRAFFIC_MANUALSNAP	0x00000118	Intelligent capturing event	DEV_EVENT_TRAFFIC_MANUALSNAP_INFO

3.2.2.2 CLIENT_ControlDeviceEx

Table 3-31 Control device.

Item	Description	
Description	Control device.	
Function	BOOL CLIENT_ControlDeviceEx(LLONG lLoginID, CtrlType emType, void* pInBuf, void* pOutBuf, int nWaitTime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] emType	Control type, see Table 3-30 and Table 3-32.
	[in] pInBuf	Control input cache, see Table 3-30 and Table 3-32.
	[in] pOutBuf	Controls output cache.
	[in] nWaitTime	Timeout.
Return value	● Success: TRUE ● Failure: FALSE	
Note	Manually trigger the capturing and receive pictures through subscribing the callback of interface.	

The following table shows information about parameter emType:

Table 3-32 emType and meaning (2)

emType enumeration definition	Meaning	The corresponding structure of pInBuf
DH_MANUAL_SNAP	Manual capture	MANUAL_SNAP_PARAMETER

3.2.2.3 CLIENT_StopLoadPic

Table 3-33 Cancel subscription of intelligent event

Item	Description	
Description	Cancel subscription of intelligent event.	
Function	BOOL CLIENT_StopLoadPic(LLONG lAnalyzerHandle);	
Parameter	[in] lAnalyzerHandle	Return value of CLIENT_RealLoadPictureEx.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	After calling this interface, you will not receive the pictures even if continue to trigger manual capturing.	

3.2.3 Upload of Intelligent Traffic Event

3.2.3.1 CLIENT_RealLoadPictureEx

For the interface function, see "3.2.2.1 CLIENT_RealLoadPictureEx".

Table 3-34 Type of intelligent traffic event

dwAlarmType macro definition	Value of macro definition	Meaning	Corresponding structure of pAlarmInfo
EVENT_IVS_ALL	0x00000001	All events	No
EVENT_IVS_TRAFFICCONTROL	0x00000015	Event of traffic control	DEV_EVENT_TRAFFICCONTROL_INFO
EVENT_IVS_TRAFFICACCIDENT	0x00000016	Event of traffic accident	DEV_EVENT_TRAFFICACCIDENT_INFO
EVENT_IVS_TRAFFICJUNCTION	0x00000017	Event of traffic conjunction	DEV_EVENT_TRAFFICJUNCTION_INFO
EVENT_IVS_TRAFFICGATE	0x00000018	Event of traffic gate	DEV_EVENT_TRAFFICGATE_INFO
EVENT_IVS_TRAFFIC_RUNREDLIGHT	0x00000100	Event of running the red light	DEV_EVENT_TRAFFIC_RUNREDLIGHT_INFO
EVENT_IVS_TRAFFIC_OVERLINE	0x00000101	Event of running over line	DEV_EVENT_TRAFFIC_OVERLINE_INFO
EVENT_IVS_TRAFFIC_RETROGRADE	0x00000102	Event of retrograde	DEV_EVENT_TRAFFIC_RETROGRADE_INFO
EVENT_IVS_TRAFFIC_TURNLEFT	0x00000103	Event of violating regulations by left turn	DEV_EVENT_TRAFFIC_TURNLEFT_INFO
EVENT_IVS_TRAFFIC_TURNRIGHT	0x00000104	Event of violating regulations by right turn	DEV_EVENT_TRAFFIC_TURNRIGHT_INFO
EVENT_IVS_TRAFFIC_UTURN	0x00000105	Event of violating regulations by turning around	DEV_EVENT_TRAFFIC_UTURN_INFO
EVENT_IVS_TRAFFIC_OVERSPEED	0x00000106	Event of running over speed	DEV_EVENT_TRAFFIC_OVERSPEED_INFO
EVENT_IVS_TRAFFIC_UNDERSPEED	0x00000107	Event of running under speed	DEV_EVENT_TRAFFIC_UNDERSPEED_INFO
EVENT_IVS_TRAFFIC_PARKING	0x00000108	Event of illegal parking	DEV_EVENT_TRAFFIC_PARKING_INFO
EVENT_IVS_TRAFFIC_WRONGROUTE	0x00000109	Event of running along the wrong route	DEV_EVENT_TRAFFIC_WRONGROUTE_INFO
EVENT_IVS_TRAFFIC_CROSSLANE	0x0000010A	Event of violating regulations by crossing lanes	DEV_EVENT_TRAFFIC_CROSSLANE_INFO
EVENT_IVS_TRAFFIC_OVERYELLOWLINE	0x0000010B	Event of running on the yellow line	DEV_EVENT_TRAFFIC_OVERYELLOWLINE_INFO
EVENT_IVS_TRAFFIC_DRIVINGONSHOULDER	0x0000010C	Event of running on the road shoulder	DEV_EVENT_TRAFFIC_DRIVINGONSHOULDER_INFO

dwAlarmType macro definition	Value of macro definition	Meaning	Corresponding structure of pAlarmInfo
EVENT_IVS_TRAFFIC_YELLOWPLATEINLANE	0x0000010E	Event of yellow plate occupying the lanes	DEV_EVENT_TRAFFIC_YELLOWPLATEINLANE_INFO
EVENT_IVS_TRAFFIC_PEDESTRAINPRIORITY	0x0000010F	Event of pedestrian priority at zebra crossing	DEV_EVENT_TRAFFIC_PEDESTRAINPRIORITY_INFO
EVENT_IVS_TRAFFIC_PARKINGONYELLOWBOX	0x0000012A	Event of capturing the cars parking at the yellow box	DEV_EVENT_TRAFFIC_PARKINGONYELLOWBOX_INFO
EVENT_IVS_TRAFFIC_PARKINGSPACEPARKING	0x0000012B	Event of parking space taken by cars	DEV_EVENT_TRAFFIC_PARKINGSPACEPARKING_INFO
EVENT_IVS_TRAFFIC_PARKINGSPACENOPARKING	0x0000012C	Event of parking space taken by no cars	DEV_EVENT_TRAFFIC_PARKINGSPACENOPARKING_INFO
EVENT_IVS_TRAFFIC_PEDESTRAIN	0x0000012D	Event about pedestrian	DEV_EVENT_TRAFFIC_PEDESTRAIN_INFO
EVENT_IVS_TRAFFIC_THROW	0x0000012E	Event of throwing objects	DEV_EVENT_TRAFFIC_THROW_INFO
EVENT_IVS_TRAFFIC_IDLE	0x0000012F	Idle event	DEV_EVENT_TRAFFIC_IDLE_INFO
EVENT_IVS_TRAFFIC_RESTRICTED_PLATE	0X00000136	Event of restricted plate	DEV_EVENT_TRAFFIC_RESTRICTED_PLATE
EVENT_IVS_TRAFFIC_OVERSTOPLINE	0X00000137	Event of pressing on the stop line	DEV_EVENT_TRAFFIC_OVERSTOPLINE
EVENT_IVS_TRAFFIC_WITHOUT_SAFEBELT	0x00000138	Event of safety belt unfastened	DEV_EVENT_TRAFFIC_WITHOUT_SAFEBELT
EVENT_IVS_TRAFFIC_DRIVER_SMOKING	0x00000139	Event of driver smoking	DEV_EVENT_TRAFFIC_DRIVER_SMOKING
EVENT_IVS_TRAFFIC_DRIVER_CALLING	0x0000013A	Event of driver calling	DEV_EVENT_TRAFFIC_DRIVER_CALLING
EVENT_IVS_TRAFFIC_PEDESTRAINRUNREDLIGHT	0x0000013B	Event of pedestrian running the red light	DEV_EVENT_TRAFFIC_PEDESTRAINRUNREDLIGHT_INFO
EVENT_IVS_TRAFFIC_PASSNOTINORDER	0x0000013C	Event of passing without order	DEV_EVENT_TRAFFIC_PASSNOTINORDER_INFO

3.2.3.2 CLIENT_StopLoadPic

For the interface function, see "3.2.2.3 CLIENT_StopLoadPic."

