NetSDK_Intelligent Building

Programming Manual

V2.0.0

Foreword

General

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

SDK, also known as network device SDK, is a development kit for developer to develop the interfaces for network communication among surveillance products such as Network Video Recorder (NVR), Network Video Server (NVS), IP camera (IPC), Speed Dome (SD), and intelligence devices.

The manual describes the SDK interfaces and processes of the general function modules for intelligent buildings. For more function modules and data structures, refer to *NetSDK Development Manual*. For detailed information on basic service processes, including initialization, login, general alarms and intelligent alarms, refer to *NetSDK Programming Guide*.

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

Intended Readers

- Software development engineers
- Product managers
- Project managers who use SDK

Safety Instructions

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

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

Revision History

Version	Revision Content	Release Time
V2.0.0	For detailed information on basic services, refer to *NetSDK Programming Guide*.	February 2025

Version	Revision Content	Release Time
V1.0.5	Updated some descriptions	February 2023
V1.0.4	● Deleted function library avnetsdk.dll and libavnetsdk.so related content, and changed font. ● Deleted fisheye correction library.	March 2021
V1.0.3	Add interfaces and functions of the second-generation access control.	June 2020
V1.0.2	● Modify the access controller models. ● Add fucntions of the first-generation access controller. ● Replace all device login interfaces with high-security login interfaces.	April 2020
V1.0.1	Delete some contents of table 1-1.	January 2019
V1.0.0	First release.	December 2017

About the Manual

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

Glossary

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

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

Table of Contents

1 Overview

1.1 General

The manual introduces SDK interfaces that include main functions, interface functions, and callback functions.

Main functions include: Common functions, alarm host, access control and other functions.

The development kit might include different files 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
Play (encoding/decoding) auxiliary library	dhplay.dll	Dahua play 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 required 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 the play library to parse and play the streams.
- The auxiliary library decodes the audio and video streams for the functions such as monitoring, playback and voice talk, and collects the local audio.

1.2 Applicability

1.2.1 Supported System

- Recommended memory: No less than 512 M.
- Operating system:
  - ◇ Windows
    - Support Windows 10/Windows 8.1/Windows 7 and Windows Server 2008/2003.
  - ◇ Linux
    - Support the common Linux systems such as Red Hat/SUSE.

1.2.2 Supported Devices

- Access Control (First-generation Device)
  - ◇ DH-ASC1201C-D
  - ◇ DH-ASC1202B-D, DH-ASC1202B-S, DH-ASC1202C-D, DH-ASC1202C-S
  - ◇ DH-ASC1204B-S, DH-ASC1204C-D, DH-ASC1204C-S
  - ◇ DH-ASC1208C-S
  - ◇ DH-ASI1201A, DH-ASI1201A-D, DH-ASI1201E-D, DH-ASI1201E
  - ◇ DH-ASI1212A(V2), DH-ASI1212A-C(V2), DH-ASI1212A-D(V2), DH-ASI1212D, DH-ASI1212D-D
  - ◇ DHI-ASC1201B-D, DHI-ASC1201C-D
  - ◇ DHI-ASC1202B-D, DHI-ASC1202B-S, DHI-ASC1202C-D, DHI-ASC1202C-S
  - ◇ DHI-ASC1204B-S, DHI-ASC1204C-D, DHI-ASC1204C-S
  - ◇ DHI-ASC1208C-S
  - ◇ DHI-ASI1201A, DHI-ASI1201A-D, DHI-ASI1201E-D, DHI-ASI1201E
  - ◇ DHI-ASI1212A(V2), DHI-ASI1212A-D(V2), DHI-ASI1212D, DHI-ASI1212D-D
  - ◇ ASC1201B-D, ASC1201C-D
  - ◇ ASC1202B-S, ASC1202B-D, ASC1202C-S, ASC1202C-D
  - ◇ ASC1204B-S, ASC1204C-S, ASC1204C-D
  - ◇ ASC1208C-S
  - ◇ ASI1201A, ASI1201A-D, ASI1201E, ASI1201E-D
  - ◇ ASI1212A(V2), ASI1212A-D(V2), ASI1212D, ASI1212D-D
- Access Control (Second-generation Device)
  - ◇ DH-ASI4213Y
  - ◇ DH-ASI4214Y
  - ◇ DH-ASI7213X, DH-ASI7213X-C, DH-ASI7213Y, DH-ASI7213Y-V3
  - ◇ DH-ASI7214X, DH-ASI7214X-C, DH-ASI7214Y, DH-ASI7214Y-V3
  - ◇ DH-ASI7223X-A, DH-ASI7223Y-A, DH-ASI7223Y-A-V3
  - ◇ DH-ASI8213Y(V2), DH-ASI8213Y-C(V2), DH-ASI8213Y-V3
  - ◇ DH-ASI8214Y, DH-ASI8214Y(V2), DH-ASI8214Y-C(V2), DH-ASI8214Y-V3
  - ◇ DH-ASI8215Y, DH-ASI8215Y(V2), DH-ASI8215Y-V3
  - ◇ DH-ASI8223Y(V2), DH-ASI8223Y-A(V2), DH-ASI8223Y, DH-ASI8233Y-A-V3
  - ◇ DHI-ASI1202M, DHI-ASI1202M-D
  - ◇ DHI-ASI4213Y, DHI-ASI4214Y
  - ◇ DHI-ASI7213X, DHI-ASI7213Y, DHI-ASI7213Y-D, DHI-ASI7213Y-V3

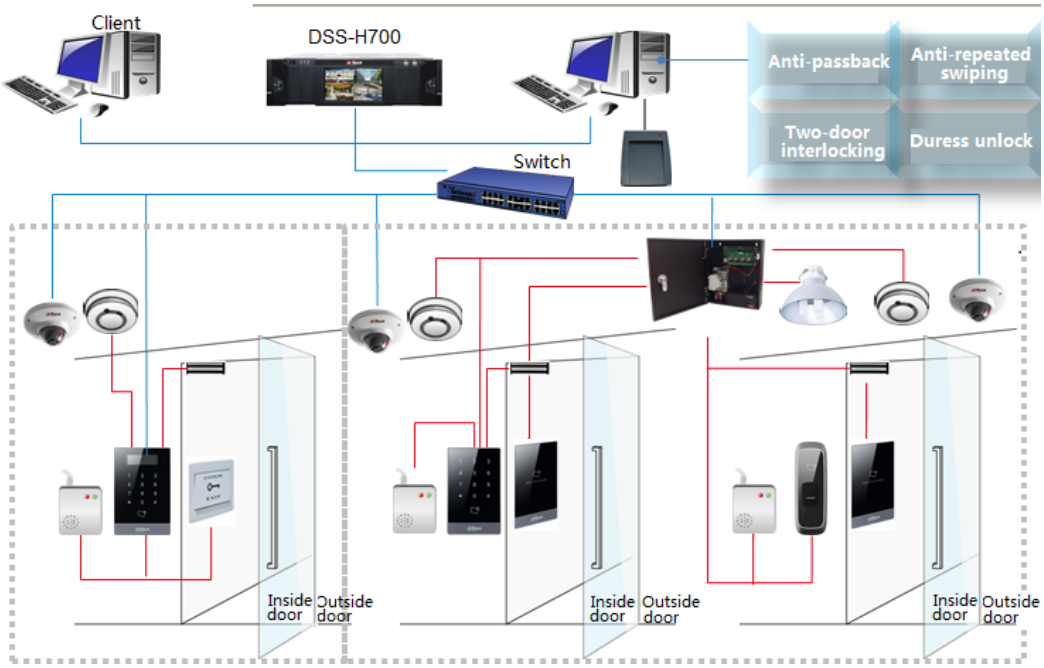
- ◇ DHI-ASI7214X, DHI-ASI7214Y, DHI-ASI7214Y-D, DHI-ASI7214Y-V3
- ◇ DHI-ASI7223X-A, DHI-ASI7223Y-A, DHI-ASI7223Y-A-V3
- ◇ DHI-ASI8213Y-V3
- ◇ DHI-ASI8214Y, DHI-ASI8214Y(V2), DHI-ASI8214Y-V3
- ◇ DHI-ASI8223Y, ASI8223Y(V2), DHI-ASI8223Y-A(V2), DHI-ASI8223Y-A-V3
- ◇ ASI1202M, ASI1202M-D
- ◇ ASI7213X, ASI7213Y-D, ASI7213Y-V3
- ◇ ASI7214X, ASI7214Y, ASI7214Y-D, ASI7214Y-V3
- ◇ ASI7223X-A, ASI7223Y-A, ASI7223Y-A-V3
- ◇ ASI8213Y-V3
- ◇ ASI8214Y, ASI8214Y(V2), ASI8214Y-V3
- ◇ ASI8223Y, ASI8223Y(V2), ASI8223Y-A(V2), ASI8223Y-A-V3

- Video Intercom
  - ◇ VTA8111A
  - ◇ VTO1210B-X, VTO1210C-X
  - ◇ VTO1220B
  - ◇ VTO2000A, VTO2111D
  - ◇ VTO6210B, VTO6100C
  - ◇ VTO9231D, VTO9241D
  - ◇ VTH1510CH, VTH1510A, VTH1550CH
  - ◇ VTH5221D, VTH5241D
  - ◇ VTS1500A, VTS5420B, VTS8240B, VTS8420B
  - ◇ VTT201, VTT2610C
- Alarm Host
  - ◇ ARC2008C, ARC2008C-G, ARC2016C, ARC2016C-G, ARC5408C, ARC5408C-C, ARC5808C, ARC5808C-C, ARC9016C, ARC9016C-G
  - ◇ DH-ARC2008C, DH-ARC2008C-G, DH-ARC2016C, DH-ARC2016C-G, DH-ARC5408C, DH-ARC5408C-C, DH-ARC5408C-E, DH-ARC5808C, DH-ARC5808C-C, DH-ARC5808C-E, DH-ARC9016C, DH-ARC9016C-G,
  - ◇ DHI-ARC2008C, DHI-ARC2008C-G, DHI-ARC2016C, DHI-ARC2016C-G, DHI-ARC5808C, DHI-ARC5808C-C, DHI-ARC5408C, DHI-ARC5408C-C, DHI-ARC9016C, DHI-ARC9016C-G,
  - ◇ ARC2008C, ARC2008C-G, ARC2016C, ARC2016C-G, ARC5408C, ARC5408C-C, ARC5408C-E, ARC5808C-C, ARC5808C, ARC5808C-E, ARC9016C, ARC9016C-G

1.3 Application Scenarios

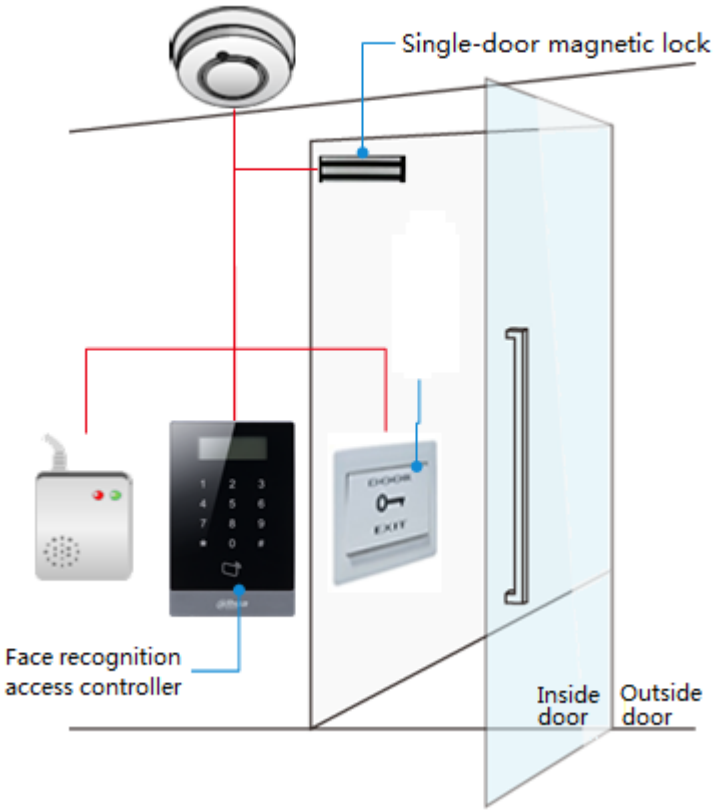
- Typical scenario.

Figure 1-1 Typical scenario



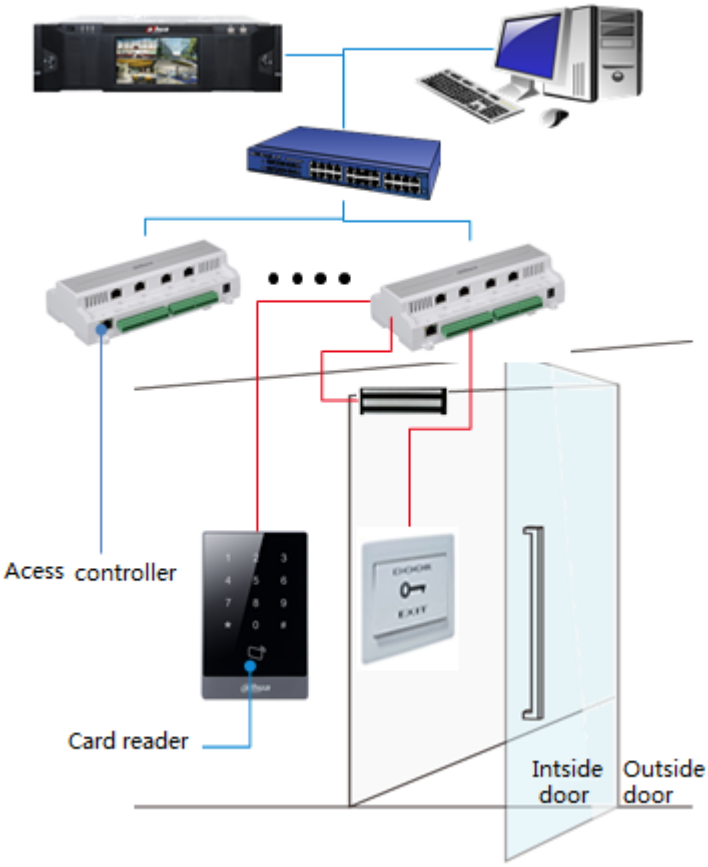
- Micro access control for small-sized office.

Figure 1-2 Micro access control



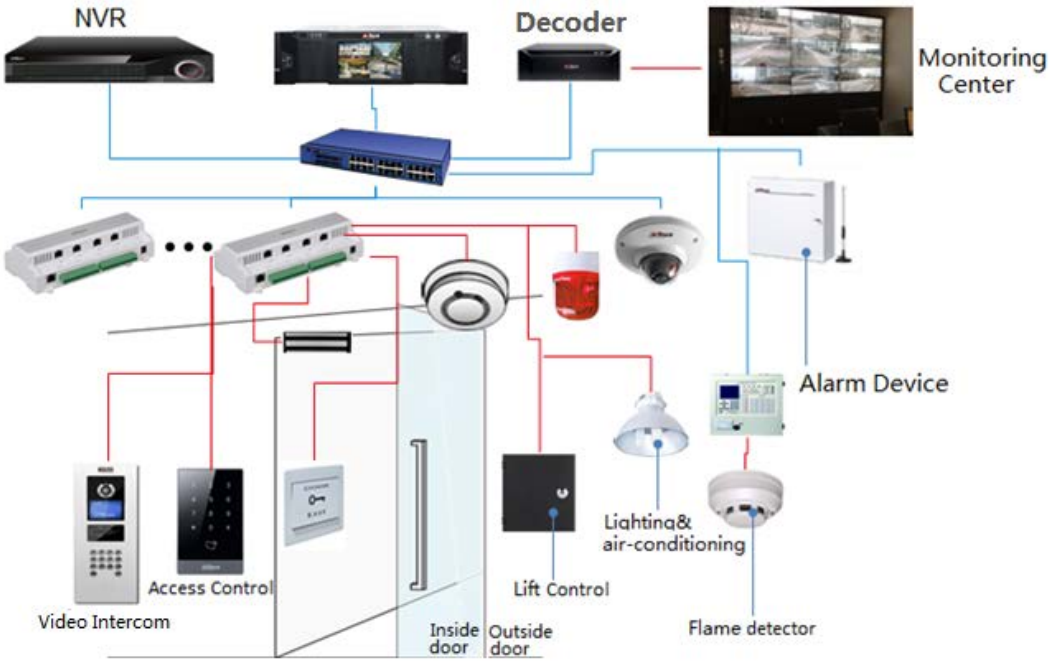
- Network access control for medium and small-sized intelligent building, treasury house and jail monitoring projects.

Figure 1-3 Network access control



- Enhanced access control.

Figure 1-4 Enhanced access control



2 Main Functions

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 resources.

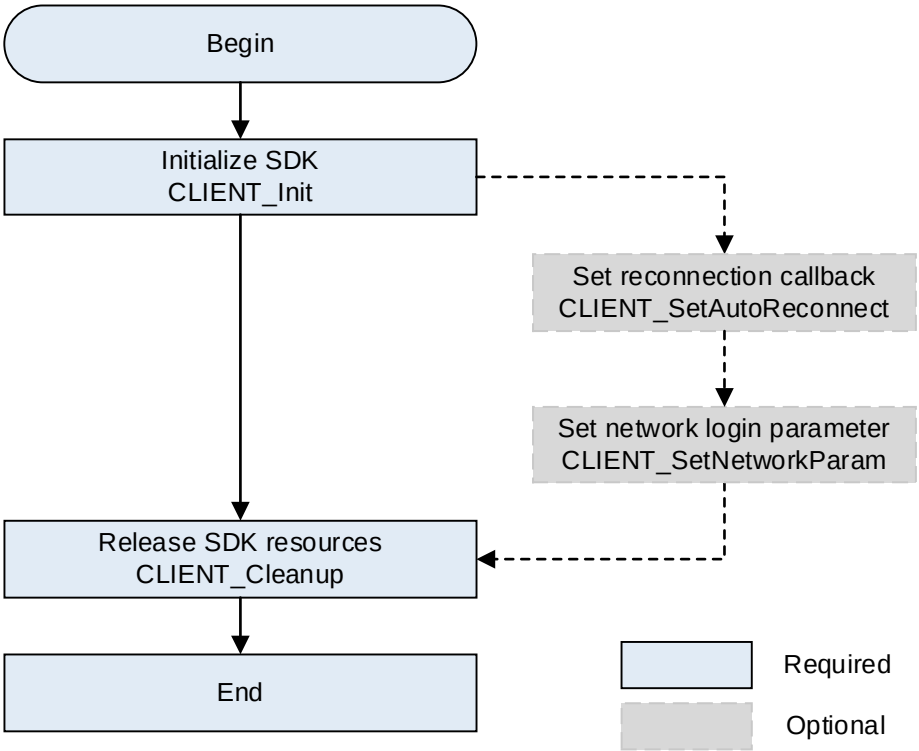
2.1.1.2 Interface Overview

Table 2-1 Description of SDK initialization interface

Interface	Description
CLIENT_Init	SDK initialization interface.
CLIENT_Cleanup	SDK cleaning up interface.
CLIENT_SetAutoReconnect	Setting of reconnection callback interface.
CLIENT_SetNetworkParam	Setting of login network environment interface.

2.1.1.3 Process Description

Figure 2-1 SDK initialization



Process

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 within SDK.

Step 3 (Optional) Call **CLIENT_SetNetworkParam** to set network login parameter that includes the timeout period for device login and the number of attempts.