3.2.4 Vehicle Flow Statistics

3.2.4.1 CLIENT_StartTrafficFluxStat

Table 3-35 Subscribe the statistics of vehicle flow

Item	Description	
Description	Subscribe the statistics of vehicle flow.	
Function	LLONG CLIENT_StartTrafficFluxStat(LLONG lLoginID, NET_IN_TRAFFICFLUXSTAT* pstInParam, NET_OUT_TRAFFICFLUXSTAT* pstOutParam);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] pstInParam	Input parameter. Vehicle flow statistics callback: fFluxStatDataCallBack.
	[out] pstOutParam	Output parameter.
Return value	● Success: Not 0. ● Failure: 0.	
Note	None.	

3.2.4.2 CLIENT_StopTrafficFluxStat

Table 3-36 Stop subscribing the statistics of vehicle flow

Item	Description	
Description	Stop subscribing the statistics of vehicle flow	
Function	BOOL CLIENT_StopTrafficFluxStat(LLONG lFluxStatHandle);	
Parameter	[in] lFluxStatHandle	Return value of CLIENT_StartTrafficFluxStat
Return value	● Success: TRUE ● Failure: FALSE	
Note	None	

3.2.5 Intelligent Traffic

3.2.5.1 CLIENT_FindRecord

Table 3-37 Start searching for traffic data records (Set searching conditions)

Item	Description	
Description	Start searching for data (Set searching conditions)	
Function	BOOL CLIENT_FindRecord(LLONG lLoginID, NET_IN_FIND_RECORD_PARAM* pInParam, NET_OUT_FIND_RECORD_PARAM* pOutParam, int waittime=1000);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] pInParam	Input search conditions
	[out] pOutParam	Output search results
Return Value	Return TRUE for success and FALSE for failure.	
Note	Record type: emType= NET_RECORD_TRAFFICFLOW_STATE	

3.2.5.2 CLIENT_QueryRecordCount

Table 3-38 The total number of searches for traffic data records

Item	Description	
Description	The total number of searches	
Function	BOOL CLIENT_QueryRecordCount(NET_IN_QUEYT_RECORD_COUNT_PARAM* pInParam, NET_OUT_QUEYT_RECORD_COUNT_PARAM* pOutParam, int waittime=1000);	
Parameter	[in] pInParam	Search for input parameter
	[out] pOutParam	Search for output parameter
	[in] waittime	Timeout duration
Return value	Return TRUE for success and FALSE for failure.	
Note	None	

3.2.5.3 CLIENT_FindNextRecord

Table 3-39 Search for specified number of traffic data records

Item	Description	
Description	Search for specified number of data	
Function	int CLIENT_FindNextRecord(NET_IN_FIND_NEXT_RECORD_PARAM* pInParam, NET_OUT_FIND_NEXT_RECORD_PARAM* pOutParam, int waittime=1000);	
Parameter	[in] pstInParam	Search for input parameter
		Search for output parameter
	[in] waittime	Timeout duration
Return Value	The number of searches.	
Note	None	

3.2.5.4 CLIENT_FindRecordClose

Table 3-40 Stop searching for vehicle flow

Item	Description	
Description	Stop searching for vehicle flow	
Function	BOOL CLIENT_FindRecordClose(LLONG lFindHandle);	
Parameter	[in] lFindHandle	Search handle
Return value	Return TRUE for success and FALSE for failure	
Note	None	

3.2.5.5 CLIENT_OperateTrafficList

Table 3-41 Adding, deleting and modifying Allowlist/Blocklist

Item	Description	
Description	Adding, deleting and modifying Allowlist/Blocklist	
Function	BOOL CLIENT_OperateTrafficList(LLONG lLoginID , NET_IN_OPERATE_TRAFFIC_LIST_RECORD* pstInParam , NET_OUT_OPERATE_TRAFFIC_LIST_RECORD *pstOutParam , int waittime)	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] pstInParam	Allowlist/Blocklist operation input parameter
	[out] pstOutParam	Allowlist/Blocklist operation output parameter
	[in] waittime	Timeout duration
Return value	Return TRUE for success and FALSE for failure	
Note	NET_TRAFFIC_LIST_INSERT// add record NET_TRAFFIC_LIST_UPDATE// edit record NET_TRAFFIC_LIST_REMOVE// delete record	

3.2.5.6 CLIENT_DownLoadMultiFile

Table 3-42 Download files in batches

Item	Description	
Description	Download files in batches	
Function	BOOL CLIENT_DownLoadMultiFile(LLONG lLoginID, NET_IN_DOWNLOAD_MULTI_FILE *pstInParam, NET_OUT_DOWNLOAD_MULTI_FILE *pstOutParam, int waittime=1000);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] pstInParam	Input parameter for downloading files in batches
	[out] pstOutParam	Output parameter for downloading files in batches

Item	Description	
	[in] waittime	Timeout duration
Return value	Return TRUE for success and FALSE for failure.	
Note	None	

3.2.5.7 CLIENT_StopLoadMultiFile

Table 3-43 Stop downloading files in batches

Item	Description	
Description	Stop downloading files in batches	
Function	BOOL CLIENT_StopLoadMultiFile(LLONG lDownLoadHandle);	
Parameter	[in] lDownLoadHandle	Batch download handle
Return value	Return TRUE for success and FALSE for failure.	
Note	None	

3.2.6 Searching for and Downloading Intelligent Event Videos or Images

3.2.6.1 CLIENT_FindFileEx

Table 3-44 Search for files based on the search conditions

Item	Description	
Description	Search for files based on the search conditions	
Function	LLONG CLIENT_FindFileEx(LLONG lLoginID, EM_FILE_QUERY_TYPE emType, void* pQueryCondition, void* reserved, int waittime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] emType	File type
	[in] pQueryCondition	Search conditions
	[in] reserved	Reserved parameter
	[in] waittime	Waiting time
Return value	If succeeded, return the search handle of LLONG type; If failed, return 0.	
Note	None	

3.2.6.2 CLIENT_GetTotalFileCount

Table 3-45 Get the number of files searched

Item	Description	
Description	Get the number of files searched	
Function	BOOL CLIENT_GetTotalFileCount(LLONG lFindHandle, int* pTotalCount, void * reserved, int waittime);	
Parameter	[in] lFindHandle	Search handle
	[out] pTotalCount	The number of files searched
	[in] reserved	Reserved parameter
	[in] waittime	Timeout duration
Return value	Return TRUE for success and FALSE for failure.	
Note	None	

3.2.6.3 CLIENT_FindNextFileEx

Table 3-46 Search for files

Item	Description	
Description	Search for files	
Function	int CLIENT_FindNextFileEx(LLONG lFindHandle, int nFilecount, void* pMediaFileInfo, int maxlen, void* reserved, int waittime);	
Parameter	[in] lFindHandle	Search handle
	[in] nFilecount	The number of files to be searched.
	[out] pMediaFileInfo	File buffering area
	[in] maxlen	Search for the buffering size of file groups
	[in] reserved	Reserved parameter
	[in] waittime	Timeout duration
Return value	If succeeded, return the number of files searched; If failed, return -1; If return 0, the search ends.	
Note	None	

3.2.6.4 CLIENT_FindCloseEx

Table 3-47 Stop searching for files

Item	Description
Description	Stop searching for files

Item	Description	
Function	BOOL CLIENT_FindCloseEx(LLONG lFindHandle);	
Parameter	[in] lFindHandle	Search handle
Return value	Return TRUE for success and FALSE for failure.	
Note	None	