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

Note

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

2.1.1.4 Example Code

```
//Set this callback through CLIENT_Init. When the device is disconnected, SDK informs the user through this callback
void CALLBACK DisConnectFunc(LLONG lLoginID, char *pchDVRIP, LONG nDVRPort, DWORD dwUser)
{
    printf("Call DisConnectFunc: lLoginID[0x%x]\n", lLoginID);
}
//Initialize SDK
CLIENT_Init(DisConnectFunc, 0);

//Call the functional interface to handle the process

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

2.1.2 Device Initialization

2.1.2.1 Introduction

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

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

2.1.2.2 Interface Overview

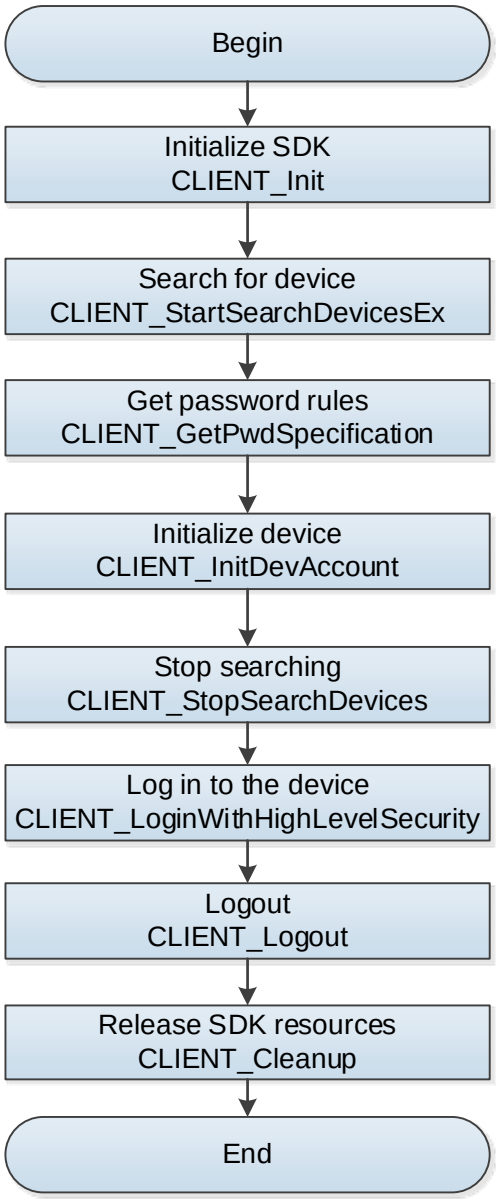
Table 2-2 Description of device initialization interfaces

Interface	Description
CLIENT_StartSearchDevicesEx	Search for devices in the LAN, and find the uninitialized devices.
CLIENT_InitDevAccount	Device 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 Description

2.1.2.3.1 Device Initialization

Figure 2-2 Device initialization



Process

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

Step 2 Call **CLIENT_StartSearchDevicesEx** to search for the devices within the LAN and get the device information (multi-thread calling is not supported).

Step 3 Call the interface **CLIENT_GetPwdSpecification** to get the password rules of the device, and confirm the password format to be set according to the rules.

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

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

Step 6 Call **CLIENT_LoginWithHighLevelSecurity** and log in to the device with the admin account and the set password.

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

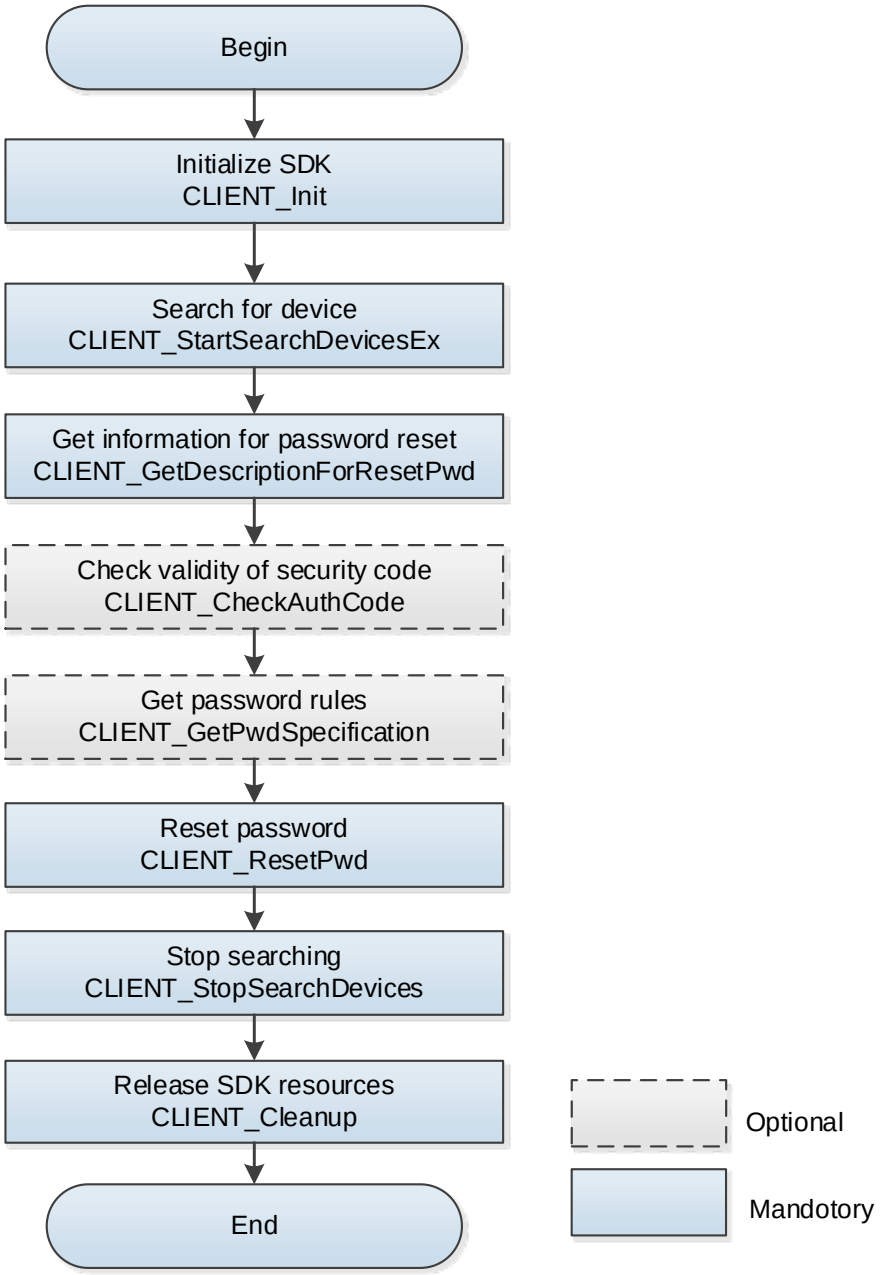
Step 8 After using all SDK functions, call **CLIENT_Cleanup** to release SDK resources.

Note

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

2.1.2.3.2 Resetting the password

Figure 2-3 Password reset and verification



Process

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

Step 2 Call **CLIENT_StartSearchDevicesEx** to search for the devices within the LAN and get the device information (multi-thread calling is not supported).

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

Step 4 (Optional) Scan the QR code obtained from the previous step in the specified way to get the security code of password reset, 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 device with the admin account and the reset password.

Step 9 After using the function module, call **CLIENT_Logou**t to log out of the device.

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

Note

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 Example Code for Device Initialization

```
//Firstly, call the interface CLIENT_StartSearchDevicesEx to get the device information in the callback.
//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 cards, enter the host PC IP for the last parameter.
Set a correct password according to the device password rules obtained, and this step is mainly to prevent
users from setting some password formats 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
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, set the szCellPhone field; if the byPwdResetWay is set as 2, set sInitAccountIn.szMail.
CLIENT_InitDevAccount(&sInitAccountIn, &sInitAccountOut, 5000, NULL);
```

2.1.2.4.2 Example Code for Password Reset

```
//Firstly, call the interface CLIENT_StartSearchDevicesEx to get the device information in the callback.
//Get the descriptive 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_StartSearchDevices, 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 cards, enter the host PC IP for the last parameter. After successful interface execution, 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
strncpy(stIn1.szSecurity, szSecu, sizeof(stIn1.szSecurity) - 1); //szSecu is the security code sent to the reserved mobile phone or email box in the previous step
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 cards, enter the host PC IP for the last parameter
//Get the password rules
NET_IN_PWD_SPECI stIn2 = {sizeof(stIn2)};
strncpy(stIn2.szMac, szMac, sizeof(stIn2.szMac) - 1); //Set mac
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 cards, enter the host PC IP for the last parameter. Set a correct password according to the device password rules successfully obtained, and this step is mainly to prevent users from setting some password formats that are not supported by the device
//Reset the 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 after scanning the QR code
```

```
stIn3.byInitStaus = bStstus; //bStstus is the value of return field byInitStatus of device search interface (callback of CLIENT_SearchDevices, CLIENT_StartSearchDevices and CLIENT_StartSearchDevicesEx, and CLIENT_SearchDevicesByIPs)
stIn3.byPwdResetWay = bPwdResetWay; //bPwdResetWay is the value of return field byPwdResetWay of device search interface (callback of CLIENT_SearchDevices and CLIENT_StartSearchDevices, CLIENT_StartSearchDevicesEx, 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 cards, 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 can obtain a unique login ID upon logging in to the device and should use the login ID before using other SDK interfaces. The login ID becomes invalid once logged out.

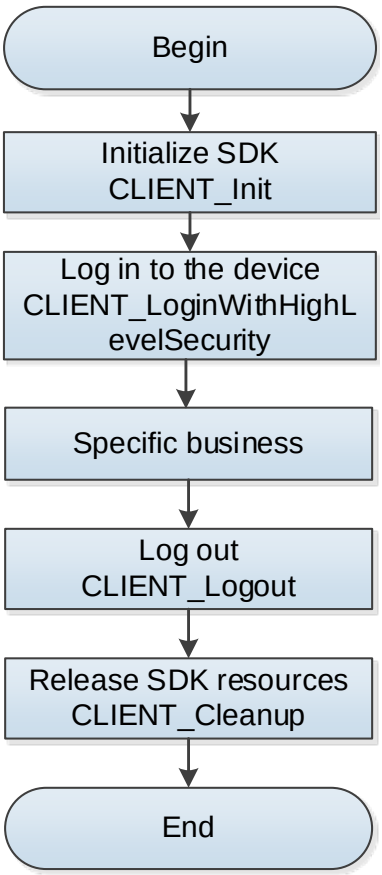
2.1.3.2 Interface Overview

Table 2-3 Description of device login interfaces

Interface	Description
CLIENT_LoginWithHighLevelSecurity	High security level login interface. You can still use CLINET_LoginEx2, but there is a security risk. Therefore, it is highly recommended to use the latest interface CLIENT_LoginWithHighLevelSecurity to log in to the device.
CLIENT_Logout	Logout interface.

2.1.3.3 Process Description

Figure 2-4 Login



Process

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 resources.

Note

- Login handle: When the login is successful, the returned value of the interface is not 0 (even the handle is smaller than 0, the login is also successful). One device can log in multiple times with different handle at each login. If there is not special function module, it is suggested to log in only one time. The login handle can be repeatedly used on other function modules.
- Logout: The interface will release the opened functions in the login session internally, but it is not suggested to rely on the cleaning up function of the logout interface. 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 releases sources once logged out.

- Login failure: It is suggested to check the failure through the error parameter (login error code) of the login interface. For the common error codes, see Table 2-4.

Table 2-4 Common error codes

Error Code	Corresponding Meaning
1	Password is wrong.
2	User name does not exist.
3	Login timeout.
4	The account has been logged in.
5	The account has been locked.
6	The account is blocklisted.
7	Out of resources, or 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 address is not authorized with login.

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

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

2.1.3.4 Example Code

```
NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY stuIn = {sizeof(stuIn)};
strncpy(stuIn.szIP, pchDVRIP, 63);
stuIn.nPort = wDVRPort;
strncpy(stuIn.szUserName, pchUserName, 63);
strncpy(stuIn.szPassword, pchPassword, 63);
stuIn.emSpecCap = EM_LOGIN_SPEC_CAP_TCP;
stuIn.pCapParam = NULL;
NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY stuOut = {sizeof(stuOut)};
// Log in to the device
LLONG lLoginHandle = CLIENT_LoginWithHighLevelSecurity(&stuIn, &stuOut);
// Log out of the device
CLIENT_Logout(lLoginHandle);
```

2.1.4 Realtime Monitor

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 logged in.

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

2.1.4.2 Interface Overview

Table 2-5 Description of real-time monitoring interfaces

Interface	Description
CLIENT_RealPlayEx	Extension interface for starting the real-time monitoring.
CLIENT_StopRealPlayEx	Extension interface for stopping the 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	Extension interface for setting the real-time monitoring data callback.

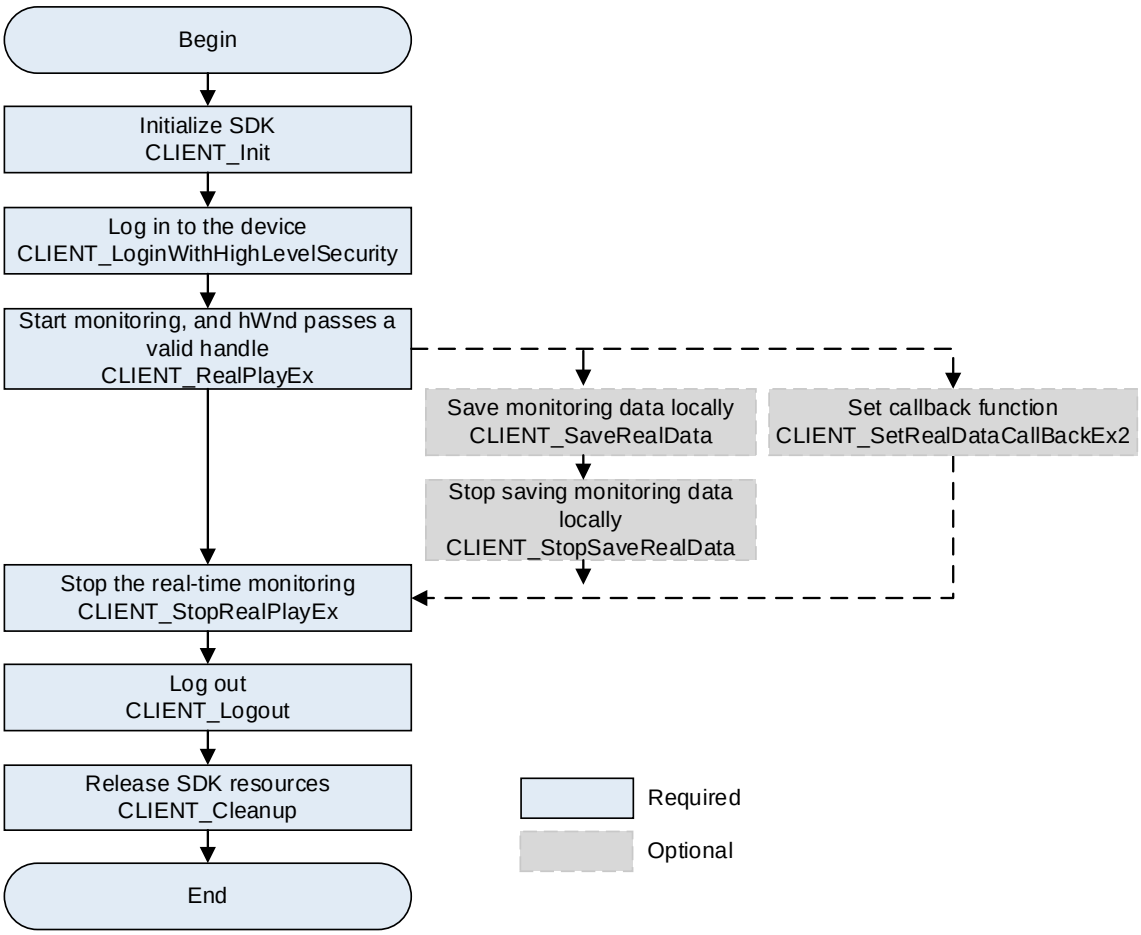
2.1.4.3 Process Description

You can realize the real-time monitoring through SDK integrated play 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 SDK decoding play



Process

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 start the real-time monitoring. The parameter **hWnd** is a valid window handle.

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

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

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

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

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 resources.

Note

- SDK decoding play only supports Windows system. You need to call the decoding after getting the stream for display 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 application for monitoring resources in the interface should make some agreements with the device before requesting the monitoring data. There are some timeout settings (see "NET_PARAM structure"), and the field related to 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 the **CLIENT_Init**.

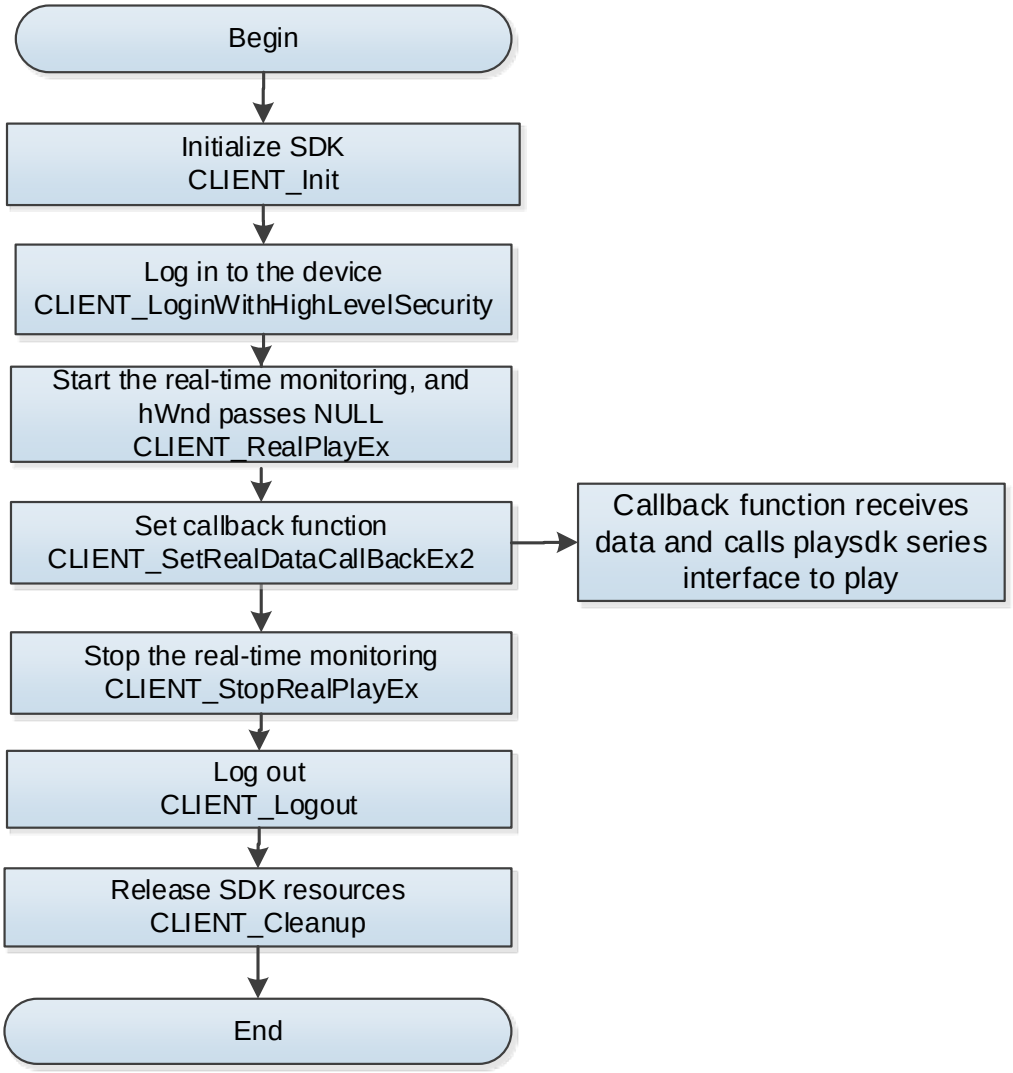
```
NET_PARAM stuNetParam = {0};
stuNetParam. nGetConnInfoTime = 5000; // in ms
CLIENT_SetNetworkParam (&stuNetParam);
```

- Failed to repeat opening: Because some devices do not support opening the monitoring function on the same channel for multiple times in one login, these devices might fail from the second opening. In this case, you can try the following:
  - ◇ Close the opened channel first. For example, if you already opened the main stream video on the channel 1 and still want to open the sub stream video on the same channel, you can close the main stream video first and then open the sub stream video.
  - ◇ Log in twice to obtain two login handles to deal with the main stream and sub stream respectively.
- Calling succeeded but no image: SDK decoding needs to use dhplay.dll. It is suggested to check if dhplay.dll and its auxiliary library are missing under the running directory. See Table 1-2 and Table 1-1.
- If the system resource is insufficient, the device might return error instead of recovering 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*.
- 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, see "2.1.4.3.2 Calling the Third-party Decoding Play Library" for details.