3.2.6.5 CLIENT_PlayBackByTimeEx2

Table 3-48 Start video playback

Item	Description	
Description	Start video playback	
Function	BFD CLIENT_PlayBackByTimeEx2 LLONG lLoginID, Int nChannelID, NET_IN_PLAY_BACK_BY_TIME_INFO* pstNetIn, NET_OUT_PLAY_BACK_BY_TIME_INFO* pstNetOut);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] nChannelID	Channel No.
	[in] pstNetIn	Playback input parameter
	[out] pstNetOut	Playback output parameter
Return value	If succeeded, return the playback handle of LLONG type; If failed, return 0.	
Note	None	

3.2.6.6 CLIENT_StopPlayBack

Table 3-49 Stop video playback

Item	Description	
Description	Stop video playback	
Function	BOOL CLIENT_StopPlayBack(LLONG lPlayHandle);	
Parameter	[in] lPlayHandle	Playback handle
Return value	Return TRUE for success and FALSE for failure.	
Note	None	

3.2.6.7 CLIENT_DownloadByTimeEx

Table 3-50 Start downloading videos

Item	Description
Description	Start downloading videos

Item	Description	
Function	LLONG CLIENT_DownloadByTimeEx(LLONG lLoginID, int nChannelId, int nRecordFileType, LPNET_TIME tmStart, LPNET_TIME tmEnd, char* sSavedFileName, fTimeDownLoadPosCallBack cbTimeDownLoadPos, LDWORD dwUserData, fDataCallBack fDownLoadDataCallBack, LDWORD dwDataUser, void* pReserved = NULL)	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] nChannelId	Channel No.
	[in] nRecordFileType	Record file type
	[in] tmStart	Start time of video downlaod
	[in] tmEnd	End time of video download
	[in] sSavedFileName	Designate storage path for videos. If no path is designated, the video is not stored.
	[int]cbTimeDownLoadPos	Callback function of video download prgress
	[in] dwUserData	User data of Callback function of video download prgress
	[in]fDownLoadDataCallBack	Callback function of video download data
	[in] dwDataUser	User data of Callback function of video download data
	[in] pReserved	Reserved parameter
Return value	Return LLONG download handle for success and 0 for failure.	
Note	None	

3.2.6.8 CLIENT_StopDownload

Table 3-51 Stop downloading videos

Item	Description	
Description	Stop downloading videos	
Function	BOOL CLIENT_StopDownload(LLONG lFileHandle);	
Parameter	[in] lFileHandle	Download handle
Return value	Return TRUE for success and FALSE for failure.	
Note	None	

3.2.6.9 CLIENT_DownloadRemoteFile

Table 3-52 Download files through file names

Item	Description	
Description	Download files through file names	
Function	BOOL CLIENT_DownloadRemoteFile(LLONG lLoginID, const DH_IN_DOWNLOAD_REMOTE_FILE* pInParam, DH_OUT_DOWNLOAD_REMOTE_FILE* pOutParam, int nWaitTime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] pInParam	File download input parameter
	[out] pOutParam	File download output parameter
	[in] nWaitTime	Timeout duration
Return value	Return TRUE for success and FALSE for failure.	
Note	None	

3.3 Parking Lot

3.3.1 Barrier Control

3.3.1.1 CLIENT_ControlDeviceEx

For the interface function, see "3.2.2.2 CLIENT_ControlDeviceEx."

Table 3-53 Control type

emType enumeration definition	Meaning	Corresponding structure of pInBuf
DH_CTRL_OPEN_STROBE	Open barrier	NET_CTRL_OPEN_STROBE
DH_CTRL_CLOSE_STROBE	Close barrier	NET_CTRL_CLOSE_STROBE

3.3.1.2 CLIENT_SetConfig

Table 3-54 Set barrier configuration

Item	Description
Description	Set barrier configuration.
Function	BOOL CLIENT_SetConfig (LLONG lLoginID NET_EM_CFG_OPERATE_TYPE emCfgOpType int nChannelID void* szInBuffer DWORD dwInBufferSize int waittime=3000 int * restart=NULL void * reserve=NULL

Item	Description	
	);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] emCfgOpType	Set cofniguration type Barrier configuration: NET_EM_CFG_TRAFFICSTROBE
	[out] nChannelID	Channel number.
	[in] szInBuffer	The buffer address of the confuguration.
	[in] dwInBufferSize	The size of the buffer address.
	[in] waittime	Timeout.
	[in] restart	Whether to restart.
	[in] reserve	Reserved parameters
Return value	● Success: TRUE ● Failure: FALSE.	
Note	None.	

3.3.1.3 CLIENT_GetConfig

Table 3-55 Get barrier configuration

Item	Description	
Description	Get barrier configuration	
Function	BOOL CLIENT_GetConfig (LLONG lLoginID NET_EM_CFG_OPERATE_TYPE emCfgOpType int nChannelID void* szOutBuffer DWORD dwOutBufferSize int waittime=3000 void * reserve=NULL);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] emCfgOpType	Set cofniguration type Barrier configuration: NET_EM_CFG_TRAFFICSTROBE
	[out] nChannelID	Channel number.
	[in] szInBuffer	Get he buffer address of the confuguration.
	[in] dwInBufferSize	The size of the buffer address.
	[in] waittime	Timeout.
	[in] reserve	The size of gotten configuration.
Return value	● Success: TRUE ● Failure: FALSE.	
Note	None.	

3.3.1.4 CLIENT_SetDVRMessCallBack

Table 3-56 Set vehicle location information callback

Item	Description	
Description	Set vehicle location information callback	
Function	void CLIENT_SetDVRMessCallBack(fMessCallBack cbMessage, LDWORD dwUser);	
Parameter	[in] cbMessage	Alarm callback
	[in] dwUser	User data.
Return value	None.	
Note	Call CLIENT_SetDVRMessCallBack interface before alarm subscribe; the set callback cannot include the event with pictures.	

3.3.1.5 CLIENT_StartListenEx

Table 3-57 Subscribe vehicle location information

Item	Description	
Description	Subscribe vehicle location information	
Function	BOOL CLIENT_StartListenEx(LLONG lLoginID);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
Return value	● Success: TRUE ● Failure: FALSE.	
Note	The all alarm events are reported to the users through the calback set by CLIENT_SetDVRMessCallBack interface.	

3.3.1.6 CLIENT_StopListen

Table 3-58 Stop subscribing vehicle location information

Item	Description	
Description	Stop subscribing vehicle location information	
Function	BOOL CLIENT_StopListen(LLONG lLoginID);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
Return value	● Success: TRUE ● Failure: FALSE.	
Note	None.	

3.3.1.7 CLIENT_RealLoadPictureEx

For details, see "3.2.2.1 CLIENT_RealLoadPictureEx."

3.3.1.8 CLIENT_StopLoadPic

For details, see "3.2.2.3 CLIENT_StopLoadPic"

3.3.2 Importing/Exporting Allowlist/Blocklist: CLIENT_FileTransmit

Table 3-59 Importing/Exporting Allowlist/Blocklist

Item	Description	
Description	Transmit files.	
Function	LLONG CLIENT_FileTransmit (LLONG lLoginID, Int nTransType char* szInBuf int nInBufLen fTransFileCallBack cbTransFile LDWORD dwUserData Int waittime);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] nTransType	File control type. See Table 3-44.
	[in] szInBuf	Input data, see Table 3-44.
	[in] nInBufLen	The size of nInBufLen is no smaller tham that of szInBufszInBuf structure.
	[in] cbTransFile	fTransFileCallBack.
	[in] dwUserData	Custom data.
	[in] waittime	Timeout.
Return value	● Start sending/downloading allowlist/blocklist, when the return file handle> 0, it is a valid handle; when the return file handle≤0, it is an invalid handle. ● Success: TRUE; failure: FALSE.	
Note	None.	

Table 3-60 File control type

nTransType enumerate definition	Value	Description	szInBuf
DH_DEV_BLACKWHITETRANS_START	0x0003	Start sending allowlist/blocklist	DHDEV_BLACKWHITE_LIST_INFO
DH_DEV_BLACKWHITETRANS_SEND	0x0004	Send allowlist/blocklist	LONG, the return enumerate of starting sending file
DH_DEV_BLACKWHITETRANS_STOP	0x0005	Stop sending allowlist/blocklist	LONG, the return enumerate of starting sending file
DH_DEV_BLACKWHITE_LOAD	0x0006	Download allowlist/blocklist	DHDEV_LOAD_BLACKWHITE_LIST_INFO
DH_DEV_BLACKWHITE_LOAD_STOP	0x0007	Stop downloading allowlist/blocklist	LONG, the return enumerate of starting sending file

3.3.3 Voice Talk

3.3.3.1 CLIENT_GetDevProtocolType

Table 3-61 Get the supported voice talk type

Item	Description	
Description	Get the supported voice talk type.	
Function	BOOL CLIENT_GetDevProtocolType(LLONG lLoginID, EM_DEV_PROTOCOL_TYPE *pemProtocolType);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[out] pemProtocolType	The supported protocol type, the corresponding structure is EM_DEV_PROTOCOL_TYPE.
Return value	● Success: TRUE ● Failure: FALSE.	
Note	None.	

3.3.3.2 CLIENT_SetDeviceMode

Table 3-62 Set the working mode of voice talk

Item	Description	
Description	Set the working mode of voice talk.	
Function	BOOL CLIENT_SetDeviceMode(LLONG lLoginID, EM_USEDEV_MODE emType, void *pValue);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[out] emType	enumeration value.
	[in] pValue	The the corresponding structure data pointer of the enumeration value, see Table 3-47.
Return value	● Success: TRUE ● Failure: FALSE.	
Note	None.	

Table 3-63 Relationship of emType and pValue

emType	Description	pValue
DH_TALK_ENCODE_TYPE	Talk in the pointed node.	DHDEV_TALKDECODE_INFO
DH_TALK_CLIENT_MODE	Set voice talk client.	None.
DH_TALK_SPEAK_PARAM	Set speak parameters.	NET_SPEAK_PARAM
DH_TALK_MODE3	Set speak parameters of the the third generation deveice.	NET_TALK_EX

3.3.3.3 CLIENT_StartTalkEx

Table 3-64 Start voice talk

Item	Description	
Description	Start voice talk.	
Function	LLONG CLIENT_StartTalkEx(LLONG lLoginID, pfAudioDataCallBack pfcb, LDWORD dwUser);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] pfcb	Audio data callback.
	[in] dwUser	The parameters of audio data callback.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.3.3.4 CLIENT_StopTalkEx

Table 3-65 Stop voice talk

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

3.3.3.5 CLIENT_RecordStartEx

Table 3-66 Start local record

Item	Description	
Description	Start local record.	
Function	BOOL CLIENT_RecordStartEx(LLONG lLoginID);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
Return value	● Success: TURE. ● Failure: FALSE.	
Note	This interface is only valid in Windows.	

3.3.3.6 CLIENT_RecordStopEx

Table 3-67 Stop local record

Item	Description	
Description	Stop local record.	
Function	BOOL CLIENT_RecordStopEx(LLONG lLoginID);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	This interface is only valid in Windows.	

3.3.3.7 CLIENT_TalkSendData

Table 3-68 Set audio data to devices

Item	Description	
Description	Set audio data to devices.	
Function	LONG CLIENT_TalkSendData(LLONG lTalkHandle, char *pSendBuf, DWORD dwBufSize);	
Parameter	[in] lTalkHandle	Return value of CLIENT_StartTalkEx.
	[in]pSendBuf	The pointer of the audio data module to be sent.
	[in]dwBufSize	The length of the audio data module to be sent, unit: byte.
Return value	● Success: The length of the audio data module. ● Failure: –1.	
Note	None.	

3.3.3.8 CLIENT_AudioDecEx

Table 3-69 Decode audio data

Item	Description	
Description	Decode audio data.	
Function	BOOL CLIENT_AudioDecEx(LLONG lTalkHandle, char *pAudioDataBuf, DWORD dwBufSize);	
Parameter	[in] lTalkHandle	Return value of CLIENT_StartTalkEx.
	[in] pAudioDataBuf	The pointer of the audio data module to be decoded.

Item	Description	
	[in] dwBufSize	The length of the audio data module to be decoded, unit: byte.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.3.3.9 CLIENT_SetDVRMessCallBack

Set the device requesting the other device to start voice talk event. For details, see "CLIENT_SetDVRMessCallBack."

3.3.3.10 CLIENT_StartListenEx

Subscribe the device requesting the other device to start voice talk event. For details, see "3.3.1.5 CLIENT_StartListenEx."

3.3.3.11 CLIENT_StopListen

Stop subscribing the device requesting the other device to start voice talk event. For details, see "3.3.1.6 CLIENT_StopListen."

3.3.4 Dot-matrix Display Content Control and Broadcast

For Interface function details, See“3.2.2.2 CLIENT_ControlDeviceEx“.

emType is DH_CTRL_SET_PARK_CONTROL_INFO.

3.3.5 Dot-matrix Display Character Control

3.3.5.1 CLIENT_SetConfig

Set the dot-matrix display configuration. For details, see "3.3.1.2 CLIENT_SetConfig." emCfgOpType is NET_EM_CFG_TRAFFIC_LATTIC_SCREEN.

3.3.5.2 CLIENT_GetConfig

Get the dot-matrix display configuration. For details, see "3.3.1.3 CLIENT_GetConfig." emCfgOpType is NET_EM_CFG_TRAFFIC_LATTIC_SCREEN.

3.3.6 Parking Space Indicator Configuration

3.3.6.1 CLIENT_PacketData

Table 3-70 Pack the configuration

Item	Description	
Description	Pack the cofiguration.	
Function	BOOL CLIENT_PacketData(char* szCommand, LPVOID lpInBuffer, DWORD dwInBufferSize, char* szOutBuffer, DWORD dwOutBufferSize);	
Parameter	[in] szCommand	Command parameter. Parking space indicator configuration: CFG_CMD_PARKING_SPACE_LIGHT_GROUP.
	[in] lpInBuffer	Input buffer.
	[in] dwInBufferSize	The size of the input buffer.
	[out] szOutBuffer	Output buffer.
	[in] dwOutBufferSize	The size of the output buffer.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.3.6.2 CLIENT_SetNewDevConfig

Table 3-71 Set the configuration

Item	Description	
Description	Set the cofiguration.	
Function	BOOL CLIENT_SetNewDevConfig(LLONG lLoginID, char* szCommand, int nChannelID, char* szInBuffer, DWORD dwInBufferSize, int *error, int *restart, int waittime=500);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] szCommand	Command parameter. Parking space indicator configuration: CFG_CMD_PARKING_SPACE_LIGHT_GROUP.