2.1.4.3.2 Calling the Third-party Decoding Play Library

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

Figure 2-6 Third-party decoding play



Process

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 start real-time monitoring. The parameter **hWnd** is **NULL**.

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

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

Step 6 After using the real-time monitoring, call **CLIENT_StopRealPlayEx** to stop it.

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 resources.

Note

- Stream format: It is recommended to use PlaySDK for decoding.
- Image lag:
  - ◇ 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 to such as 3 M.

◇ SDK callbacks can only move 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 window handle
LLONG lRealHandle = CLIENT_RealPlayEx(lLoginHandle, 0, hWnd, DH_RType_Realplay);
if (NULL == lRealHandle)
{
printf("CLIENT_RealPlayEx: failed! Error code: %x.\n", CLIENT_GetLastError());
}
printf("input any key to quit!\n");
getchar();
//Stop live view
if (NULL != lRealHandle)
{
CLIENT_StopRealPlayEx(lRealHandle);
}
```

2.1.4.4.2 Calling 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; //Flag of raw data
CLIENT_SetRealDataCallBackEx2(lRealHandle, &RealDataCallBackEx, NULL, dwFlag);
}

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

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

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

2.1.5 Voice Talk

2.1.5.1 Introduction

Voice talk realizes the voice interaction between the local platform and the environment where front-end devices are located, to meet the need of voice communication between the local platform and the site environment.

This chapter introduces how to use SDK to realize the voice talk with devices.

2.1.5.2 Interface Overview

Table 2-6 Description of voice talk interfaces

Interface	Description
CLIENT_StartTalkEx	Extension interface for starting the voice talk.
CLIENT_StopTalkEx	Extension interface for stopping the voice talk.
CLIENT_RecordStartEx	Extension interface for starting the client record (valid only in Windows system).
CLIENT_RecordStopEx	Extension interface for stopping the client record (valid only in Windows system).
CLIENT_TalkSendData	Send voice data to the device.
CLIENT_AudioDecEx	Extension interface for decoding audio data (valid only in Windows system).
CLIENT_SetDeviceMode	Set device voice talk mode.

2.1.5.3 Process Description

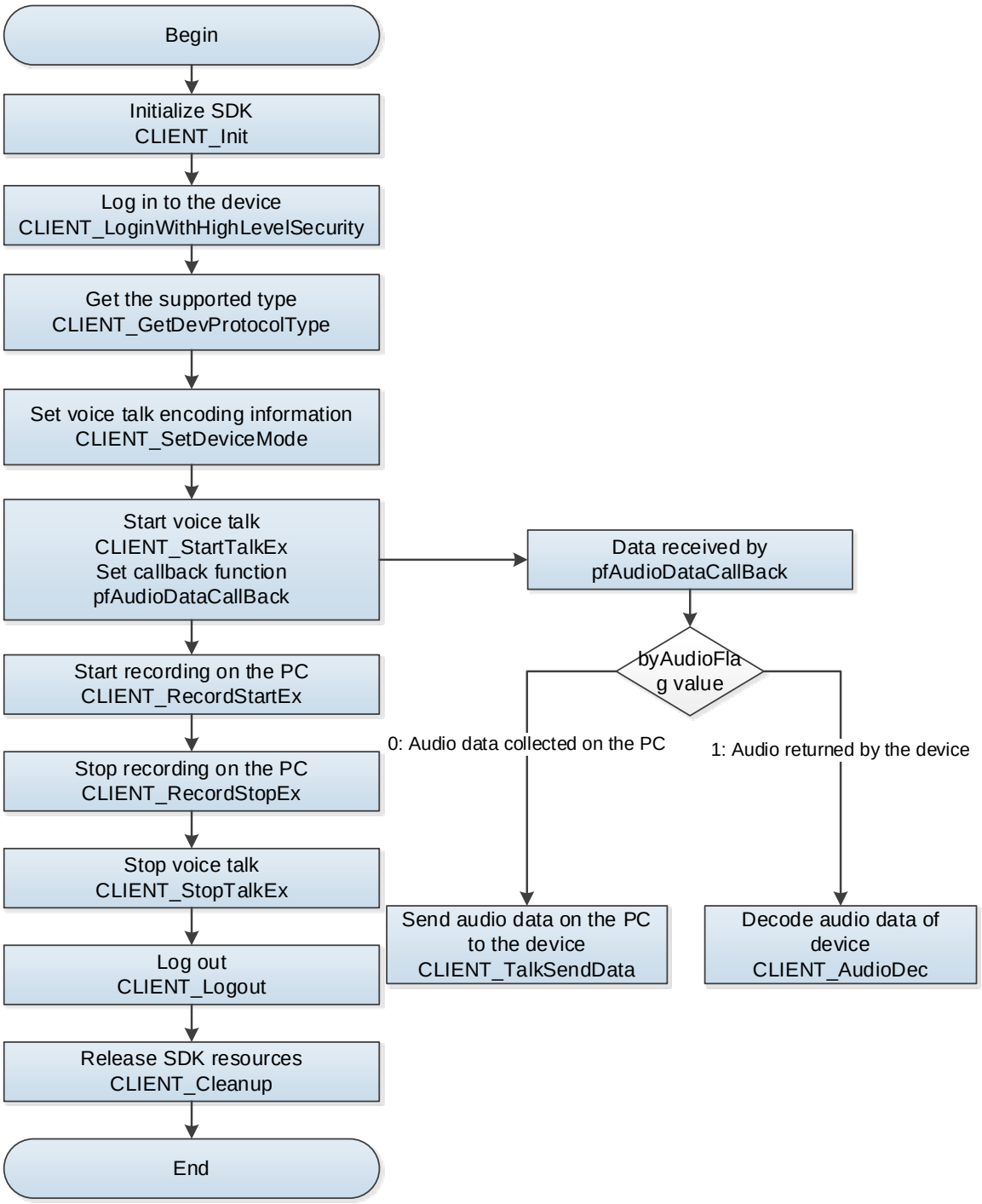
When SDK collects the audio data from the local audio card or receives the audio data from the front-end devices, it will call the audio data callback. You can call the SDK interface in the callback to

send the local audio data collected to the front-end devices, or call the SDK interface to decode and play back the audio data received from the front-end devices.

The process is valid only in Windows system.

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

Figure 2-7 Second-generation voice talk



Process

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

Step 2 After successful initialization, call **CLIENT_LoginWithHighLevelSecurity** to log in to the device.

Step 3 Call **CLIENT_GetDevProtocolType** to get support for the second-generation or third-generation voice talk.

Step 4 Call **CLIENT_SetDeviceMode** to set voice talk parameters.

- For the second-generation voice talk: Set encoding mode, client mode and speak mode. The parameter **emType** is set as **DH_TALK_ENCODE_TYPE, DH_TALK_CLIENT_MODE** and **DH_TALK_SPEAK_PARAM**.
- For the third-generation voice talk: Set encoding mode, client mode, and parameters for third-generation voice talk. The parameter **emType** is set as **DH_TALK_ENCODE_TYPE, DH_TALK_CLIENT_MODE** and **DH_TALK_MODE3**.

Step 5 Call **CLIENT_StartTalkEx** to set callback and start voice talk. In the callback, call **CLIENT_AudioDec** to decode the audio data sent from the decoding device, and call **CLIENT_TalkSendData** to send the audio data from the PC to the device.

Step 6 Call **CLIENT_RecordStartEx** to start recording on the PC. After this interface is called, the voice talk callback set by **CLIENT_StartTalkEx** will receive the local audio data.

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

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

Step 9 Call **CLIENT_Logout** to log out of the device.

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

Note

- Voice encoding format: The example uses the common PCM format. SDK supports getting the voice encoding format supported by the device. The example code is detailed in the SDK package on the website. If the default PCM can meet the requirement, it is not necessary to get the voice encoding format supported by the device.
- No sound at the device: The audio data needs to be collected from devices such as microphone. It is recommended to check if the microphone or other equivalent device is plugged in and if the interface **CLIENT_RecordStartEx** succeeded in returning.

2.1.5.4 Example Code

```
//Get to know whether the device supports the second-generation or third-generation voice talk.
EM_DEV_PROTOCOL_TYPE emTpye = EM_DEV_PROTOCOL_UNKNOWN;
CLIENT_GetDevProtocolType(g_lLoginHandle, &emTpye);

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

```
//Set voice talk parameters according to the obtained type
if (emTpye == EM_DEV_PROTOCOL_V3) //Only the third-generation voice talk needs such parameters
{
NET_TALK_EX stuTalk = {sizeof(stuTalk)};
        stuTalk.nAudioPort = RECEIVER_AUDIO_PORT;    //User-defined 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 recording
CLIENT_RecordStartEx(lLoginHandle);
//Stop local recording
CLIENT_RecordStopEx(lLoginHandle)
//Stop voice talk
CLIENT_StopTalkEx(lTalkHandle);
//Process the voice talk callback data
void CALLBACK AudioDataCallBack(LLONG lTalkHandle, char *pDataBuf, DWORD dwBufSize, BYTE byAudioFlag,
LDWORD dwUser)
{
if(0 == byAudioFlag)
{
    //Send the audio card data detected by the local PC to the device
    CLIENT_TalkSendData(lTalkHandle, pDataBuf, dwBufSize);
  }
else if(1 == byAudioFlag)
  {
    //Pass the audio data sent from the device to SDK for decoding play
    CLIENT_AudioDec(pDataBuf, dwBufSize);
  }
}
```

2.1.6 Event Listening

2.1.6.1 Introduction

Alarm reporting method: Use SDK to log in to the device and subscribe to the alarm function from the device. When the device detects the alarm event, it will send the event to SDK immediately. The user can get the corresponding alarm information through the alarm callback.

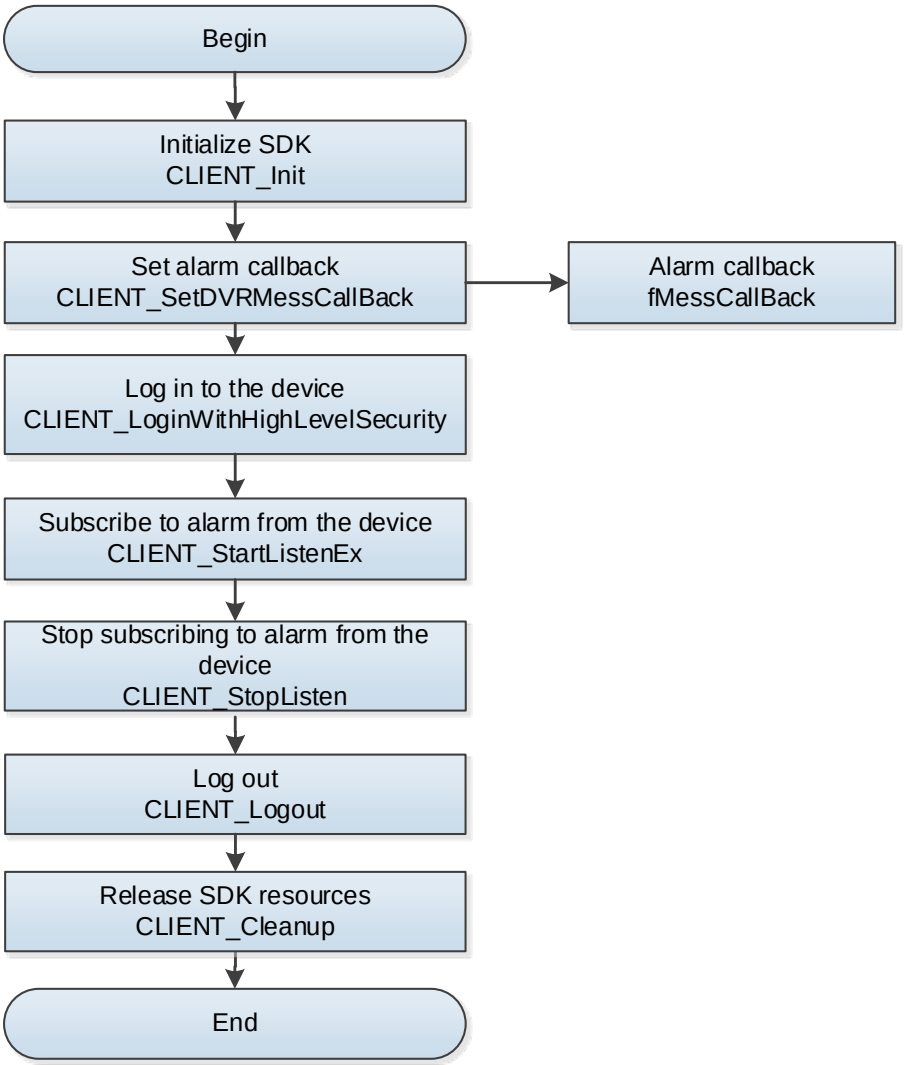
2.1.6.2 Interface Overview

Table 2-7 Description of alarm listening and reporting interfaces

Interface	Description
CLIENT_SetDVRMessCallBack	Set alarm callback.
CLIENT_StartListenEx	Extension interface for subscribing to alarm.
CLIENT_StopListen	Stop subscribing to alarm.

2.1.6.3 Process Description

Figure 2-8 Alarm reporting



Process

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

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

The interface needs to be called before subscribing to alarm.

Step 3 Call the **CLIENT_LoginWithHighLevelSecurity** to log in to the device.

Step 4 Call the **CLIENT_StartListenEx** to subscribe to alarm from the device. After successful subscription, use the callback set by **CLIENT_SetDVRMessCallBack** to inform the user of the alarm events reported by the device.

For alarm events related to alarm host, access control and voice talk, see "4.7 Alarm Callback fMessCallBack."

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

Step 6 Call the **CLIENT_Logout** to log out of the device.

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

Note

- If the alarms that were reported before are no longer reported, check if the device is disconnected. If yes, please be noted that there will be no alarm reported after the device is reconnected, and in this case, you need to cancel the subscription and subscribe to alarm again.
- It is recommended to process the alarm information in the callback fMessCallBack in somewhere else to avoid blocking the callback operations.

2.1.6.4 Example Code

```
//Alarm callback
int CALLBACK afMessCallBack(LONG lCommand, LLONG lLinID, char *pBuf, DWORD dwBufLen,
char *pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
if(lCommand == DH_ALARM_ACCESS_CTL_EVENT) // Access control event. For more events related to access
control, see "4.7 Alarm Callback fMessCallBack."
{
ALARM_ACCESS_CTL_EVENT_INFO* pstAccessInfo =
  (ALARM_ACCESS_CTL_EVENT_INFO*)pBuf;
          //Then you can get the corresponding alarm information through pstAccessInfo.
}
……..
}
//Set alarm callback
CLIENT_SetDVRMessCallBack(afMessCallBack,0);
//Subscribe to alarm
CLIENT_StartListenEx(lLoginHandle);
//Stop subscribing to alarm
CLIENT_StopListen(lLoginHandle);
```

2.1.7 Subscribing to Intelligent Event

2.1.7.1 Introduction

Intelligent event subscribe, is that the front-end devices or the back-end devices do the real-time stream analyzing. When detect the preset intelligent event, it uploads the event to the user. The intelligent events in this manual contain general action analysis (such as tripwire, Intrusion), target detection, object recognition, body detection, the intelligent events of intelligent traffic (such as traffic junction, over speed, low speed and traffic jam).

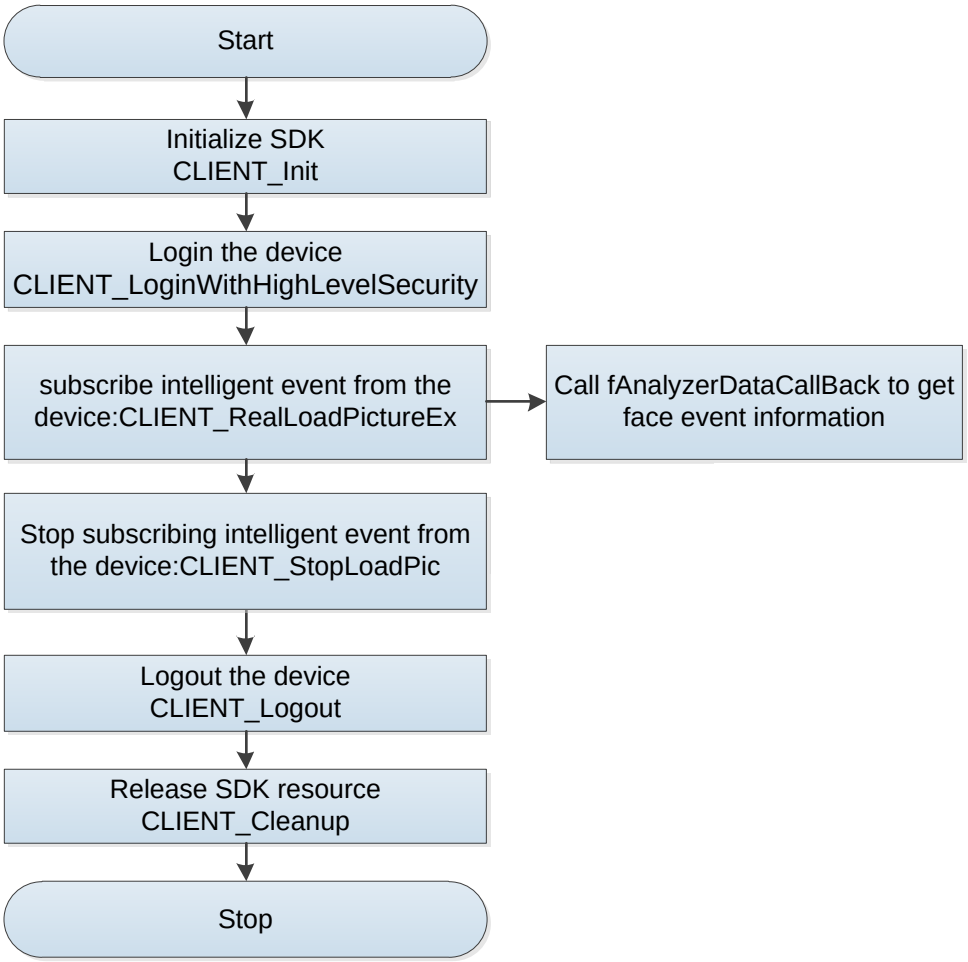
2.1.7.2 Interface Overview

Table 2-8 Interfaces of subscribing to intelligent event

Interface	Implication
CLIENT_RealLoadPictureEx	Subscribe intelligent event.
CLIENT_StopLoadPic	Cancel subscribing the intelligent event.

2.1.7.3 Process

Figure 2-9 Process of uploading face event



Process Description

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

Step 1 Call CLIENT_LoginWithHighLevelSecurity to login the device.

Step 2 Call **CLIENT_RealLoadPictureEx** to subscribe intelligent event from the device.

Step 3 After successful subscribe, call fAnalyzerDataCallBack to upload the intelligent events. Through this function, you can filter out the intelligent events you need.

Step 4 After using the intelligent event function, call **CLIENT_StopLoadPic** to stop subscribing intelligent events.

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

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

Notes for Process

- Support to subscribe single intelligent event and all the intelligent events (EVENT_IVS_ALL).
- 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.
- Set whether to receive picture or not: You can call CLIENT_RealLoadPictureEx to set bNeedPicFile as False, and then SDK will only receive the face event without picture.