Item	Description	
	[in] nChannelID	Channel number.
	[in] szInBuffer	Input buffer. It is used for the configured json series information.
	[in] dwInBufferSize	The size of the buffer address.
	[out] error	Error code address.
	[in] restart	Restart sign address.
	[in] waittime	Timeout.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.3.6.3 CLIENT_GetNewDevConfig

Table 3-72 Get the configuration

Item	Description	
Description	Get the cofiguration.	
Function	BOOL CLIENT_GetNewDevConfig(LLONG lLoginID, char* szCommand, int nChannelID, char* szOutBuffer, DWORD dwOutBufferSize, int *error, int waittime=500);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] szCommand	Command parameter. Parking space indicator configuration: CFG_CMD_PARKING_SPACE_LIGHT_GROUP.
	[in] nChannelID	Channel number.
	[in] szOutBuffer	Output buffer. It is used for the configured json series information..
	[in] dwInBufferSize	The size of the buffer address.
	[out] error	Error code address.
	[in] waittime	Timeout.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.3.6.4 CLIENT_ParseData

Table 3-73 Parse the configuration

Item	Description	
Description	Paese the cofiguration.	
Function	BOOL CLIENT_ParseData(char* szCommand, char* szInBuffer, LPVOID lpOutBuffer, DWORD dwOutBufferSize, void* pReserved);	
Parameter	[in] szCommand	Command parameter. Parking space indicator configuration: CFG_CMD_PARKING_SPACE_LIGHT_GROUP.
	[in] szInBuffer	Input buffer, character configuration buffer.
	[in] lpOutBuffer	Output buffer.
	[out]dwOutBufferSize	The size of output buffer.
	[in] pReserved	Reserved parameters.
Return value	● Success: TRUE. ● Failure: FALSE.	
Note	None.	

3.3.7 Parking Space Status Indicator Configuration

3.3.7.1 CLIENT_SetConfig

Set the parking space status indicator. For details, see "3.3.1.2 CLIENT_SetConfig."

emCfgOpType is NET_EM_CFG_PARKINGSPACELIGHT_STATE

3.3.7.2 CLIENT_GetConfig

Get the parking space status indicator. For details, see "3.3.1.3 CLIENT_GetConfig."

emCfgOpType is NET_EM_CFG_PARKINGSPACELIGHT_STATE.

3.4 Device Configuration

3.4.1 Auto Registration

3.4.1.1 CLIENT_ParseData

Table 3-74 Parse the searched configuration information

Item	Description
Description	Parse the searched configuration information.

Item	Description	
Function	BOOL CLIENT_ParseData (char *szCommand, char *szInBuffer, LPVOID lpOutBuffer, DWORD dwOutBufferSize, int *pReserved);	
Parameter	[in] szCommand	Command parameter
	[in] szInBuffer	Input buffer: Character configuration buffer
	[out] lpOutBuffer	Output buffer
	[in] dwOutBufferSize	Output buffer size
	[in] pReserved	Reserved parameter
Return value	Return TRUE for success and FALSE for failure.	
Note	None	

Table 3-75 Comparison of szCommand, search type and corresponding structure

szCommand	Search Type	Corresponding Structure
CFG_CAP_CMD_ACCESSCONTROLMANAGER	Access control capability	CFG_CAP_ACCESSCONTROL
CFG_CMD_NETWORK	IP configuration	CFG_NETWORK_INFO
CFG_CMD_DVRIP	Auto registration configuration	CFG_DVRIP_INFO
CFG_CMD_NTP	NTP time synchronization	CFG_NTP_INFO
CFG_CMD_ACCESS_EVENT	Access control configuration(door configuration information, period configuration of Normally Open (NO) and Normally Closed (NC), unlock at designated intervals, first card unlocking configuration)	CFG_ACCESS_EVENT_INFO
CFG_CMD_ACCESSTIMESCHEDULE	Card swiping period for access control (period configuration)	CFG_ACCESS_TIMESCHEDULE_INFO
CFG_CMD_OPEN_DOOR_GROUP	Group combination unlock configuration	CFG_OPEN_DOOR_GROUP_INFO
CFG_CMD_ACCESS_GENERAL	Basic configuration for access control (multi-door interlock)	CFG_ACCESS_GENERAL_INFO
CFG_CMD_OPEN_DOOR_ROUTE	Collection of routes to open the door, also called anti-passback route configuration	CFG_OPEN_DOOR_ROUTE_INFO

3.4.1.2 CLIENT_GetNewDevConfig

Table 3-76 Get configurations in string format

Item	Description
Description	Get configurations in string format

Item	Description	
Function	BOOL CLIENT_GetNewDevConfig (LLONG lLoginID, char *szCommand, int nChannelID, char *szOutBuffer, DWORD dwOutBufferSize, int *error, int waittime =500);	
Parameter	[in] lLoginID	Login handle
	szCommand	Command parameter. Refer to"parsing the configuration information searched: CLIENT_ParseData".
	[in] nChannelID	Channel No.
	[out]szOutBuffer	Output buffer
	[in] dwOutBufferSize	Output buffer size
	[out] error	Error code
	[in] waittime	Timeout period for waiting
Return value	If succeeded, return True; If failed, return False.	
Note	Get configuration in string format and parse with CLIENT_ParseData.	

Table 3-77 Parameter error code and description

Error code	Description
0	Succeeded
1	Failed
2	Invalid data.
3	Unable to set for now
4	No permission.

3.4.1.3 CLIENT_SetNewDevConfig

Table 3-78 Get configuration information in string format

Item	Description	
Description	Get configuration information in string format.	
Function	BOOL CLIENT_SetNewDevConfig (LLONG lLoginID, char *szCommand, int nChannelID, char *szInBuffer, DWORD dwInBufferSize, int *error, int * restart int waittime =500);	
Parameter	[in] lLoginID	Login handle

Item	Description	
	szCommand	Command parameter information. See“3.4.1.1CLIENT_ParseData”.
	[in] nChannelID	Channel No.
	[in] szInBuffer	Output buffer
	[in] dwInBufferSize	Output buffer size
	[out] error	Error code
	[out] restart	Configure whether to restart the device or not: 1 means restarting and 0 means not restarting.
	[in] waittime	Timeout period for waiting
Return value	If succeeded, return True; If failed, return False.	
Note	Set configuration information in string format, and pack with CLIENT_PacketData.	

Table 3-79 Parameter error code and description

Error code	Description
0	Succeeded
1	Failed
2	Invalid data.
3	Unable to set for now
4	No permission.

3.4.1.4 CLIENT_PacketData

Table 3-80 Pack the configuration informationin string format.

Item	Description	
Description	Pack the configuration informationin string format.	
Function	BOOL CLIENT_PacketData (char *szCommand, LPVOID lpInBuffer, DWORD dwInBufferSize, char *szOutBuffer, DWORD dwOutBufferSize);	
Parameter	[out] szCommand	Command parameter. For details, see“3.4.1.1 CLIENT_ParseData”.
	[in] lpInBuffer	Input buffer. For structure type, see“3.4.1.1CLIENT_ParseData”.
	[in] dwInBufferSize	Input buffer size
	[out] szOutBuffer	Output buffer
	[in] dwOutBufferSize	Output buffer size
Return value	If succeeded, return TRUE; If failed, return FALSE.	
Note	None	

3.4.2 Viewing Device Information

3.4.2.1 CLIENT_QueryNewSystemInfo

Table 3-81 Search for system capability information in string format

Item	Description	
Description	Search for system capability information in string format.	
Function	LLONG lLoginID, char *szCommand, int nChannelID, char *szOutBuffer, DWORD dwOutBufferSize, int *error, int nWaitTime = 1000);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in] szCommand	Command parameter
	[in] nChannelID	Channel No.
	[out] szOutBuffer	Protocol buffer received
	[in] dwOutBufferSize	Total number of bytes received (in bytes)
	[out] error	Error code
	[in]waittime	Timeout period: 1000 ms by default,or set based on actual needs.
Return value	If succeeded, return TRUE; If failed, return FALSE.	
Note	Get information in string format and parse with CLIENT_ParseData.	

Table 3-82 Parameter error code and description

Error code	Description
0	Succeeded
1	Failed
2	Invalid data
3	Unable to set for now
4	No permission.