2.1.7.4 Example Code

```
// Intelligent event uploading callback function
int CALLBACK AnalyzerDataCallBack(LLONG lAnalyzerHandle, DWORD dwAlarmType, void* pAlarmInfo, BYTE *pBuffer, DWORD dwBufSize, LDWORD dwUser, int nSequence, void *reserved)
{
    switch(dwAlarmType)
    {
    // Filter out the right intelligent events
    ............
    case EVENT_IVS_ACCESS_CTL:
    ............
    default:
    break;
    }
}

// Subscribe the uploading of the intelligent 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;
}

// Cancel Subscribing the uploading of the intelligent event
CLIENT_StopLoadPic(lAnalyerHandle);
```

2.2 Alarm host

2.2.1 Arming and Disarming

2.2.1.1 Introduction

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

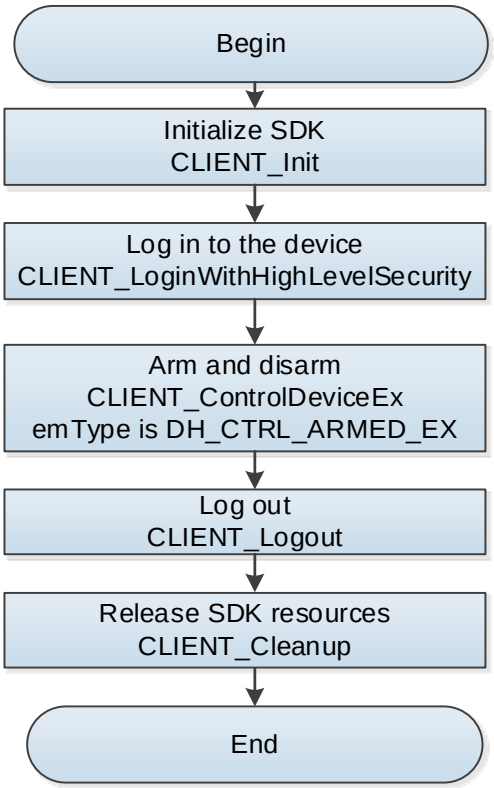
2.2.1.2 Interface Overview

Table 2-9 Description of arming and disarming interfaces

Interface	Description
CLIENT_ControlDeviceEx	Device control extension interface.

2.2.1.3 Process Description

Figure 2-10 Arming and disarming



Process

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

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

Step 3 Call the **CLIENT_ControlDeviceEx** to arm or disarm the device. The parameter **emType** value is **DH_CTRL_ARMED_EX**.

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.2.1.4 Example Code

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

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

2.2.2 Protection Zone Status Setting

2.2.2.1 Introduction

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

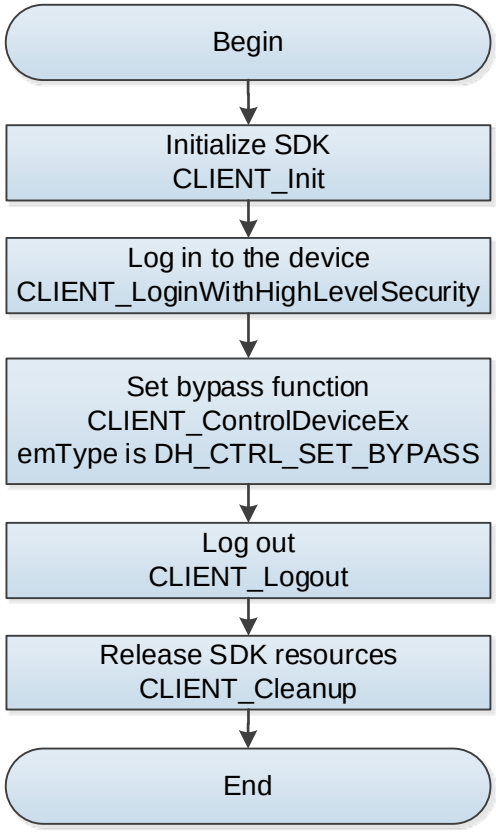
2.2.2.2 Interface Overview

Table 2-10 Description of interfaces for setting protection zone status

Interface	Description
CLIENT_ControlDeviceEx	Device control extension interface.

2.2.2.3 Process Description

Figure 2-11 Protection zone status setting



Process

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

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

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

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.2.2.4 Example Code

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

```
CLIENT_ControlDeviceEx(g_lLoginHandle, DH_CTRL_SET_BYPASS, &stuParam, NULL, 3000);
```

2.2.3 Protection Zone Status Query

2.2.3.1 Introduction

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

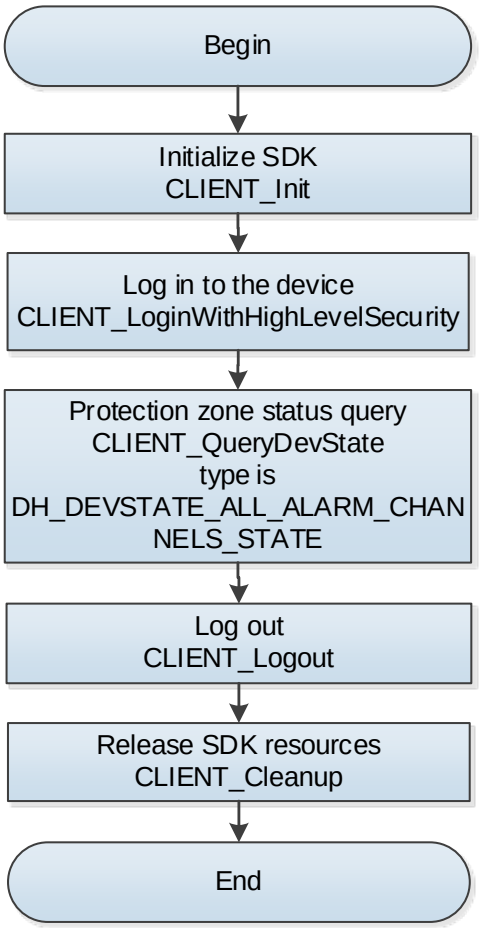
2.2.3.2 Interface Overview

Table 2-11 Description of protection zone status query interface

Interface	Description
CLIENT_QueryDevState	Status query interface.

2.2.3.3 Process Description

Figure 2-12 Protection zone status query



Process

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

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

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

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

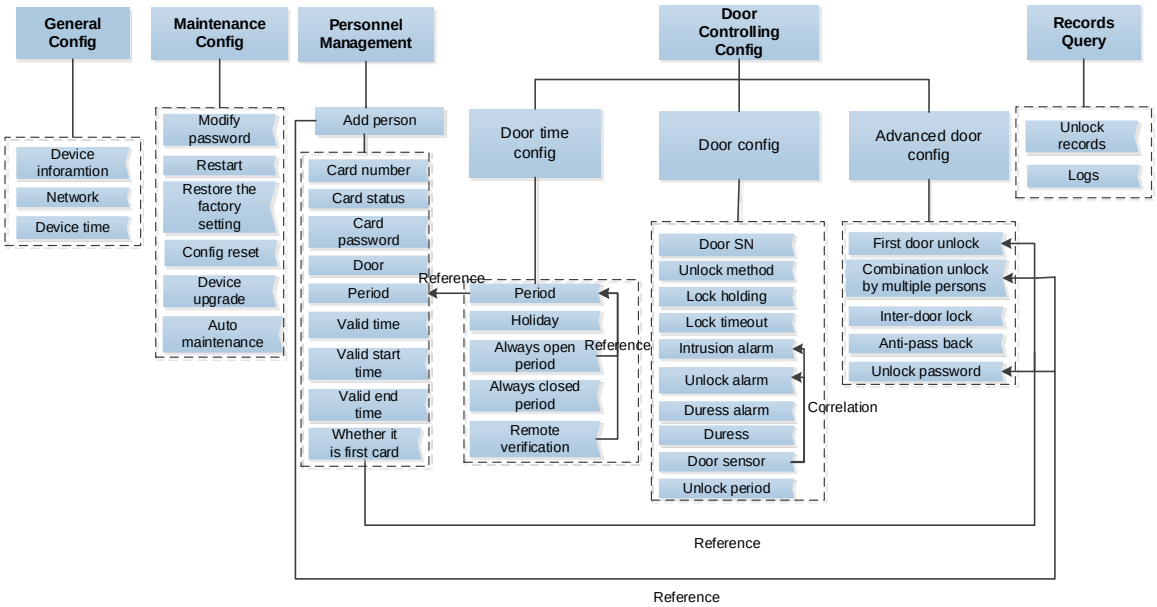
2.2.3.4 Example Code

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

//Initialize the fields related to the alarm signal channel
stuAlarmChannelState.nAlarmBellCount = nNum;
stuAlarmChannelState.pbAlarmBellState = new BOOL[stuAlarmChannelState.nAlarmBellCount];
memset(stuAlarmChannelState.pbAlarmBellState, 0, stuAlarmChannelState.nAlarmBellCount * sizeof(BOOL));

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

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

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

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

int nRetLen = 0;
```

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

2.3 Access Controller/All-in-one Fingerprint Machine (First-generation)

Figure 2-13 Function calling relationship



Here are the meanings of reference and correlation.

- Reference: The function pointed by the end point of the arrow refers to the function pointed by the start point of the arrow.
- Correlation: Whether the function started by the arrow can be used normally is related to the function configuration pointed by the end point of the arrow.

2.3.1 Access Control

2.3.1.1 Introduction

It is used to control the opening and closing of the access, and get door sensor status. Without personnel information, it can remotely open and close the door directly.

2.3.1.2 Interface Overview

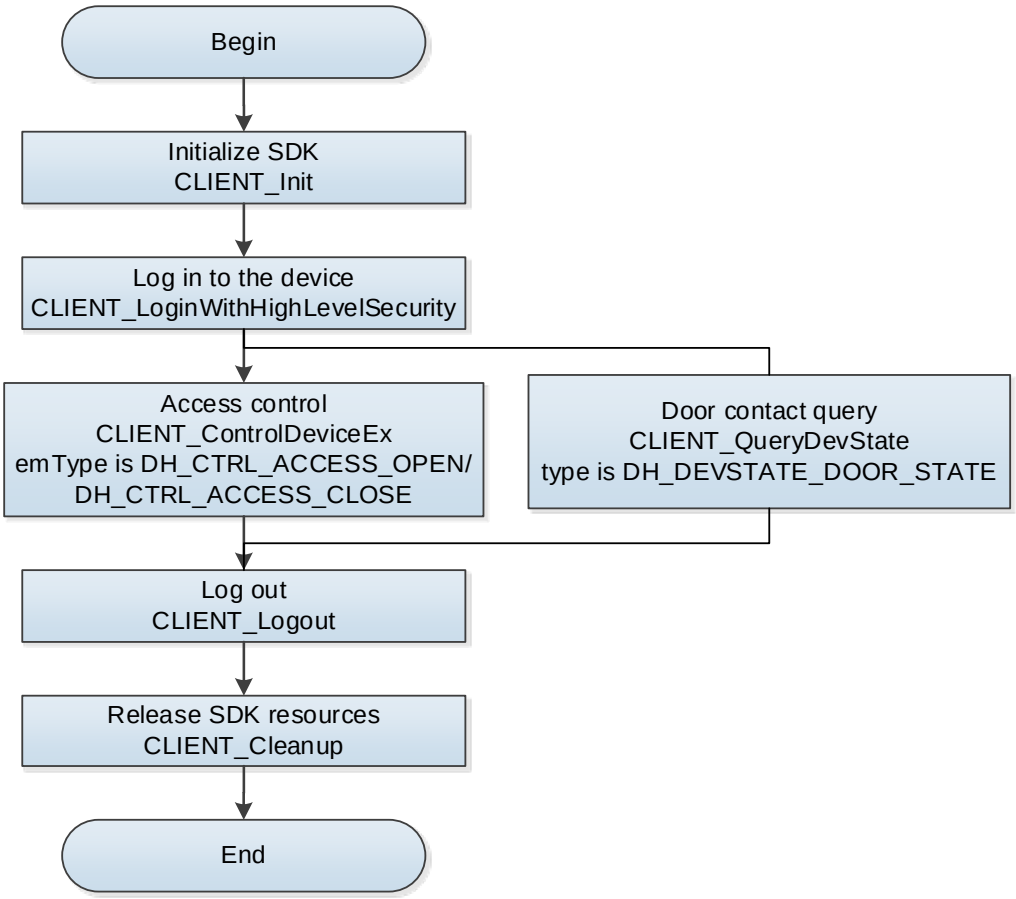
Table 2-12 Description of access control interface

Interface	Description
CLIENT_ControlDeviceEx	Device control extension interface.

CLIENT_QueryDevState	Status query interface.

2.3.1.3 Process Description

Figure 2-14 Access control



Process

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

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

Step 3 Call the **CLIENT_ControlDeviceEx** to control the access.

- Open the access: The **emType** value is **DH_CTRL_ACCESS_OPEN**.
- Close the access: The **emType** value is **DH_CTRL_ACCESS_CLOSE**.

Step 4 Call **CLIENT_QueryDevState** to query the door sensor.

Step 5 Type: DH_DEVSTATE_DOOR_STATE

Step 6 pBuf: NET_DOOR_STATUS_INFO.

Step 7 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.1.4 Example Code

```
//Open the access
NET_CTRL_ACCESS_OPEN stOpen = {sizeof(stOpen)};
stOpen.nChannelID = 0;
```

```
strncpy(stOpen.szUserID, “admin”, sizeof(stOpen.szUserID) - 1);
CLIENT_ControlDeviceEx((LLONG)g_lLoginHandle, DH_CTRL_ACCESS_OPEN, &stOpen, NULL, 3000);

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

//Query information on door sensor status
int nRet = 0;
    NET_DOOR_STATUS_INFO stuInfo = {sizeof(stuInfo)};
    stuInfo.nChannel = 0;
    BOOL bReturn = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DOOR_STATE, (char *)&stuInfo,
sizeof(stuInfo), &nRet, 5000);
    if (bReturn)
    {
        printf("door sensor status: %d\n",stuInfo.emStateType);
    }
    else{
        printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.2 Alarm Event

2.3.2.1 Introduction

The process to get event is that, you call the SDK interface. SDK actively connect to the device, and subscribe to alarm from the device, including door opening event and alarm event. Device sends events to the SDK immediately when events generate. Stop susbcribtion if you want to stop receiving events from device.

2.3.2.2 Interface Overview

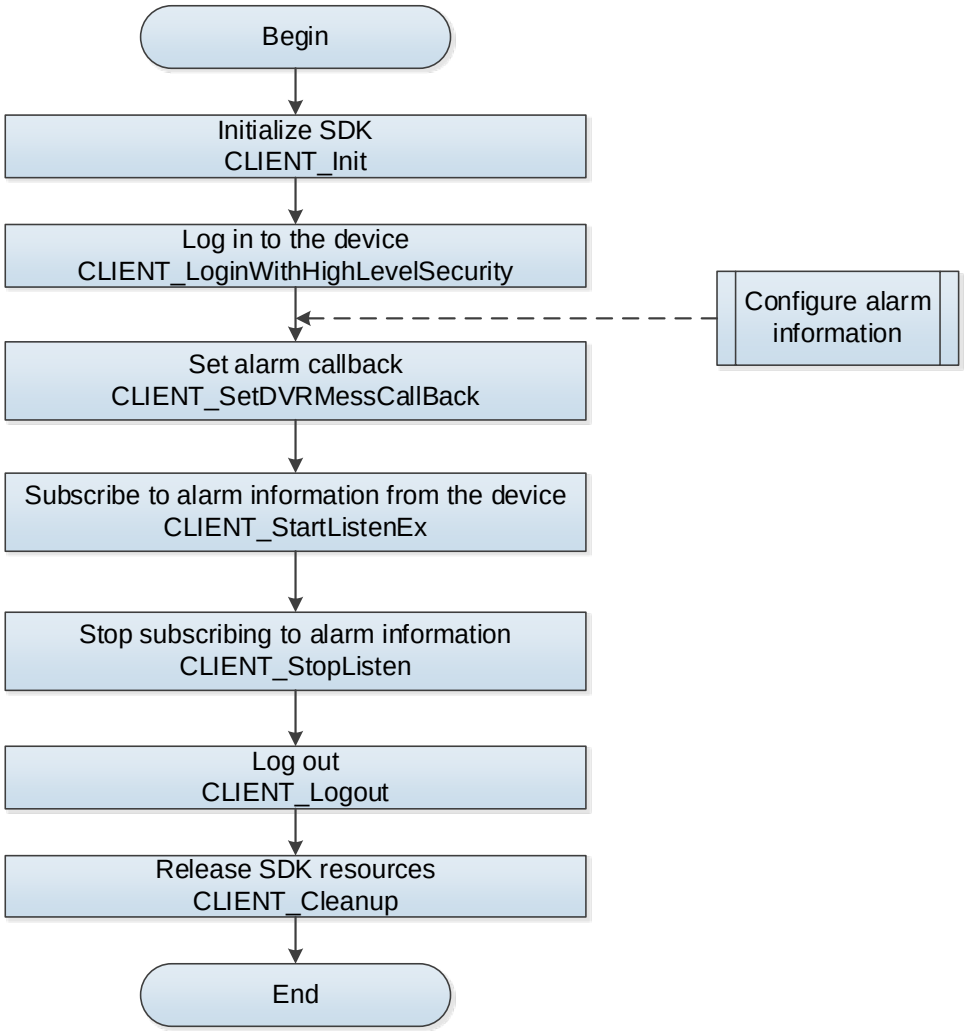
Table 2-13 Description of alarm event interface

Interface	Description
CLIENT_StartListenEx	Subscribe to alarm from the device.
CLIENT_SetDVRMessCallBack	Set device message callback to get the current device status information; this function is independent of the calling sequence, and the SDK is not called back by default. The callback must call the alarm message subscription interface CLIENT_StartListen or CLIENT_StartListenEx first before it takes effect.

CLIENT_StopListen	Stop subscription.

2.3.2.3 Process Description

Figure 2-15 Alarm event



Process

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

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

Step 3 Set alarm arming config (you can ignore this if the alarm arming has been configured).

Step 4 Set the alarm callback CLIENT_SetDVRMessCallBack.

Step 5 Call the **CLIENT_StartListenEx** to subscribe to alarm information from the device.

Step 6 After the alarm reporting process ends, you need to stop the interface for subscribing to alarm **CLIENT_StopListen**.

Step 7 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.2.4 Example Code

```
BOOL CALLBACK MessCallBack(LONG lCommand, LLONG lLoginID, char *pBuf, DWORD dwBufLen, char
*pchDVRIP, LONG nDVRPort, LDWORD dwUser)
{
    //Dismantlement prevention for device/card reader
    if (DH_ALARM_CHASSISINTRUDED == lCommand)
    {
        ALARM_CHASSISINTRUDED_INFO* pstAlarm = (ALARM_CHASSISINTRUDED_INFO*)pBuf;
        printf("Chassis intrusion\n");
        printf("nAction:%d\n", pstAlarm->nAction);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    //External alarm event
    else if (DH_ALARM_ALARM_EX2 == lCommand)
    {
        ALARM_ALARM_INFO_EX2* pstAlarm = (ALARM_ALARM_INFO_EX2*)pBuf;
        printf("LocalAlarm\n");
        printf("nAction:%d\n", pstAlarm->nAction);
        printf("ChannelID:%d,SenseType:%d\n", pstAlarm->nChannelID, pstAlarm->emSenseType);
        printf("DefenceAreaType:%d\n", pstAlarm->emDefenceAreaType);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    //Door timeout event
    else if (DH_ALARM_ACCESS_CTL_NOT_CLOSE == lCommand)
    {
        ALARM_ACCESS_CTL_NOT_CLOSE_INFO* pstAlarm =
(ALARM_ACCESS_CTL_NOT_CLOSE_INFO*)pBuf;
        printf("DoorNotClosed\n");
        printf("nAction:%d\n", pstAlarm->nAction);
        printf("DoorNO.:%d,EventID:%d\n", pstAlarm->nDoor, pstAlarm->nEventID);
        printf("DoorName:%s\n", pstAlarm->szDoorName);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
```

```
    //Intrusion event
    else if (DH_ALARM_ACCESS_CTL_BREAK_IN == lCommand)
    {
        ALARM_ACCESS_CTL_BREAK_IN_INFO* pstAlarm = (ALARM_ACCESS_CTL_BREAK_IN_INFO*)pBuf;
        printf("BreakIn\n");
        printf("DoorNO.:%d\n", pstAlarm->nDoor);
        printf("BreakMethod:%d,EventID:%d\n", pstAlarm->emMethod, pstAlarm->nEventID);
        printf("DoorName:%s\n", pstAlarm->szDoorName);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    //Forced event
    else if (DH_ALARM_ACCESS_CTL_DURESS == lCommand)
    {
        ALARM_ACCESS_CTL_DURESS_INFO* pstAlarm = (ALARM_ACCESS_CTL_DURESS_INFO*)pBuf;
        printf("Duress\n");
        printf("DoorNO.:%d\n", pstAlarm->nDoor);
        printf("CardNo:%d,EventID:%d\n", pstAlarm->szCardNo, pstAlarm->nEventID);
        printf("DoorName:%s,SN:%s,UserID:%s\n", pstAlarm->szDoorName, pstAlarm->szSN,
pstAlarm->szUserID);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    //Passback event
    else if (DH_ALARM_ACCESS_CTL_REPEAT_ENTER == lCommand)
    {
        ALARM_ACCESS_CTL_REPEAT_ENTER_INFO* pstAlarm =
(ALARM_ACCESS_CTL_REPEAT_ENTER_INFO*)pBuf;
        printf("Duress\n");
        printf("DoorNO.:%d\n", pstAlarm->nDoor);
        printf("CardNo:%d,EventID:%d\n", pstAlarm->szCardNo, pstAlarm->nEventID);
        printf("DoorName:%s\n", pstAlarm->szDoorName);
        printf("%d.%d.%d %d:%d:%d:%d\n",
pstAlarm->stuTime.dwYear,pstAlarm->stuTime.dwMonth,pstAlarm->stuTime.dwDay,
            pstAlarm->stuTime.dwHour,pstAlarm->stuTime.dwMinute,pstAlarm->stuTime.dwSecond);
    }
    return TRUE;
}
```

```
void StartListen(LLONG g_lLoginHandle)
{
    CLIENT_SetDVRMessCallBack(MessCallBack, NULL);
    BOOL bRet = CLIENT_StartListenEx(g_lLoginHandle);

    if (bRet)
    {
        printf("CLIENT_StartListenEx success!\n");
    }
    else
    {
        printf("CLIENT_StartListenEx failed! LastError = %x\n" , CLIENT_GetLastError());
    }
}
```

2.3.3 Viewing Device Information

2.3.3.1 Capability Set Query

2.3.3.1.1 Introduction

The process to view device information is that, you issue a command through SDK to the access control device, to get the capability of another device.