3.4.2.2 CLIENT_ParseData

Table 3-83 Parse the searched configuration information

Item	Description
Description	Parse the searched configuration information

Item	Description	
Function	BOOL CLIENT_ParseData (char *szCommand, char *szInBuffer, LPVOID lpOutBuffer, DWORD dwOutBufferSize, int *pReserved);	
Parameter	[in] szCommand	Command parameter
	[in] szInBuffer	Input buffer: character configuration buffer.
	[out] lpOutBuffer	Output buffer. For structure types, see Figure 3-87.
	[in] dwOutBufferSize	Output buffer size
	[in] pReserved	Reserved parameter
Return value	If succeeded, return TRUE; If failed, return FALSE.	
Note	None	

Table 3-84 Comparison of szCommand, search type and corresponding structure.

szCommand	Search Type	Corresponding Structure
CFG_CAP_CMD_ACCESS CONTROLMANAGER	Access control capability	CFG_CAP_ACCESSCONTROL
CFG_CMD_NETWORK	IP configuration	CFG_NETWORK_INFO
CFG_CMD_DVRIP	Auto registration configuration	CFG_DVRIP_INFO
CFG_CMD_NTP	NTP time synchronization	CFG_NTP_INFO
CFG_CMD_ACCESS_EVEN T	Access control configuration (door configuration information, period configuration of NO and NC, unlock at designated intervals, first card unlocking configuration)	CFG_ACCESS_EVENT_INFO
CFG_CMD_ACCESSTIMES CHEDULE	Card swiping period for access control (period configuration)	CFG_ACCESS_TIMESCHEDUL E_INFO
CFG_CMD_OPEN_DOOR_ GROUP	Group combination unlock configuration	CFG_OPEN_DOOR_GROUP_I NFO
CFG_CMD_ACCESS_GEN ERAL	Basic configuration for access control (multi-door interlock)	CFG_ACCESS_GENERAL_INF O
CFG_CMD_OPEN_DOOR_ ROUTE	Collection of routes to open the door, also called anti-passback route configuration	CFG_OPEN_DOOR_ROUTE_I NFO

3.4.2.3 CLIENT_GetDevCaps

Table 3-85 Get device capabilities

Item	Description
Description	Get device capabilities

Item	Description	
Function	BOOL CLIENT_GetDevCaps (LLONG lLoginID, int nType, void* pInBuf, void* pOutBuf, int nWaitTime);	
Parameter	[in] lLoginID	Login handle
	[in] nType	Device Type Control parameters vary by type
	[in] pInBuf	Get device capabilities (input parameter)
	[out] pOutBuf	Get device capabilities (output parameter)
	[in] nWaitTime	Timeout duration
Return value	If succeeded, return TRUE; If failed, return FALSE.	
Note	None	

For the comparison of nType, pInBuf and pOutBuf, see Table 3-89.

Table 3-86 Comparison of nType, pInBuf and pOutBuf

nType	Description	pInBuf	pOutBuf
NET_FACEINFO_CAPS	Get the capability collection of face access controller	NET_IN_GET_FACEINFO_CAPS	NET_OUT_GET_FACEINFO_CAPS

3.4.2.4 CLIENT_QueryDevState

Table 3-87 Get the working status of camera

Item	Description	
Description	Get the working status of camera	
Function	BOOL CLIENT_QueryDevState (LLONG lLoginID, int nType, char *pBuf, int nBufLen, int *pRetLen, int waittime=1000);	
Parameter	[in] lLoginID	Login handle
	[in] nType	Device Type Control parameters vary by type.
	[out] pBuf	Output parameter, used to receive the returned data buffer from search. Based on different search types, the structures of returned data vary.
	[in] nBufLen	Buffer length (in bytes)
	[in] waittime	Timeout duration
Return value	If succeeded, return TRUE. If failed, return FALSE.	

Item	Description
Note	None

For the correspondence between nType, search type and structure, see Table 3-91.

Table 3-88 Correspondence between nType, search type and structure

nType	Description	pBuf
DH_DEVSTATE_SOFTWARE	Search for the software version information of the device	DHDEV_VERSION_INFO
DH_DEVSTATE_NETINTERFACE	Search for the network interface information	DHDEV_NETINTERFACE_INFO
DH_DEVSTATE_DEV_RECORDSET	Search for the device record database	NET_CTRL_RECORDSET_PARAM
DH_DEVSTATE_DOOR_STATE	Search for access control status (door contact)	NET_DOOR_STATUS_INFO

3.4.3 Importing and Exporting Configuration Information

3.4.3.1 CLIENT_ImportConfigFileJson

Table 3-89 Importing configuration information

Item	Description	
Description	Import configuration information	
Function	CLIENT_NET_API BOOL CALL_METHOD CLIENT_ImportConfigFileJson(LLONG lLoginID, char *pSendBuf, int nSendBufLen, void* reserved=NULL, int nWaitTime=3000);	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] pSendBuf	Configure input buffer area. The user determines the memeory.
	[in] nSendBufLen	Configure the length of the input buffer area. The user determines the length.
	[in] reserved	Reserve parameter
	[in] nWaitTime	Timout threshold 3000 ms by default.
Return value	● If succeeded, return True. ● If failed, return False.	
Note	None	

3.4.3.2 CLIENT_ExportConfigFileJson

Table 3-90 Exporting configuration information

Item	Description
Description	Export configuration information
Function	CLIENT_NET_API BOOL CALL_METHOD CLIENT_ExportConfigFileJson(LLONG lLoginID, char *pOutBuffer, int maxlen, int *nRetlen, void* reserved=NULL, int nWaitTime=3000);

Item	Description	
Parameter	[in] lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[out] pOutBuffer	Configure receive buffer area. The user determines the memeory.
	[in]maxlen	Configure the length of the receive buffer area. The user determines the length.
	[out] nRetlen	The actual length of exported configuration
	[in] reserved	Reserved parameter
	[in] nWaitTime	Timeout threshold: maximum 3000 ms by default.
Return value	● If succeeded, return True. ● If failed, return False.	
Note	None	

4 Callback Definition

4.1 fSearchDevicesCB

Table 4-1 Callback of searching devices (1)

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

4.2 fSearchDevicesCBEx

Table 4-2 Callback of searching devices (2)

Item	Description	
Description	Callback of searching devices.	
Function	typedef void(CALLBACK * fSearchDevicesCBEx)(LLONG lSearchHandle, DEVICE_NET_INFO_EX2 *pDevNetInfo, void* pUserData);	
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
Description	Disconnection callback.
Function	typedef void (CALLBACK *fDisConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser

Item	Description	
	);	
Parameter	[out] lLoginID	Return value of 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	
Description	Reconnection callback.	
Function	typedef void (CALLBACK *fHaveReConnect)(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, LDWORD dwUser);	
Parameter	[out] lLoginID	Return value of 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	
Description	Callback of real-time monitoring data.	
Function	typedef void (CALLBACK * fRealDataCallBackEx2)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LLONG param, LDWORD dwUser);	
Parameter	[out] lRealHandle	Return value of CLIENT_RealPlayEx.
	[out] dwDataType	Data type: ● 0: Initial data. ● 1: Data with frame information.

Item	Description	
		● 2: YUV data. ● 3: PCM audio data.
	[out] pBuffer	Address of monitoring data block.
	[out] dwBufSize	Length (unit: byte) of the monitoring data block
	[out] param	Callback parameter structure. Different dwDataType value corresponds to different type. ● The param is blank pointer when dwDataType is 0. ● The param is the pointer of tagVideoFrameParam structure when dwDataType is 1. ● The param is the pointer of tagCBYUVDataParam structure when dwDataType is 2. ● The param is the pointer of tagCBPCMDataParam structure when dwDataType is 3.
	[out] dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.6 fDownLoadPosCallBack

Table 4-6 Callback of media file download process

Item	Description	
Description	Callback of media file download process.	
Function	typedef void (CALLBACK *fDownLoadPosCallBack)(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);	
Parameter	[out]lPlayHandle	Return value of CLIENT_DownloadMediaFile.
	[out]dwTotalSize	Total size.
	[out]dwDownLoadSize	The downloaded data size. ● -1: Download finish. ● -2: Data write error during downloading.
	[out]dwUser	User parameter of the callback.
Return value	None.	
Note	None.	