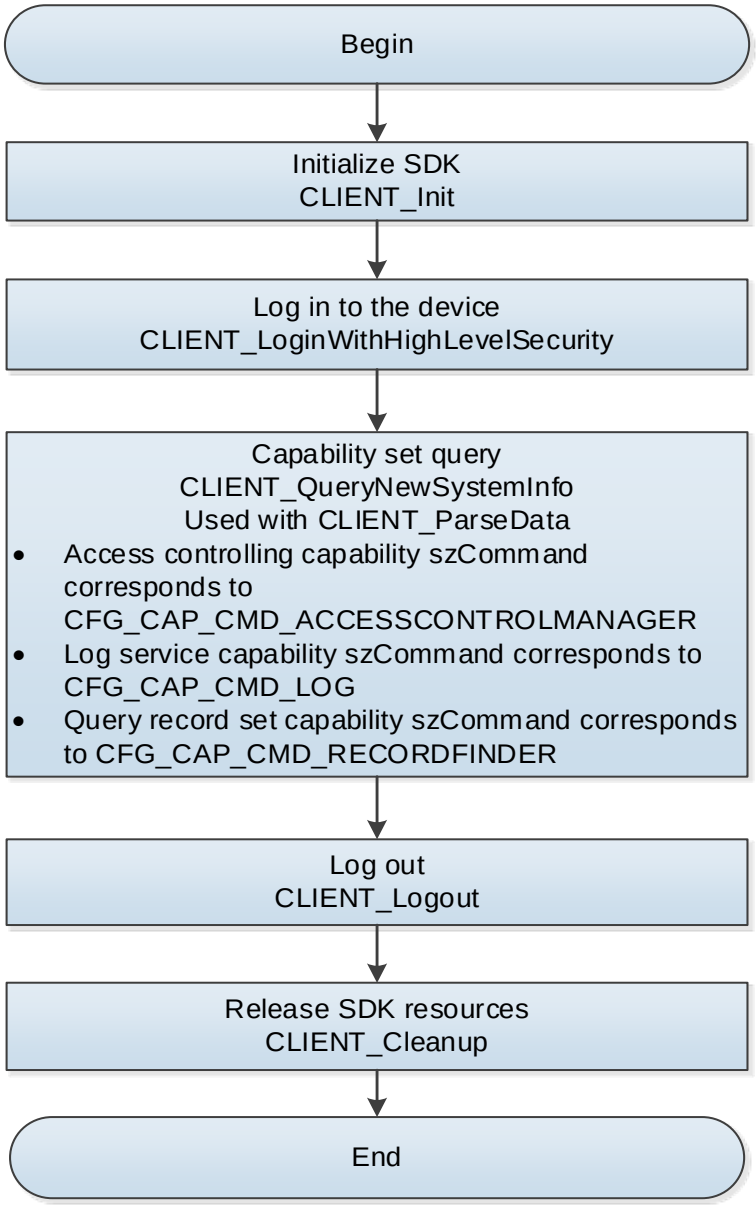
2.3.3.1.2 Interface Overview

Table 2-14 Description of capability set query interface

Interface	Description
CLIENT_QueryNewSystemInfo	Query information on system capabilities (sucha as logs, record sets, and door control capabilities).
CLIENT_ParseData	Parse the queried config information.

2.3.3.1.3 Process Description

Figure 2-16 Device information viewing



Process

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

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

Step 3 Call **CLIENT_QueryNewSystemInfo** and **CLIENT_ParseData** to query access control capability set.

Table 2-15 Description and structure of szCommand

szCommand	Description	szOutBuffer
CFG_CAP_CMD_ACCESSCONTROLMANAGER	Access controlling capability	CFG_CAP_ACCESSCONTROL
CFG_CAP_CMD_LOG	Log getting capability	CFG_CAP_LOG
CFG_CAP_CMD_RECORDFINDER	Query record set capability	CFG_CAP_RECORDFINDER_INFO

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.3.1.4 Example Code

```
//Capability set query
char szBuf[1024] = {0};
int nError = 0;
BOOL bRet = CLIENT_QueryNewSystemInfo(m_lLoginID, CFG_CAP_CMD_ACCESSCONTROLMANAGER, -1,
szBuf, sizeof(szBuf), &nError, 3000);
if (bRet)
{
    CFG_CAP_ACCESSCONTROL stuCap = {0};
    DWORD dwRet = 0;
    bRet = CLIENT_ParseData(CFG_CAP_CMD_ACCESSCONTROLMANAGER, szBuf, &stuCap, sizeof(stuCap),
&dwRet);
    if (bRet && dwRet == sizeof(CFG_CAP_ACCESSCONTROL))
    {
      int nCount = stuCap.nAccessControlGroups;
    }
    else
    {
      return FALSE;
    }
}
```

2.3.3.2 Viewing Device Version and MAC

2.3.3.2.1 Introduction

The process to view device version and MAC is that, you issue a command through SDK to the access control device, to get device information such as serial number, version number and Mac address.

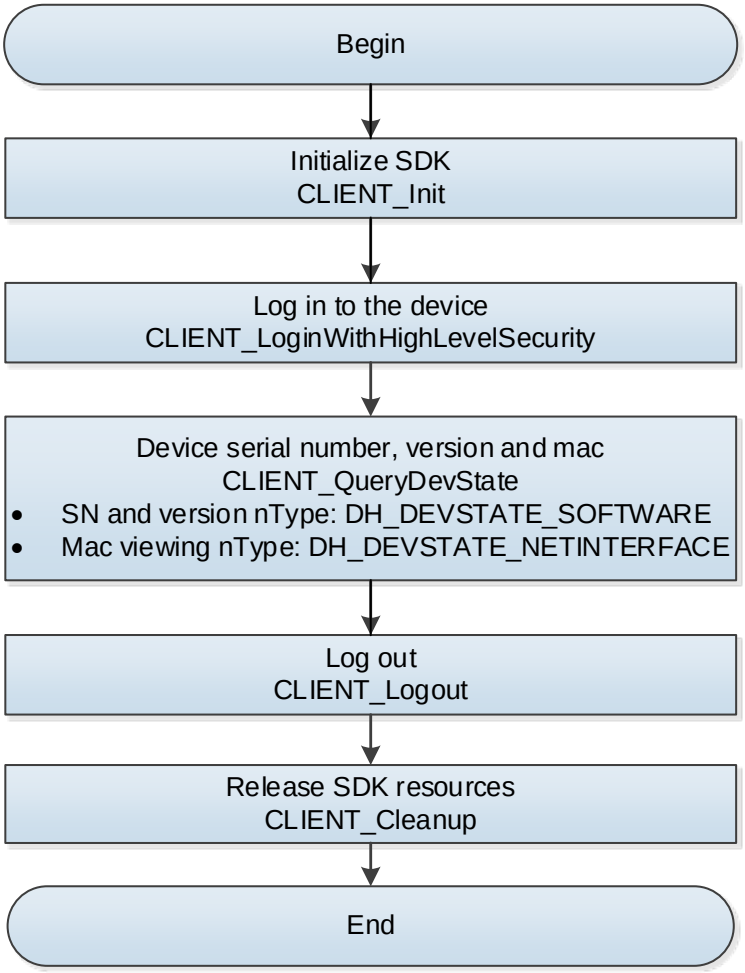
2.3.3.2.2 Interface Overview

Table 2-16 Description of interfaces for viewing device version and MAC

Interface	Description
CLIENT_QueryDevState	Query device status (query serial number, software version, compiling time, Mac address).

2.3.3.2.3 Process Description

Figure 2-17 Device information viewing



Process

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

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

Step 3 Call the **CLIENT_QueryDevState** to query access control device information such as serial number, version and mac.

Table 2-17 Description and structure of nType

nType	Description	pBuf
DH_DEVSTATE_SOFTWARE	Serial number and version	DHDEV_VERSION_INFO
DH_DEVSTATE_NETINTERFACE	Mac address	DHDEV_NETINTERFACE_INFO

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.3.2.4 Example Code

```
//Query the serial number of the device
int nRet = 0;
DHDEV_VERSION_INFO stuVersion = {sizeof(stuVersion)};
```

```
BOOL bRet = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_SOFTWARE, (char *)&stuVersion, sizeof(stuVersion), &nRet, 5000);
//View Mac
int nRet = 0;
DHDEV_NETINTERFACE_INFO stuNet = {sizeof(stuNet)};
BOOL bRet0 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_NETINTERFACE, (char *)&stuNet, sizeof(stuNet), &nRet, 5000);
```

2.3.4 Network Setting

2.3.4.1 IP Settings

2.3.4.1.1 Introduction

IP setting process is that, you call SDK interface to get and configure device information such as IP, including IP address, subnet mask, and default gateway.

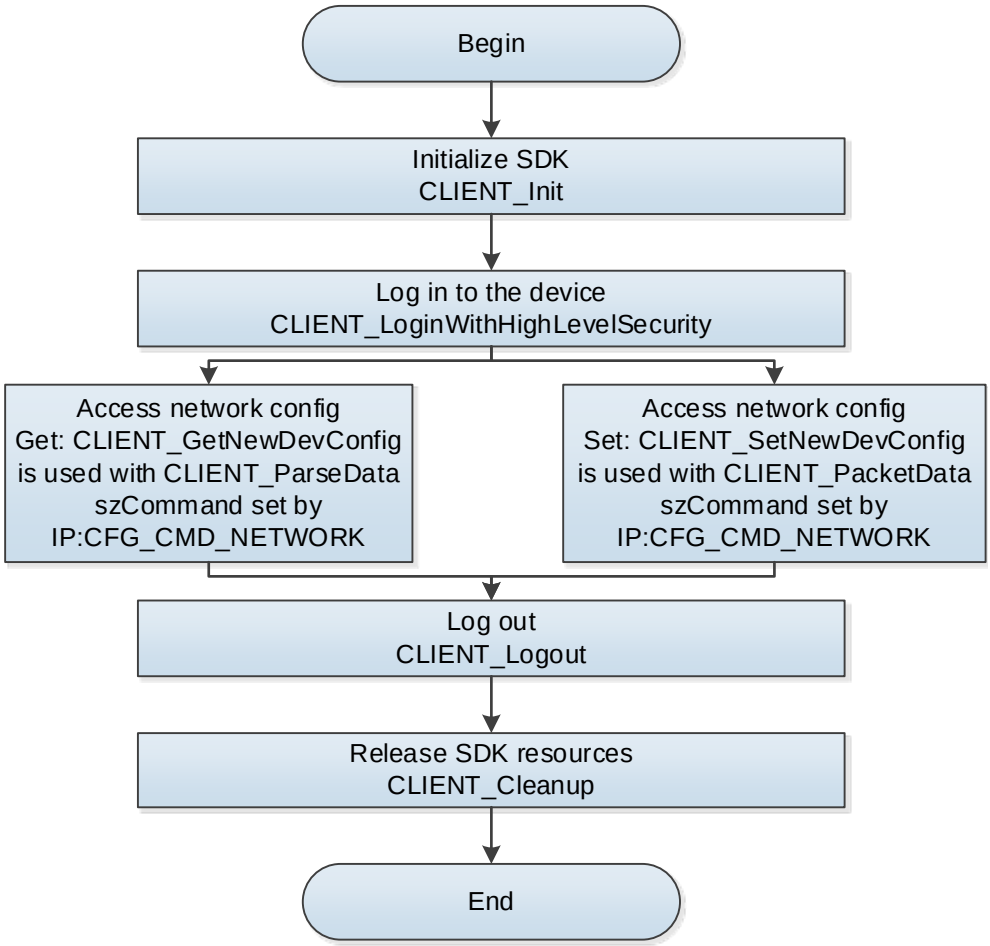
2.3.4.1.2 Interface Overview

Table 2-18 Description of IP setting interface

Interface	Description
CLIENT_GetNewDevConfig	Query config information
CLIENT_ParseData	Parse the queried config information
CLIENT_SetNewDevConfig	Set config information
CLIENT_PacketData	Pack the config information to be set into the string format

2.3.4.1.3 Process Description

Figure 2-18 IP setting



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access IP network config.

- szCommand: CFG_CMD_NETWORK.
- pBuf: CFG_NETWORK_INFO.

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** to set the access IP network config.

- szCommand: CFG_CMD_NETWORK.
- pBuf: CFG_NETWORK_INFO.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.4.1.4 Example Code

```
//Get IP network config information
char * szOut1 = new char[1024*32];
CFG_NETWORK_INFO stOut2 = {sizeof(stOut2)};
int nError = 0;
```

```
BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_NETWORK, 0, szOut1, 1024*32, &nError, 3000);
if(bRet){
	BOOL bRet1 = CLIENT_ParseData(CFG_CMD_NETWORK, szOut1, &stOut2, sizeof(CFG_NTP_INFO), NULL);
}
else{
		printf("parse failed!!!");
}
//Set IP network config information
char * szOut = new char[1024*32];
stOut2.nInterfaceNum = 1;
memcpy(stOut2.stuInterfaces[0].szIP, "192.168.1.108", sizeof(stOut2.stuInterfaces[0].szIP)-1);
memcpy(stOut2.stuInterfaces[0].szDefGateway, "192.168.1.1", sizeof(stOut2.stuInterfaces[0]. szDefGateway)-1);
memcpy(stOut2.stuInterfaces[0].szSubnetMask, "255.255.255.0", sizeof(stOut2.stuInterfaces[0]. szSubnetMask)-1);
BOOL bRet0 = CLIENT_PacketData(CFG_CMD_NETWORK, (char *)&stOut2, sizeof(CFG_NETWORK_INFO), szOut, 1024*32);
if(bRet){
	BOOL bRet1 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_NETWORK, 0, szOut, 1024*32, NULL, NULL, 3000);
}
```

2.3.4.2 Auto Register Config

2.3.4.2.1 Introduction

The auto register config process is that, you call SDK interface to configure auto register information of the device, including auto register enabling, device ID, server.

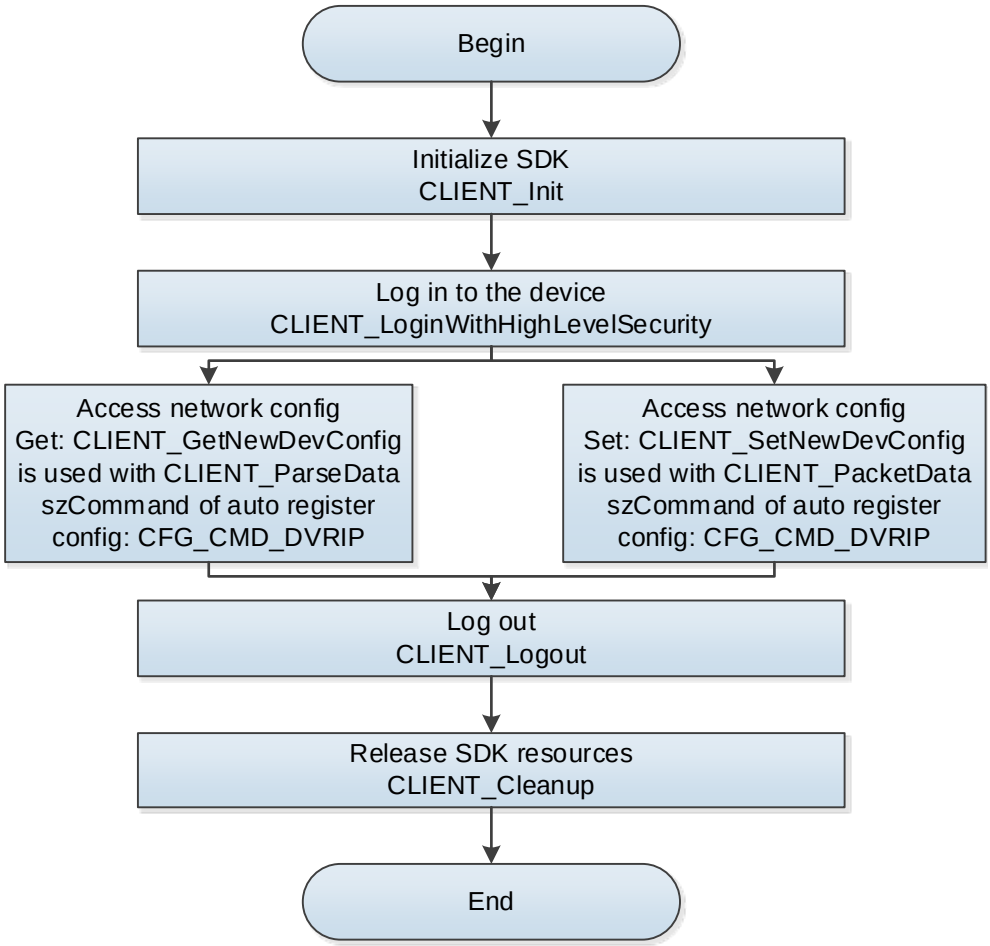
2.3.4.2.2 Interface Overview

Table 2-19 Description of interfaces for setting auto register

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.4.2.3 Process Description

Figure 2-19 Auto register setting



Process

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

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

Step 3 Access network config.

- Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access IP network config.
  - szCommand: CFG_CMD_DVRIP.
  - pBuf: CFG_DVRIP_INFO.
- Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** to set the access IP network config.
  - szCommand: CFG_CMD_DVRIP.
  - pBuf: CFG_DVRIP_INFO.

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.4.2.4 Example Code

```
//Get auto register network config 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 register network config 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);
}
```

2.3.5 Device Time Setting

2.3.5.1 DeviceTime Setting

2.3.5.1.1 Introduction

Device time setting process is that, you call SDK interface to get and set the device time.

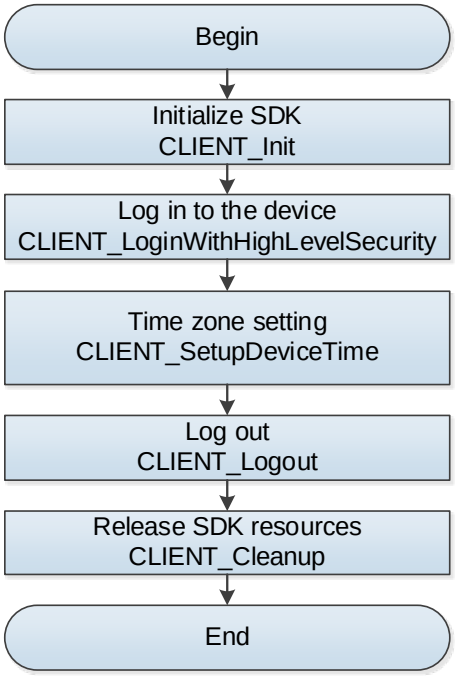
2.3.5.1.2 Interface Overview

Table 2-20 Description of time setting interfaces

Interface	Description
CLIENT_SetupDeviceTime	Set the current time of the device.