4.7 fAnalyzerDataCallBack

Table 4-7 Callback of intelligent event information

Item	Description
Description	Callback of intelligent event information.

Item	Description	
Function	typedef int (CALLBACK *fAnalyzerDataCallBack)(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE* pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void* reserved);	
Parameter	[out]lAnalyzerHandle	Return value of CLIENT_RealLoadPictureEx.
	[out]dwAlarmType	Type of intelligent event, see Table 3-30.
	[out]pAlarmInfo	Cache of event information, see Table 3-30.
	[out]pBuffer	Pictures cache.
	[out]dwBufSize	Cache size of pictures.
	[out]dwUser	User parameter of the callback.
	[out] nSequence	The value of nSequenc denotes the occurance of the same image. ● 0: means the image appears for the first time. ● 1: means the image will appear later. ● 2: means the image appears for the last time or appears only once.
	[out]reserved	Reserved.
Return value	None.	
Note	None.	

4.8 fFluxStatDataCallBack

Table 4-8 Callback of intelligent event information

Item	Description	
Description	Callback of intelligent event information.	
Function	typedef int (CALLBACK *fFluxStatDataCallBack)(LLONG lFluxStatHandle, DWORD dwEventType, void* pEventInfo, BYTE* pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void* reserved);	
Parameter	[out]lFluxStatHandle	Return value of CLIENT_StartTrafficFluxStat.
	[out]dwEventType	Type of intelligent event information.
	[out]pEventInfo	Vehicle flow event information.

Item	Description	
	[out]pBuffer	Data cache.
	[out]dwBufSize	Data size.
	[out]dwUser	User parameter of the callback.
	[out]nSequence	Sequence.
	[out]reserved	Reserved.
Return value	None.	
Note	The pEventInfo corresponds to DEV_EVENT_TRAFFIC_FLOWSTAT_INFO structure.	

4.9 fTransFileCallBack

Table 4-9 Callback of file transmission

Item	Description	
Description	Callback of file transmission.	
Function	typedef int (CALLBACK *fFluxStatDataCallBack)(LLONG lHandle, int nTransType, int nState, int nSendSize, int nTotalSize, LDWORD dwUser);	
Parameter	c	File transmission handle.
	[out] lHandle	The type of file transmission.
	[out] nTransType	The status of file transmission.
	[out] nState	The length of the sent file.
	[out] nSendSize	The total size of the file.
	[out] nTotalSize	Custom data.
Return value	None.	
Note	None.	

4.10 pfAudioDataCallBack

Table 4-10 Callback of audio data of voice talk

Item	Description	
Description	Callback of audio data of voice talk.	
Function	typedef void (CALLBACK *pfAudioDataCallBack)(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag, LDWORD dwUser);	
Parameter	[out] lTalkHandle	Return value of CLIENT_StartTalkEx.

Item	Description	
	[out] pDataBuf	The address of audio data module.
	[out] dwBufSize	The length of audio data module.
	[out] byAudioFlag	Data type signs: ● 0: Indicates it is from local collection. ● 1: Indicates it is from device sending.
	[out] dwUser	Callback of user parameters .
Return value	None.	
Note	None.	

4.11 fDataCallBack

Table 4-11 Video playback data callback function

Item	Description	
Description	Video playback data callback function	
Preconditions	None	
Function	typedef int (CALLBACK *fDataCallBack)(LLONG lRealHandle, DWORD dwDataType, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser);	
Parameter	[out] lRealHandle	Video playback handle Return value of video playback interfaces including CLIENT_PlayBackByTimeEx
	[out]dwDataType	0 (original data)
	[out] pBuffer	Data buffer, used to store the video data for the callback.
	[out] dwBufSize	buffer length (in bytes)
	[out] dwUser	User data, identical with the imported data when users set fDataCallBack.
Return value	● 0: This callback failed, and the same data return for next callback. ● 1: This callback succeeded, and subsequent data are returned for next callback.	
Note	Set the callback function in video playback interfaces including CLIENT_PlayBackByTimeEx. When setting the callback function, if hWnd isn't null, we reagrd the callback succeeded whatever returned value of the callback function is and subsequent data will be returned for next callback. Users can only tell the corresponding callback data of stream pulling through lRealHandle in the callback function.	

4.12 fTimeDownLoadPosCallBack

Table 4-12 Video download process callback function

Item	Description
Description	Video download process callback function
Preconditions	None
Function	typedef void (CALLBACK *fTimeDownLoadPosCallBack) (LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, int index, NET_RECORDFILE_INFO recordfileinfo, LDWORD dwUser);
Parameter	lPlayHandle Record download handle Return value of video playback interfaces including CLIENT_DownloadByTimeEx. dwTotalSize Total size (KB) dwDownLoadSize Download size (KB) index The serial number of the video being downloaded, starting from 0. recordfileinfo Information about the video being downloaded. For details, see NET_RECORDFILE_INFO structure description. dwUser User data, identical with the imported data when users set fDataCallBack.
Return value	None
Note	Set the callback function in video playback by time interfaces including CLIENT_PlayBackByTimeEx. Users can only tell the corresponding process callback of video download through lRealHandle in the callback function.

4.13 fDownLoadPosCallBack

Table 4-13 Callback function of playback progress by time

Item	Description
Description	Callback function of playback progress by time
Function	typedef void (CALLBACK *fDownLoadPosCallBack)(LLONG lPlayHandle, DWORD dwTotalSize, DWORD dwDownLoadSize, LDWORD dwUser);

Item	Description	
Parameter	[out]lPlayHandle	Return value of the playback or download interface
	[out]dwTotalSize	Total size (KB)
	[out]dwDownLoadSize	Download size (KB) ● -1: Playback ends ● -2: Failed to write the file
	[out]dwUser	User data
Return value	None	
Note	None	

4.14 fCameraStateCallBack

Table 4-14 Remote device status callback

Item	Description	
Description	Remote device status callback	
Function	void (CALLBACK *fCameraStateCallBack) (LLONG lLoginID, LLONG lAttachHandle, const NET_CB_CAMERASTATE *pBuf, int nBufLen, LDWORD dwUser);	
Parameter	[out] lLoginID	Return value of login interface
	[out] lAttachHandle	Return value of subscription interface
	[out] pBuf	Camera status
	[out] nBufLen	Return data length
	[out] dwUser	User-defined data
Return value	None	
Note	After subscribing to remote device status, when camera status changes, the device reports corresponding information to users.	