2.3.5.1.3 Process Description

Figure 2-20 Time setting



Process

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

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

Step 3 Call the **CLIENT_SetupDeviceTime** to set the access control time.

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.5.1.4 Example Code

```
//Set time zone
NET_TIME stuInfo = {sizeof(stuInfo)};
stuInfo.dwDay = 15;
stuInfo.dwYear = 2019;
stuInfo.dwMonth = 12;
stuInfo.dwHour = 17;
stuInfo.dwMinute = 45;
stuInfo.dwSecond = 25;
BOOL bRet = CLIENT_SetupDeviceTime(g_lLoginHandle, &stuInfo);
```

2.3.5.2 NTP Server and Time Zone Setting

2.3.5.2.1 Introduction

NTP server and time zone setting process is that, you call SDK interface to get and set the NTP server and time zone.

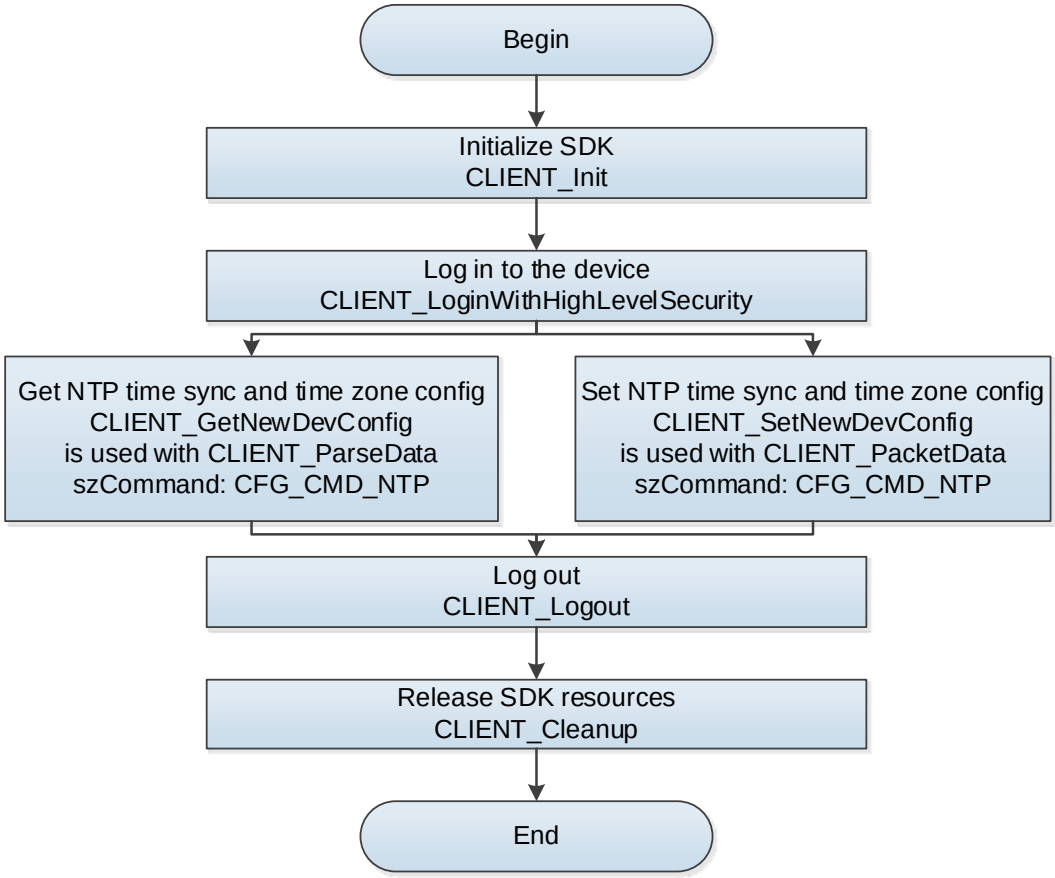
2.3.5.2.2 Interface Overview

Table 2-21 Description of NTP server and time zone interfaces

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.5.2.3 Process Description

Figure 2-21 NTP time sync



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access NTP time sync and time zone config.

- szCommand: CFG_CMD_NTP.
- pBuf: CFG_NTP_INFO.

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** to set the access NTP time sync and time zone config.

- szCommand: CFG_CMD_NTP.
- pBuf: CFG_NTP_INFO.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.5.2.4 Example Code

```
//Set NTP time sync and time zone config 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 sync and time zone config 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);
    }
```

2.3.5.3 DST Setting

2.3.5.3.1 Introduction

Daylight saving time (DST) setting process is that, you call SDK interface to get and set the DST.

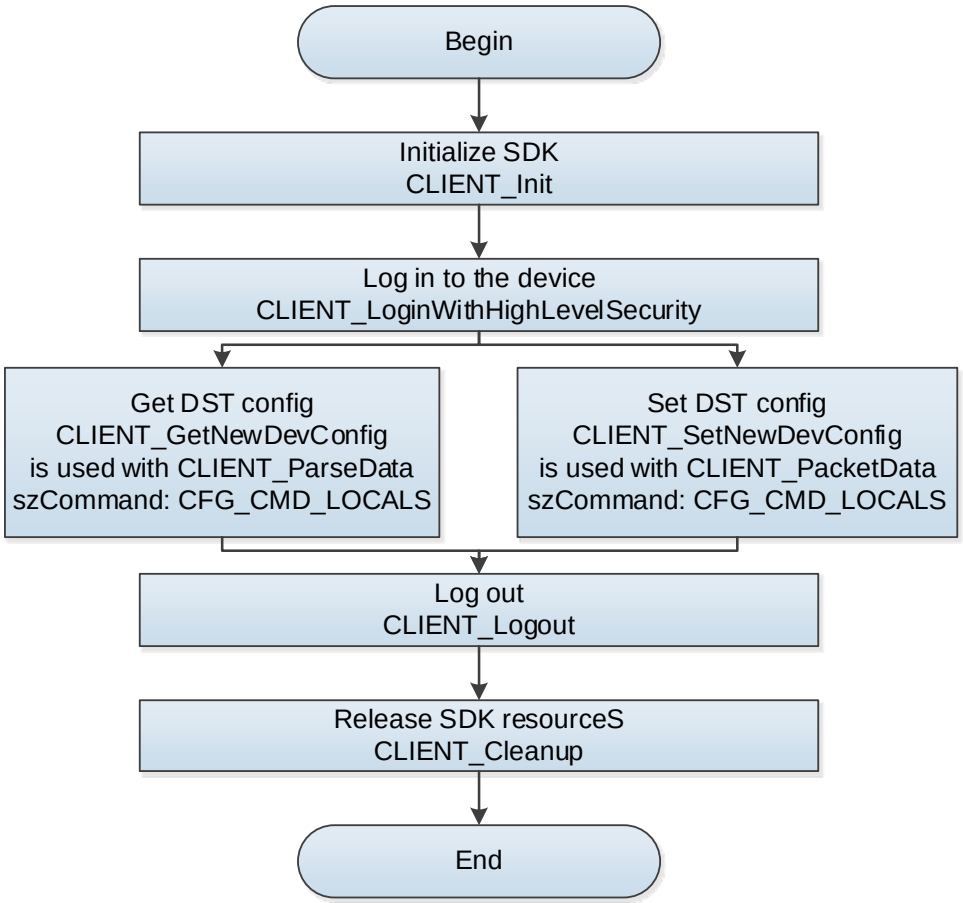
2.3.5.3.2 Interface Overview

Table 2-22 Description of DST setting interfaces

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.5.3.3 Process Description

Figure 2-22 DST setting



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access DST config.

- szCommand: CFG_CMD_LOCALS.
- pBuf: AV_CFG_Locales.

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** to set the access DST config.

- szCommand: CFG_CMD_ LOCALS.
- pBuf: AV_CFG_Locales.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.5.3.4 Example Code

```
//Set DST config information
char * szOut1 = new char[1024*32];
    AV_CFG_Locales stOut2 = {sizeof(stOut2)};
    int nError = 0;
   BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_LOCALS, 0, szOut1, 1024*32, &nError,
3000);
    if(bRet){
```

```
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_NTP, szOut1, &stOut2, sizeof(AV_CFG_Locales), NULL);
    }
    else{
        printf("parse failed!!!");
    }
//Set DST config information
    char * szOut = new char[1024*32];
    stOut2.bEnable = TRUE;
    BOOL bRet0 = CLIENT_PacketData(CFG_CMD_LOCALS, (char *)&stOut2, sizeof(AV_CFG_Locales), szOut,
1024*32);
    if(bRet)
    {
        BOOL bRet1 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_LOCALS, 0, szOut, 1024*32,
NULL, NULL, 3000);
    }
```

2.3.6 Maintenance Config

2.3.6.1 Modifying Login Password

2.3.6.1.1 Introduction

The process to modify login password is that, you call SDK interface to modify the device login password.

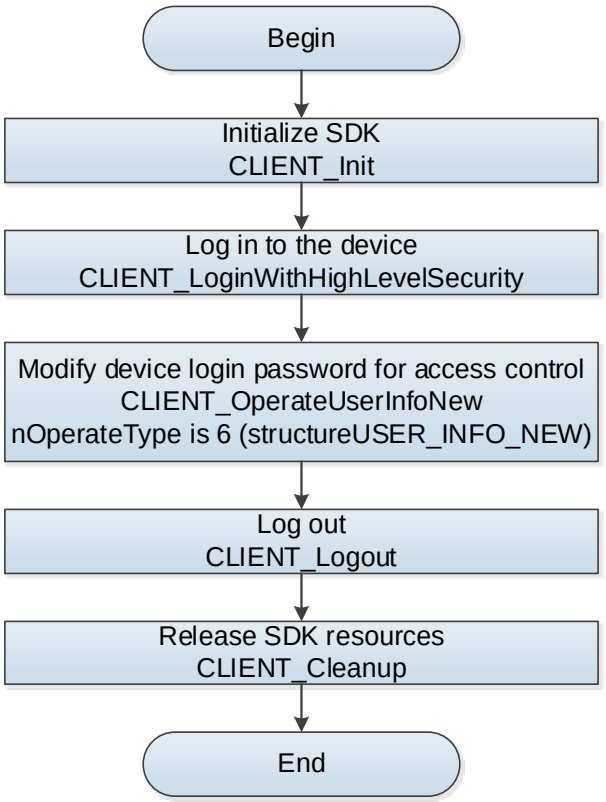
2.3.6.1.2 Interface Overview

Table 2-23 Description of interfaces for modifying login password

Interface	Description
CLIENT_OperateUserInfoNew	Make operations of device user.

2.3.6.1.3 Process Description

Figure 2-23 Maintenance config



Process

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

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

Step 3 Call the **CLIENT_OperateUserInfoNew** to operate user info to modify the device login password.

- Step 4 nOperateType: 6.
- Step 5 opParam and subParam: USER_INFO_NEW.

Step 6 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.6.1.4 Example Code

```
//Modify device login password
USER_INFO_NEW stuNewInfo = {sizeof(stuNewInfo)};
memcpy(stuNewInfo.passWord, "admin", sizeof(stuNewInfo.passWord)-1);

USER_INFO_NEW stuOldInfo = {sizeof(stuOldInfo)};
memcpy(stuOldInfo.passWord, "admin123", sizeof(stuOldInfo.passWord)-1);

BOOL bRet = CLIENT_OperateUserInfoNew(g_lLoginHandle, 6, &stuNewInfo, &stuOldInfo, NULL, 3000);
```

2.3.6.2 Restart

2.3.6.2.1 Introduction

The restart process is that, you call SDK interface to restart the device.

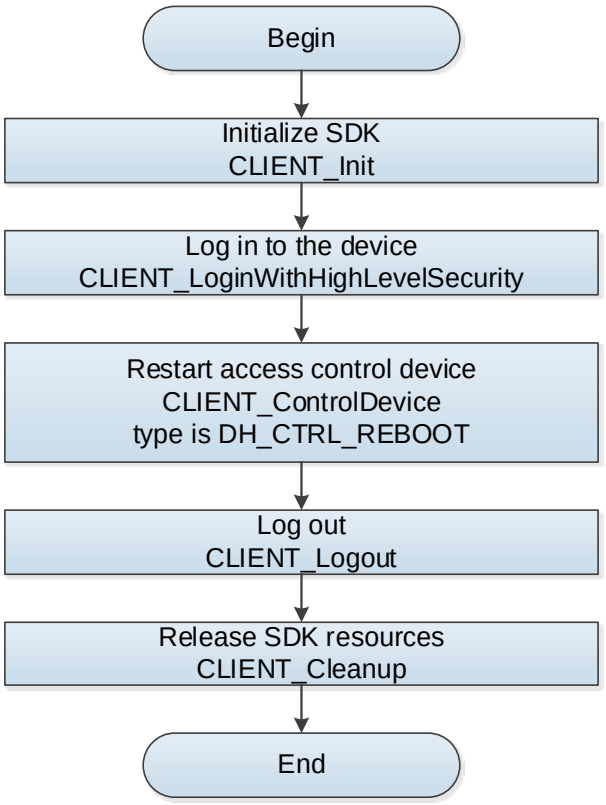
2.3.6.2.2 Interface Overview

Table 2-24 Description of device restart interface

Interface	Description
CLIENT_ControlDevice	Device control.

2.3.6.2.3 Process Description

Figure 2-24 Device restart



Process

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

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

Step 3 Call the **CLIENT_ControlDevice** to restart the device.

Step 4 Type: DH_CTRL_REBOOT.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.6.2.4 Example Code

```
//Restart
```

```
CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_REBOOT, NULL, 3000);
```

2.3.6.3 Restoring the Factory Settings

2.3.6.3.1 Introduction

The process to restore factory defaults is that, you call SDK interface to restore factory defaults of the device. After taking effect, all configurations and personnel information on the device will be cleared.

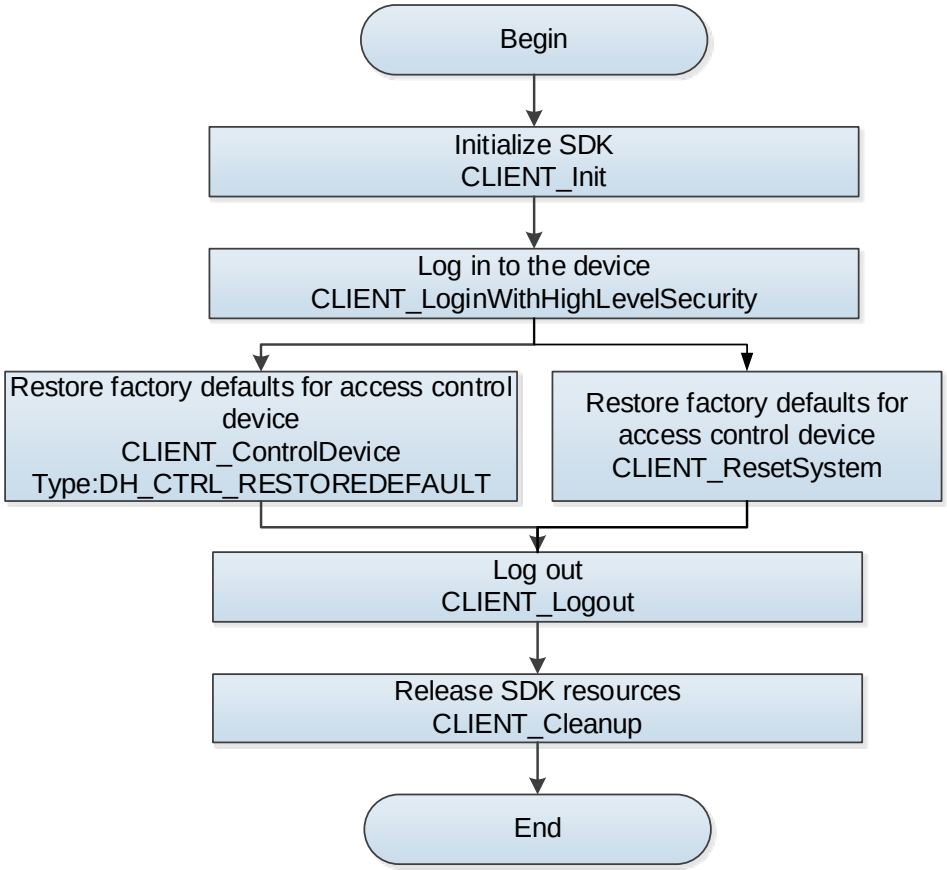
2.3.6.3.2 Interface Overview

Table 2-25 Description of interfaces for restoring factory defaults

Interface	Description
CLIENT_ControlDevice	Control device (to restore factory defaults), supporting all-in-one machine and controller.
CLIENT_ResetSystem	Control device (to restore factory defaults), supporting all-in-one machine (recommended).

2.3.6.3.3 Process Description

Figure 2-25 Factory defaults restoring



Process

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

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

Step 3 Call the **CLIENT_ResetSystem** to control the device (all-in-one fingerprint machine) to restore factory defaults.

Step 4 Call the **CLIENT_ControlDevice** to control the device (controller or all-in-one fingerprint machine) to restore factory defaults.
Type: DH_CTRL_RESTOREDEFAULT.
Param: DH_RESTORE_COMMON.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.6.3.4 Example Code

```
//Restore factory defaults
NET_IN_RESET_SYSTEM stind = {sizeof(stind)};
    NET_OUT_RESET_SYSTEM stoutd = {sizeof(stoutd)};
    BOOL bRet = CLIENT_ResetSystem(m_lLoginID,&stind, &stoutd ,5000);//You can reset the all-in-one machine
    if (!bRet)
    {
        DWORD nparam = DH_RESTORE_ALL;
        BOOL bRet = CLIENT_ControlDevice(m_lLoginID, DH_CTRL_RESTOREDEFAULT, (void*)&nparam, 3000);//You can reset the all-in-one machine and controller
        if (!bRet)
        {
            return FALSE;
        }
    }
```

2.3.6.4 Device Upgrade

2.3.6.4.1 Introduction

The device upgrade process is that, you call SDK interface to upgrade the device program.

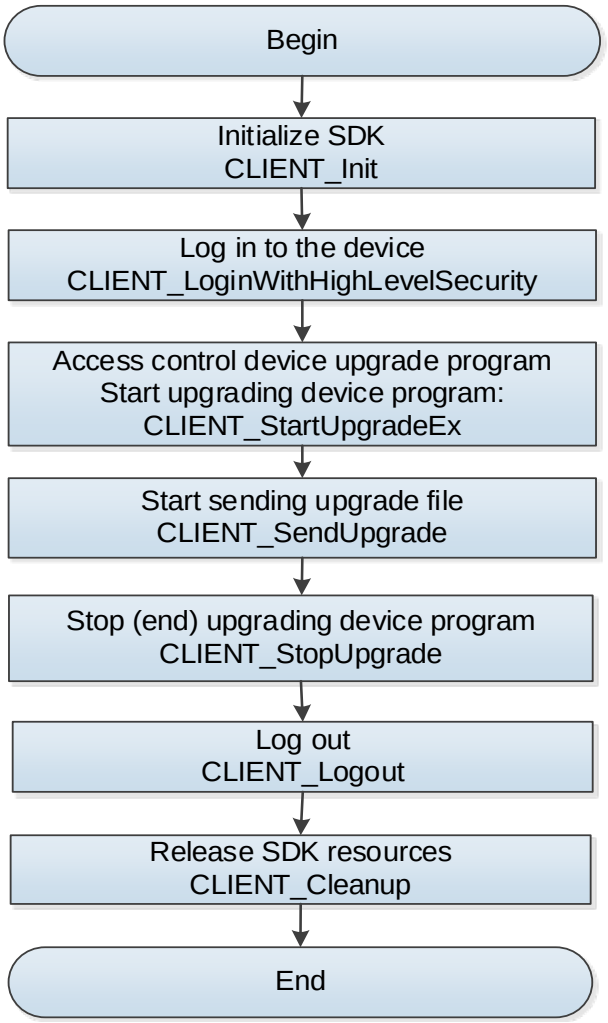
2.3.6.4.2 Interface Overview

Table 2-26 Description of device upgrade interfaces

Interface	Description
CLIENT_StartUpgradeEx	Start upgrading device program—extension.
CLIENT_SendUpgrade	Start sending upgrade file.
CLIENT_StopUpgrade	Stop upgrading.

2.3.6.4.3 Process Description

Figure 2-26 Device upgrade



Process

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

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

Step 3 Call the **CLIENT_StartUpgradeEx** to start upgrading the device program.

Step 4 Call the **CLIENT_SendUpgrade** to send the device upgrade file.

Step 5 Call the **CLIENT_StopUpgrade** to stop/end upgrading the device program.

Step 6 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.6.4.4 Example Code

```
BOOL m_isNeedStop = FALSE;
void CALLBACK UpgradeCallBack(LLONG lLoginID, LLONG lUpgradechannel, int nTotalSize, int nSendSize,
LDWORD dwUser)
{
    if (0 == lLoginID || 0 == lUpgradechannel)
    {
        cout << "lLoginID or lUpgradechannel is zero" << endl;
```

```
            m_isNeedStop = TRUE;
            return;
        }
        if (0 == nTotalSize && -1 == nSendSize) //It represents the end of upgrade
        {
            m_isNeedStop = TRUE;
            cout << "Upgrade completed!" << endl;
        }
        else if (0 == nTotalSize && -2 == nSendSize) //It represents upgrade error
        {
            m_isNeedStop = TRUE;
            cout << "Upgrade error" << endl;
        }
        else if (nTotalSize > 0 && nSendSize >= 0) // It represents the sending progress
        {
            float fPross = (float)(nSendSize/nTotalSize);
            printf("Upgrade file sending progress (total file size: % d, sent size: % d, sending progress: %.2f%%）
\n", nTotalSize, nSendSize, fPross*100);
            if (nTotalSize == nSendSize)
            {
                cout << "The upgrade file has been sent! The device start upgrading ....." << endl;
            }
        }
        else if (nTotalSize == -1 && nSendSize >= 0)
        {
            cout << "...................Upgrade progress: " << nSendSize << "......................." << endl;
        }
}

void Test()
{
        char szFileName[256] = {0};
        cout << "Enter the upgrade program file name (including the full path):" << endl;
        cin >> szFileName;
        //Start upgrading the device program
        LLONG lUpHandle = CLIENT_StartUpgradeEx(g_lLoginHandle, DH_UPGRADE_BOOT_YPE, szFileName,
UpgradeCallBack, 0);
        if (0 == lUpHandle)
        {
            printf("CLIENT_StartUpgrade failed. ErrorCode[%x]\n", CLIENT_GetLastError());
```

```
        return;
    }
    //Send the upgrade file
    BOOL bRet = CLIENT_SendUpgrade(lUpHandle);
    if (!bRet)
    {
        printf("CLIENT_SendUpgrade failed. ErrorCode[%x]\n", CLIENT_GetLastError());
        //Stop upgrading the program
        CLIENT_StopUpgrade(lUpHandle);
        return;
    }
    while (true)
    {
        if (m_isNeedStop)
        {
            //Stop upgrading the program
            bRet = CLIENT_StopUpgrade(lUpHandle);
            if (!bRet)
            {
                printf("CLIENT_SendUpgrade failed. ErrorCode[%x]\n", CLIENT_GetLastError());
                return;
            }
            cout << "Success to stop upgrade!!" << endl;
            break;
        }
    }
}
```

2.3.6.5 Auto Maintenance

2.3.6.5.1 Introduction

The auto maintenance process is that, you call SDK interface to configure the auto maintenance of device, including information such as auto restart time.

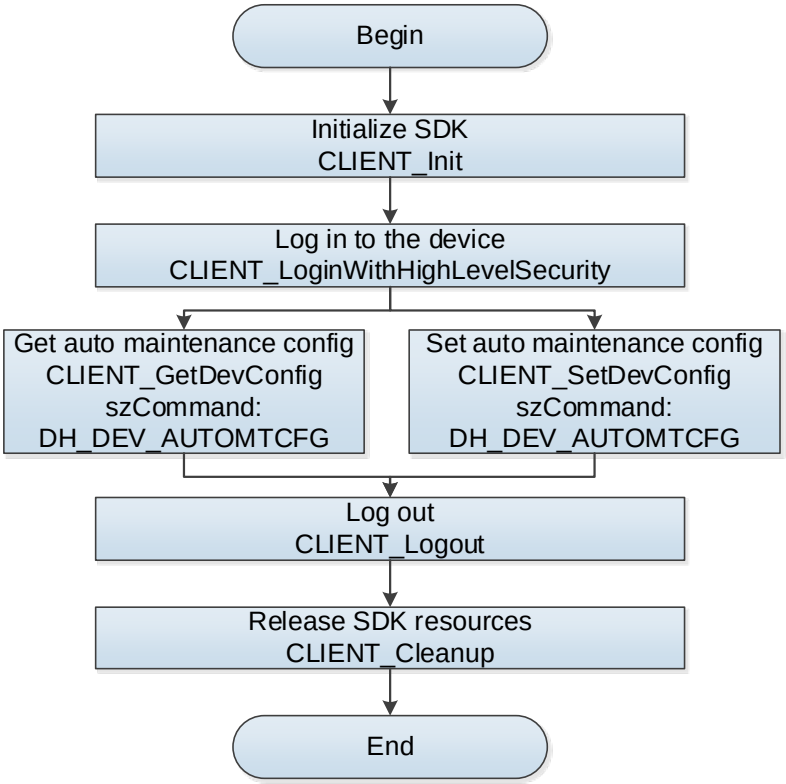
2.3.6.5.2 Interface Overview

Table 2-27 Description of auto maintenance interfaces

Interface	Description
CLIENT_GetDevConfig	Query config information.
CLIENT_SetDevConfig	Set config information.

2.3.6.5.3 Process Description

Figure 2-27 Auto maintenance



Process

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

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

Step 3 Call the **CLIENT_GetDevConfig** to query the access auto maintenance info.

- szCommand: DH_DEV_AUTOMTCFG.
- pBuf: DHDEV_AUTOMT_CFG.

Step 4 Call the **CLIENT_SetDevConfig** to set the access auto maintenance info.

- szCommand: DH_DEV_AUTOMTCFG.
- pBuf: DHDEV_AUTOMT_CFG.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.6.5.4 Example Code

```
//Get the auto maintenance config information
DHDEV_AUTOMT_CFG stInfo = {sizeof(stInfo)};
DWORD lpBytesReturned = 0;
BOOL bRet12 = CLIENT_GetDevConfig(g_lLoginHandle, DH_DEV_AUTOMTCFG, 0, &stInfo, sizeof(stInfo),
&lpBytesReturned, 5000);
//Set the auto maintenance config information
stInfo.byAutoRebootDay = 1;
BOOL bRet11 = CLIENT_SetDevConfig(g_lLoginHandle, DH_DEV_AUTOMTCFG, 0, &stInfo, sizeof(stInfo), 5000);
```

2.3.7 Personnel Management

2.3.7.1 Introduction

For personnel information, you can call SDK to add, delete, query and modify personnel information fields of the access device (including No., name, face, card, fingerprint, password, user permission, period, holiday plan and user type).

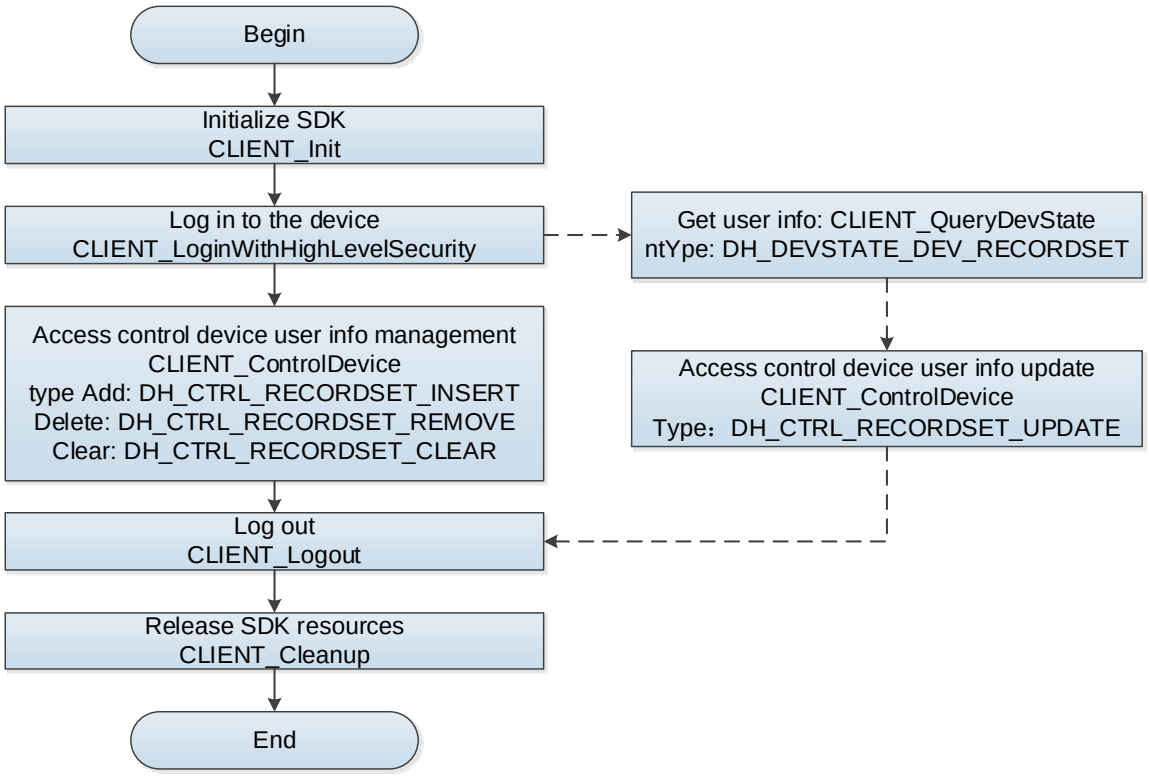
2.3.7.2 Interface Overview

Table 2-28 Description of personnel information interfaces

Interface	Description
CLIENT_ControlDevice	Control device.
CLIENT_QueryDevState	Query device status.

2.3.7.3 Process Description

Figure 2-28 User information management



Process

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

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

Step 3 Call the **CLIENT_ControlDevice** to operate holiday information.

Table 2-29 Description and structure of type

Type	Description	emType	Param
● DH_CTRL_RECORDSET_INSERT ● DH_CTRL_RECORDSET_INSERTEX	Add user info	NET_RECORD_ACCESSCTLCARD	● NET_CTRL_RECORDSET_INSERT_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD
DH_CTRL_RECORDSET_REMOVE	Delete user info	NET_RECORD_ACCESSCTLCARD	● NET_CTRL_RECORDSET_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD
DH_CTRL_RECORDSET_CLEAR	Clear user info	NET_RECORD_ACCESSCTLCARD	NET_CTRL_RECORDSET_PARAM

Step 4 Call the **CLIENT_QueryDevState** interface to get user information.

Table 2-30 Description and structure of type

Type	Description	emType	Param
DH_DEVSTATE_DEV_RECORDSET	Get user info	NET_RECORD_ACCESSCTLCARD	● NET_CTRL_RECORDSET_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD

Step 5 Call the **CLIENT_ControlDevice** to update user information.

Table 2-31 Description and structure of type

Type	Description	emType	Param
● DH_CTRL_RECORDSET_UPDATE ● DH_CTRL_RECORDSET_UPDATEEX	Update user info	NET_RECORD_ACCESSCTLCARD	● NET_CTRL_RECORDSET_PARAM ● NET_RECORDSET_ACCESS_CTL_CARD

Step 6 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

Note

- Card number: Personnel card number.
- Card type: When the card is set as duress card, if the person bound to this card opens the door with card password, unlock password or by fingerprint, the duress alarm will be triggered.
- Card password: Suitable for card + password mode.
- Period: Select the serial number corresponding to the configured time period. If there is no serial number, set it in "2.3.9.1 Period Config."
- Unlock password: After setting this password, you can directly enter the password to open the door without swiping card. For details, see "2.3.10.5 Unlock Password."
- Valid number of times: Only guest users can set this field.
- Whether it is first card: Select as needed. For according to the actual situation. For the configuration method of the first card, see "2.3.10.1 Unlock at Designated Intervals and First Card Unlock."

2.3.7.4 Example Code

```
NET_RECORDSET_ACCESS_CTL_CARD stuInfo = {sizeof(stuInfo)};
    stuInfo.emSex = NET_ACCESSCTLCARD_SEX_MALE;
    stuInfo.nDoorNum = 2;
    stuInfo.sznDoors[0] = 1223;
    memcpy(stuInfo.szUserID, "ddjdj", sizeof(stuInfo.szUserID));
    memcpy(stuInfo.szPsw, "543543", sizeof(stuInfo.szPsw));



    NET_CTRL_RECORDSET_INSERT_PARAM stuParam = {sizeof(stuParam)};
    stuParam.stuCtrlRecordSetInfo.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_IN);
    stuParam.stuCtrlRecordSetInfo.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    stuParam.stuCtrlRecordSetInfo.pBuf = (void*)&stuInfo;
    stuParam.stuCtrlRecordSetInfo.nBufLen = sizeof(stuInfo);

    stuParam.stuCtrlRecordSetResult.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_OUT);

    BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_INSERT, &stuParam, 5000);
//Delete
    stuInfo.nRecNo = 123456789;
    NET_CTRL_RECORDSET_PARAM stuParam1 = {sizeof(stuParam1)};
    stuParam1.emType = NET_RECORD_ACCESSCTLCARD;
    stuParam1.pBuf = (void*)&stuInfo.nRecNo;
    stuParam1.nBufLen = sizeof(stuInfo.nRecNo);

    BOOL bRet1 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_REMOVE, &stuParam1,
5000);
//Clear
    NET_CTRL_RECORDSET_PARAM stuParam2 = {sizeof(stuParam2)};
    stuParam2.emType = NET_RECORD_ACCESSCTLCARD;
    BOOL bRet2 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_CLEAR, &stuParam2, 5000);
//Get
    stuInfo.nRecNo = 123456789;
    NET_CTRL_RECORDSET_PARAM stuParam3 = {sizeof(stuParam3)};
    stuParam3.emType = NET_RECORD_ACCESSCTLCARD;

    NET_RECORDSET_HOLIDAY stuHoliday = {sizeof(stuHoliday)};
    stuHoliday.nRecNo = stuInfo.nRecNo;
    stuParam3.pBuf = &stuHoliday;
```

```
    int nRet = 0;
    BOOL bRet3 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam3,
        sizeof(stuParam3), &nRet, 5000);
//Update
    stuInfo.nRecNo = 123456789;
    NET_CTRL_RECORDSET_PARAM stuParam4 = {sizeof(stuParam4)};
    stuParam4.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    stuParam4.pBuf = (void*)&stuInfo;
    stuParam4.nBufLen = sizeof(stuInfo);

    int nRet4 = 0;
    BOOL bRet4 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam4,sizeof(stuParam4), &nRet4, 5000);
    if (bRet4)
    {
        stuInfo.emSex = NET_ACCESSCTLCARD_SEX_MALE;
        stuInfo.nDoorNum = 2;
        stuInfo.sznDoors[0] = 1223;
        memcpy(stuInfo.szUserID, "2222", sizeof(stuInfo.szUserID));
        memcpy(stuInfo.szPsw, "fdsfds", sizeof(stuInfo.szPsw));

        stuParam4.emType = NET_RECORD_ACCESSCTLHOLIDAY;
        stuParam4.pBuf = (void*)&stuInfo;
        stuParam4.nBufLen = sizeof(stuInfo);

        // Update info
        BOOL bRet4 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_UPDATE, &stuParam,
5000);
    }
    else{
        printf("CLIENT_QueryDevState failed!");
    }
```

2.3.8 Door Config

2.3.8.1 Introduction

For door config information, you can call SDK interface to get and set door config of the access device, including unlock mode, lock holding, lock timeout, holiday period number, unlock period, and alarm enabling option.

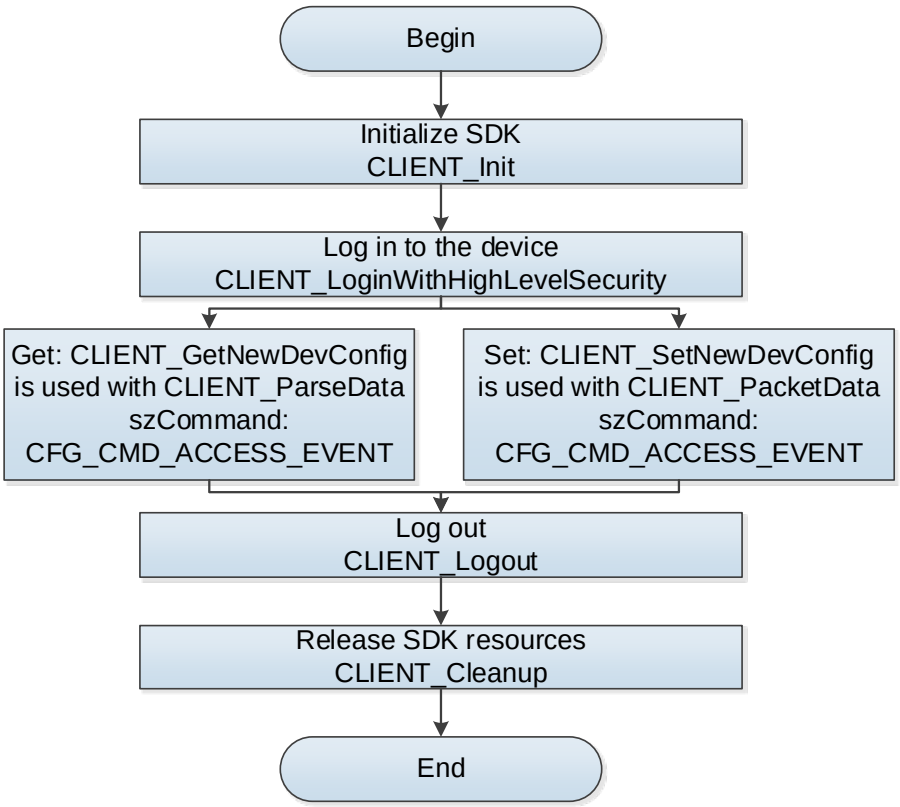
2.3.8.2 Interunlockface Overview

Table 2-32 Description of door config information interfaces

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.8.3 Process Description

Figure 2-29 Door config information



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access door info.

- szCommand: CFG_CMD_ACCESS_EVENT.
- pBuf: CFG_ACCESS_EVENT_INFO.

Table 2-33 Description of CFG_ACCESS_EVENT_INFO

CFG_ACCESS_EVENT_INFO	Description
emState	Door status
nUnlockHoldInterval	Unlock duration
nCloseTimeout	Lock timeout period
emDoorOpenMethod	Unlock mode
bDuressAlarmEnable	duress
bBreakInAlarmEnable	Intrusion alarm enabling
bRepeatEnterAlarm	Repeat entry alarm enabling
abDoorNotClosedAlarmEnable	Interlock alarm enabling
abSensorEnable	Door sensor enabling

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** to set the access door info.

- szCommand: CFG_CMD_ACCESS_EVENT.
- pBuf: CFG_ACCESS_EVENT_INFO.

Table 2-34 Description of CFG_ACCESS_EVENT_INFO

CFG_ACCESS_EVENT_INFO	Description
emState	Door status
nUnlockHoldInterval	Unlock duration
nCloseTimeout	Lock timeout period
emDoorOpenMethod	Unlock mode
bDuressAlarmEnable	duress
bBreakInAlarmEnable	Intrusion alarm enabling
bRepeatEnterAlarm	Repeat entry alarm enabling
abDoorNotClosedAlarmEnable	Interlock alarm enabling
abSensorEnable	Door sensor enabling

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

Note

- When the intrusion alarm and unlock alarm are enabled, users need enable door sensor so that the intrusion alarm and door open alarm can be implemented.
- Set the serial number of always open period, always close period and remote verifitication. For details, see "2.3.9.1 Period Config."