5 Intelligent Traffic Event Macro

Table 5-1 Intelligent traffic event macro

Event	Macro	Value
ANPR	EVENT_IVS_PEDESTRIAN_JUNCTION	0x00000230
Running a Red Light	EVENT_IVS_TRAFFIC_NONMOTOR_RUN_REDLIGHT	0x00000310
Crossing Solid White Line	EVENT_IVS_TRAFFIC_PARKINGSPACEOVERLINE	0x00000134
Wrong-way Driving	EVENT_IVS_TRAFFIC_RETROGRADE	0x00000102
Driving Slowly	EVENT_IVS_TRAFFIC_UNDERSPEED	0x00000107
Speeding	EVENT_IVS_HIGHSPEED	0x0000022B
Vehicle in Lane	EVENT_IVS_TRAFFIC_VEHICLEINROUTE	0x0000011B
Heavy Vehicle in Lane	EVENT_IVS_TRAFFIC_YELLOWPLATEINLANE	0x0000010E
Illegal left turn	EVENT_IVS_TRAFFIC_TURNLEFT	0x00000103
Illegal right turn	EVENT_IVS_TRAFFIC_TURNRIGHT	0x00000104
Illegal U-turn	EVENT_IVS_TRAFFIC_UTURN	0x00000105
Illegal parking	EVENT_IVS_PARKINGDETECTION	0x00000116
Traffic Congestion	EVENT_IVS_CONGESTION_DETECTION	0x00000284
Illegal Lane Change	EVENT_IVS_TRAFFIC_CROSSLANE	0x0000010A
Crossing Solid Yellow Line	EVENT_IVS_TRAFFIC_OVERYELLOWLINE	0x0000010B
Traffic Standstill	EVENT_IVS_TRAFFIC_STAY	0x0000011A
Failed to Yield to Pedestrians	EVENT_IVS_TRAFFIC_PEDESTRAINPRIORITY	0x0000010F
Disobeying Lane Direction Sign	EVENT_IVS_TRAFFIC_WRONGROUTE	0x00000109
Illegal Backing	EVENT_IVS_TRAFFIC_BACKING	0x00000125
Crossing Stop Line	EVENT_IVS_TRAFFIC_OVERSTOPLINE	0x00000137
Running a Yellow Light	EVENT_IVS_TRAFFIC_RUNYELLOWLIGHT	0x00000127
Parking in Yellow Grid	EVENT_IVS_TRAFFIC_PARKINGONYELLOWBOX	0x0000012A
Restricted License Plates	EVENT_IVS_TRAFFIC_RESTRICTED_PLATE	0x00000136
No Entry	EVENT_IVS_TRAFFIC_NOPASSING	0x00000111
Failed to Yield to Vehicles Going Straight When Turning Right	EVENT_IVS_TRAFFIC_TURNRIGHTAFTERSTRAIGHT	0x0000021E
Failed to Yield to Pedestrians Going Straight When Turning Right	EVENT_IVS_TRAFFIC_TURNRIGHTAFTERPEOPLE	0x0000021F
Smoking while driving	EVENT_IVS_SMOKING_DETECT	0x0000025B
Calling While Driving	EVENT_IVS_PHONECALL_DETECT	0x0000025A
Pedestrian Running a Red Light	EVENT_IVS_TRAFFIC_PEDESTRAINRUNREDLIGHT	0x0000013B
No Entry during Traffic Congestion	EVENT_IVS_TRAFFIC_JAM_FORBID_INTO	0x00000163

Event	Macro	Value
Failed to Pass Orderly According to Regulations	EVENT_IVS_TRAFFIC_PASSNOTINORDER	0x0000013C
Littering	EVENT_IVS_SPILLEDMATERIAL_DETECTION	0x00000248
Pedestrian Event	EVENT_IVS_TRAFFIC_PEDESTRAIN	0x0000012D
Failed to Yield to Vehicles Going Straight When Turning Left	EVENT_IVS_TRAFFIC_TURNLEFTAFTERSTRAIGHT	0x00000218
Turn Left without Taking the Lane Closest to the Middle of the Street	EVENT_IVS_TRAFFIC_BIGBENDSMALLTURN	0x00000219
Vehicle Queue Jumping	EVENT_IVS_TRAFFIC_QUEUEJUMP	0x0000021C
Failed to Yield to Vehicles Going Straight When Turning Right	EVENT_IVS_TRAFFIC_TURNRIGHTAFTERSTRAIGHT	0x0000021E
Failed to Yield to Pedestrians Going Straight When Turning Right	EVENT_IVS_TRAFFIC_TURNRIGHTAFTERPEOPLE	0x0000021F
High Beam Violation	EVENT_IVS_TRAFFIC_HIGH_BEAM	0x00000228
No Trucks	EVENT_IVS_TRAFFIC_TRUCKFORBID	0x00000229
Pedestrian ANPR	EVENT_IVS_PEDESTRIAN_JUNCTION	0x00000230
Non-motor Vehicle in Lane	EVENT_IVS_TRAFFIC_NONMOTORINMOTORROUTE	0x0000001C
Illegal Parking B	EVENT_IVS_TRAFFIC_PARKING_B	0x00000240
Illegal Parking C	EVENT_IVS_TRAFFIC_PARKING_C	0x00000241
Illegal Parking D	EVENT_IVS_TRAFFIC_PARKING_D	0x00000242
Non-motor Vehicle Overload	EVENT_IVS_TRAFFIC_NONMOTOR_OVERLOAD	0x0000024B
No Helmet	EVENT_IVS_TRAFFIC_NONMOTOR_WITHOUTSAFEHAT	0x0000024C
Non-motor Vehicle Mounting Umbrella	EVENT_IVS_TRAFFIC_NONMOTOR_HOLDUMBRELLA	0x00000254
Out-of-store Operation	EVENT_IVS_SHOPPRESENCE	0x00000246
Motor Vehicle Illegal Parking	EVENT_IVS_CITY_MOTORPARKING	0x0000024F
Stains Obscuring License Plate	EVENT_IVS_TRAFFIC_PLATE_OCCLUSION	0x0000030B
Car racing	EVENT_IVS_TRAFFIC_VEHICLE_RACE	0x00000309
Failed to Keep a Safe Distance with the Vehicle in the Front	EVENT_IVS_NEAR_DISTANCE_DETECTION	0x00000174
Security Warning	EVENT_IVS_TRAFFIC_ROAD_ALERT	0x0000030E
Non-motor Vehicle Running a Red Light	EVENT_IVS_TRAFFIC_NONMOTOR_RUN_REDLIGHT	0x00000310
Traffic Accident	EVENT_IVS_TRAFFIC_REAREND_ACCIDENT	0x00000322

Event	Macro	Value
Emergency Lane Occupancy	EVENT_IVS_TRAFFIC_VEHICLE_IN_EMERGENCY_LANE	0x00000311
Failed to Use the Turning Signal According to Regulations	EVENT_IVS_TRAFFIC_WRONG_TURN_LIGHT	0x00000321
Non-motor Vehicle Wrong-Way Driving	EVENT_IVS_TRAFFIC_NON_MOTOR_RETROGRADE	0x00000328
Non-motor Vehicle Parking over Line	EVENT_IVS_TRAFFIC_NON_MOTOR_OVER_STOP_LINE	0x00000329
Non-motor Vehicle Illegal Parking	EVENT_IVS_CITY_NONMOTORPARKING	0x00000250
Mobile Vendors	EVENT_IVS_FLOWBUSINESS	0x0000024E
Intrusion	EVENT_IVS_INREGIONDETECTION	0x00000114
Roadblock	EVENT_IVS_TRAFFIC_ROAD_BLOCK	0x00000271
Smoke Alarm	EVENT_IVS_SMOKEDETECTION	0x0000000D
Flame Detection	EVENT_HY_FIRE_DETECTION	0x01000001
Road Construction	EVENT_IVS_TRAFFIC_ROAD_CONSTRUCTION	0x00000272
Full Garbage Can	EVENT_IVS_DUSTBIN_OVER_FLOW	0x00000260
Garbage Exposure	EVENT_IVS_GARBAGE_EXPOSURE	0x0000025F
Illegal Umbrellas	EVENT_IVS_HOLD_UMBRELLA	0x0000025E
Dirty Front Door	EVENT_IVS_DOOR_FRONT_DIRTY	0x00000261
No Motorcycle	EVENT_IVS_TRAFFIC_MOTORCYCLE_FORBID	0x00000364
Front Seat Passenger Not Wearing Seatbelt	EVENT_IVS_TRAFFIC_ASSISTANT_WITHOUT_SAFEBELT	0x0000034D
Crossing Diversion Line	EVENT_IVS_TRAFFIC_OVER_GUIDE_LINE	0x00000319
Trucks Failed to Stop While Turning Right	EVENT_IVS_TRAFFIC_TURN_RIGHT_NO_STOP	0x00000358

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.