2.3.8.4 Example Code

```
//Get door config information
char * szOut1 = new char[1024*32];
    CFG_ACCESS_EVENT_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
```

```
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut1, 1024*32,
&nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESS_EVENT, szOut1, &stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), NULL);
        if (bRet1)
        {
            printf("door status: %d\n",stOut2.emState);
            printf("unlock duration: %d\n",stOut2.nUnlockHoldInterval);
            printf("lock timeout period: %d\n",stOut2.nCloseTimeout);
            printf("unlock mode: %d\n",stOut2.emDoorOpenMethod);
            printf("duress: %d\n",stOut2.bDuressAlarmEnable);
        }
    }
    else{
        printf("parse failed!!!");
    }
//Set door config information
    char * szOut = new char[1024*32];
    stOut2.emState = ACCESS_STATE_NORMAL;//Door status
    stOut2.nUnlockHoldInterval = 10;//Unlock duration
    stOut2.nCloseTimeout = 10;//Lock timeout period
    stOut2.emDoorOpenMethod = CFG_DOOR_OPEN_METHOD_PWD_ONLY;//Unlock mode
    stOut2.bDuressAlarmEnable = FALSE;//Duress
    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESS_EVENT, (char *)&stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), szOut, 1024*32);
    if(bRet2)
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut,
1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
```

```
}
```

2.3.9 Door Time Config

2.3.9.1 Period Config

2.3.9.1.1 Introduction

For period config information, you can call SDK interface to get and set the door period of the access control device. The configuration of this template cannot directly take effect on the device and needs to be called by other function modules.

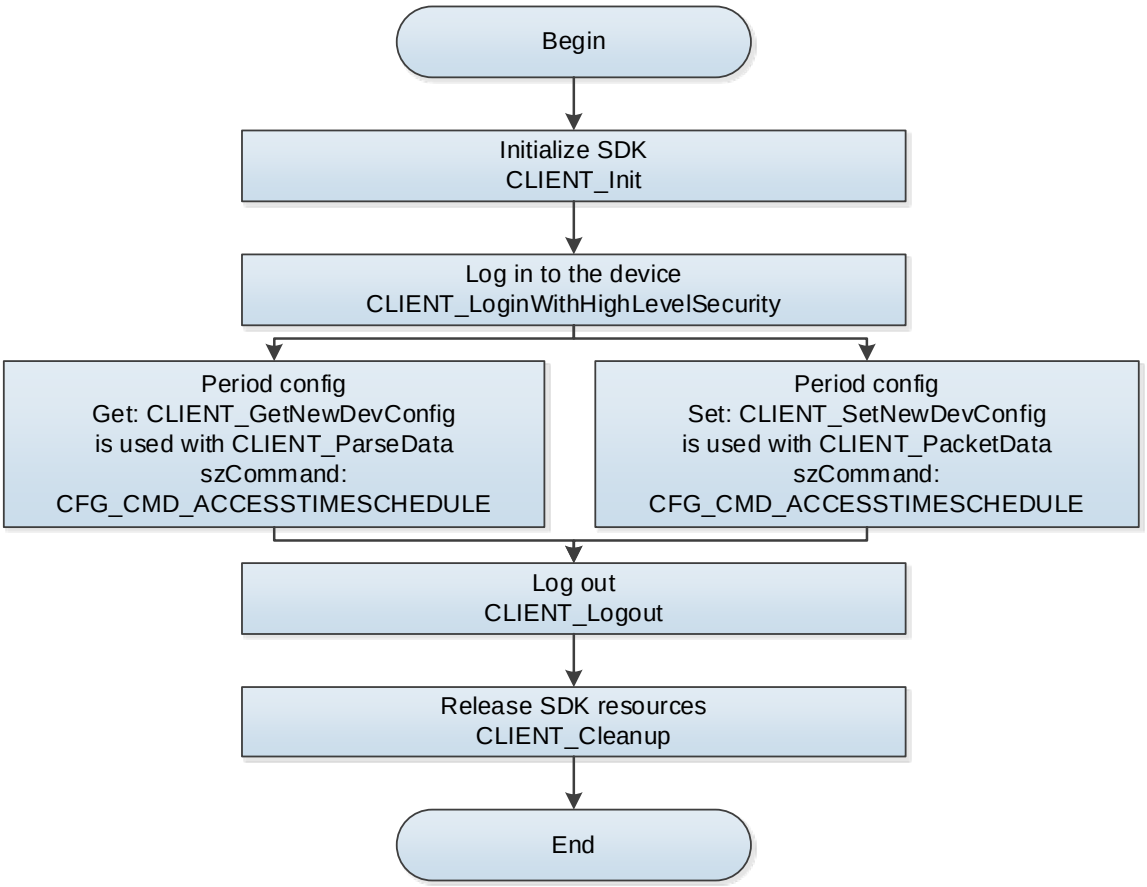
2.3.9.1.2 Interface Overview

Table 2-35 Description of period interfaces

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.9.1.3 Process Description

Figure 2-30 Period config



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access period info.

- szCommand: CFG_CMD_ACCESSTIMESCHEDULE.
- pBuf: CFG_ACCESS_TIMESCHEDULE_INFO.

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** to set the access period info.

- szCommand: CFG_CMD_ACCESSTIMESCHEDULE.
- pBuf: CFG_ACCESS_TIMESCHEDULE_INFO.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.9.1.4 Example Code

```
//Get period config information
char * szOut1 = new char[1024*32];
    CFG_ACCESS_TIMESCHEDULE_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESSTIMESCHEDULE, 0, szOut1,
1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESSTIMESCHEDULE, szOut1, &stOut2,
sizeof(CFG_ACCESS_TIMESCHEDULE_INFO), NULL);
        if (bRet1)
        {
            printf("enabling: %d\n",stOut2.bEnable);
            printf("custom name: %s\n",stOut2.szName);
        }
    }
    else{
        printf("parse failed!!!");
    }
//Set period config information.
    char * szOut = new char[1024*32];
    stOut2.bEnable = TRUE;
    memcpy(stOut2.szName, "ghgj", sizeof(stOut2.szName));

    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESSTIMESCHEDULE, (char *)&stOut2,
sizeof(CFG_ACCESS_TIMESCHEDULE_INFO), szOut, 1024*32);
    if(bRet2)
    {
```

```
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESSTIMESCHEDULE, 0,
szOut, 1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.9.2 Always Open and Always Closed Period Config

2.3.9.2.1 Introduction

For always open and always closed period config, you can call SDK interface to get and set the period config of the access control device, including always open period, always closed period, remote verification period.

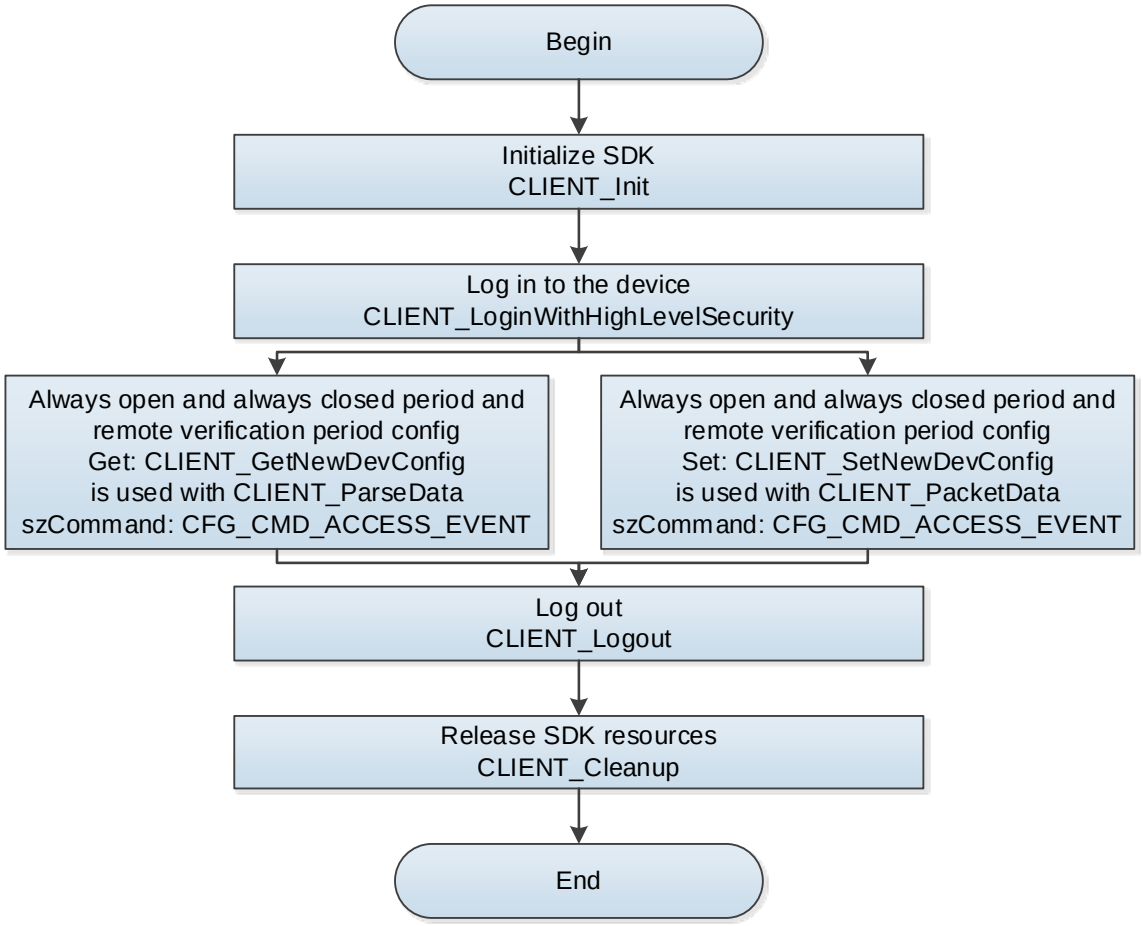
2.3.9.2.2 Interface Overview

Table 2-36 Description of always open and always closed period config interfaces

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.9.2.3 Process Description

Figure 2-31 Always open and always closed period config



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access always open and always closed period info, and remote verification period.

- szCommand: CFG_CMD_ACCESS_EVENT.
- pBuf: CFG_ACCESS_EVENT_INFO.

Table 2-37 Description of CFG_ACCESS_EVENT_INFO

CFG_ACCESS_EVENT_INFO	Description
nOpenAlwaysTimeIndex	Always open period config
nCloseAlwaysTimeIndex	Always closed period config
stuAutoRemoteCheck	Remote verification period

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** in pairs to set the access always open and always closed period info, and remote verification period.

- szCommand: CFG_CMD_ACCESS_EVENT.
- pBuf: CFG_ACCESS_EVENT_INFO.

Table 2-38 Description of CFG_ACCESS_EVENT_INFO

CFG_ACCESS_EVENT_INFO	Description
nOpenAlwaysTimeIndex	Always open period config

nCloseAlwaysTimeIndex	Always closed period config
stuAutoRemoteCheck	Remote verification period

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

Note

Set the serial number of always open period, always close period and remote verifitication. For details, see "2.3.9.1 Period Config."

2.3.9.2.4 Example Code

```
//Get always open, always closed and remote verification period config information
char * szOut1 = new char[1024*32];
    CFG_ACCESS_EVENT_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut1, 1024*32,
&nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESS_EVENT, szOut1, &stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), NULL);
        if (bRet1)
        {
            printf("always open period config: %d\n",stOut2.nOpenAlwaysTimeIndex);
            printf("always clsoed period config: %s\n",stOut2.nCloseAlwaysTimeIndex);
            printf("remote verification period enabling: %d\n", stOut2.stuAutoRemoteCheck.bEnable);
        }
    }
    else{
        printf("parse failed!!!");
    }
    char * szOut = new char[1024*32];
    stOut2.nOpenAlwaysTimeIndex = 02;
    stOut2.nCloseAlwaysTimeIndex = 03;
    stOut2.stuAutoRemoteCheck.bEnable = TRUE;
//Get always open, always closed and remote verification period config information
    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESS_EVENT, (char *)&stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), szOut, 1024*32);
    if(bRet2)
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut,
1024*32, NULL, NULL, 3000);
        if (bRet3)
```

```
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.9.3 Holiday Config

2.3.9.3.1 Introduction

For holiday config, you can call SDK interface to get and configure the holiday of the access control device.

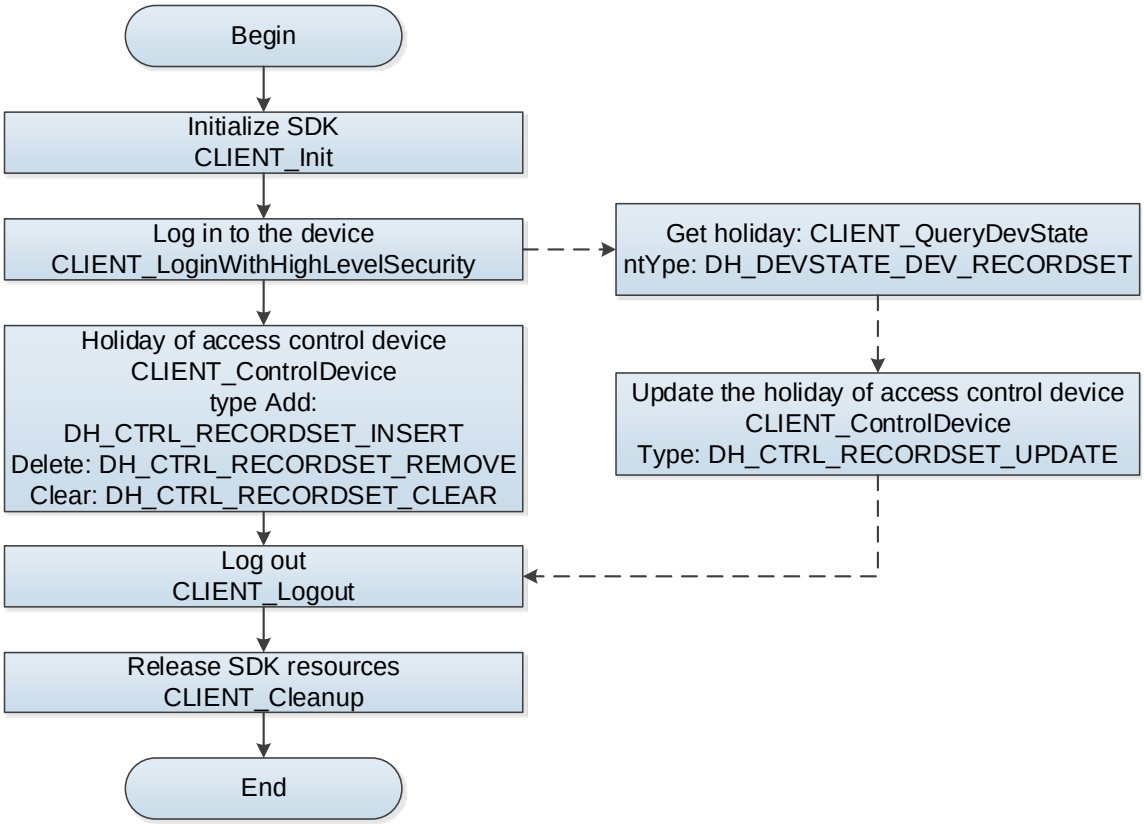
2.3.9.3.2 Interface Overview

Table 2-39 Description of holiday config interfaces

Interface	Description
CLIENT_ControlDevice	Control device.
CLIENT_QueryDevState	Query device status.

2.3.9.3.3 Process Description

Figure 2-32 Holiday config



Process

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

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

Step 3 Call the **CLIENT_ControlDevice** to operate holiday information.

Table 2-40 Description and structure of type

Type	Description	emType	Param
DH_CTRL_RECORDSET_INSERT	Add holiday	NET_RECORD_ACCESSCTLHOLIDAY	• NET_CTRL_RECORDSET_INSERT_PARAM • NET_RECORDSET_HOLIDAY
DH_CTRL_RECORDSET_REMOVE	Delete holiday	NET_RECORD_ACCESSCTLHOLIDAY	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_HOLIDAY
DH_CTRL_RECORDSET_CLEAR	Clear holiday	NET_RECORD_ACCESSCTLHOLIDAY	NET_CTRL_RECORDSET_PARAM

Step 4 Call the **CLIENT_QueryDevState** interface to **get holiday** information.

Table 2-41 Description and structure of type

Type	Description	emType	Param
DH_DEVSTATE_DEV_RECORDSET	Get holiday	NET_RECORD_ACCESSCTLHOLIDAY	• NET_CTRL_RECORDSET_PARAM • NET_RECORDSET_HOLIDAY

Step 5 Call the CLIENT_ControlDevice to update holiday information.

Table 2-42 Description and structure of type

Type	Description	emType	Param
DH_CTRL_RECORDSET_UPDATE	Update holiday	NET_RECORD_ACCESSCTLHOLIDAY	• NET_CTRL_RECORDSET_PARAM • NET_RECORDSET_HOLIDAY

Step 6 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.3.9.3.4 Example Code

```
//Add holiday
NET_RECORDSET_HOLIDAY stuInfo = {sizeof(stuInfo)};
    stuInfo.bEnable = TRUE;
    stuInfo.nDoorNum = 2;
    stuInfo.sznDoors[0] = 1223;
    stuInfo.stuEndTime.dwYear = 2019;
    stuInfo.stuEndTime.dwMonth = 12;
    stuInfo.stuEndTime.dwDay = 4;
    stuInfo.stuEndTime.dwHour = 12;
    stuInfo.stuEndTime.dwMinute = 22;
    stuInfo.stuEndTime.dwSecond = 12;

    stuInfo.stuStartTime.dwYear = 2019;
    stuInfo.stuStartTime.dwMonth = 12;
    stuInfo.stuStartTime.dwDay = 6;
    stuInfo.stuStartTime.dwHour = 12;
    stuInfo.stuStartTime.dwMinute = 22;
    stuInfo.stuStartTime.dwSecond = 12;

    memcpy(stuInfo.szHolidayName, "May Day", sizeof(stuInfo.szHolidayName));
    memcpy(stuInfo.szHolidayNo, "12345", sizeof(stuInfo.szHolidayNo));



    NET_CTRL_RECORDSET_INSERT_PARAM stuParam = {sizeof(stuParam)};
    stuParam.stuCtrlRecordSetInfo.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_IN);
    stuParam.stuCtrlRecordSetInfo.emType = NET_RECORD_ACCESSCTLHOLIDAY;
    stuParam.stuCtrlRecordSetInfo.pBuf = (void*)&stuInfo;
    stuParam.stuCtrlRecordSetInfo.nBufLen = sizeof(stuInfo);

    stuParam.stuCtrlRecordSetResult.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_OUT);

    BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_INSERT, &stuParam, 5000);
```

```
//Delete holiday
stuInfo.nRecNo = 123456789;
NET_CTRL_RECORDSET_PARAM stuParam1 = {sizeof(stuParam1)};
	stuParam1.emType = NET_RECORD_ACCESSCTLHOLIDAY;
	stuParam1.pBuf = (void*)&stuInfo.nRecNo;
	stuParam1.nBufLen = sizeof(stuInfo.nRecNo);

	BOOL bRet1 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_REMOVE, &stuParam1,
5000);
//Clear holiday
	NET_CTRL_RECORDSET_PARAM stuParam = {sizeof(stuParam)};
	stuParam.emType = NET_RECORD_ACCESSCTLHOLIDAY;
	BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_CLEAR, &stuParam, 5000);
//Get holiday
stuInfo.nRecNo = 123456789;
	NET_CTRL_RECORDSET_PARAM stuParam3 = {sizeof(stuParam3)};
	stuParam3.emType = NET_RECORD_ACCESSCTLHOLIDAY;
	NET_RECORDSET_HOLIDAY stuHoliday = {sizeof(stuHoliday)};
	stuHoliday.nRecNo = stuInfo.nRecNo;
	stuParam3.pBuf = &stuHoliday;
	int nRet = 0;
	BOOL bRet3 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam3,sizeof(stuParam3), &nRet, 5000);
//Update holiday
stuInfo.nRecNo = 123456789;
	NET_CTRL_RECORDSET_PARAM stuParam = {sizeof(stuParam)};
	stuParam.emType = NET_RECORD_ACCESSCTLHOLIDAY;
	stuParam.pBuf = (void*)&stuInfo;
	stuParam.nBufLen = sizeof(stuInfo);

	int nRet = 0;
	BOOL bRet = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam,sizeof(stuParam), &nRet, 5000);
	if (bRet)
	{
		stuInfo.bEnable = TRUE;
		stuInfo.nDoorNum = 2;
		stuInfo.sznDoors[0] = 1223;
		stuInfo.stuEndTime.dwYear = 2019;
		stuInfo.stuEndTime.dwMonth = 10;
```

```
        stuInfo.stuEndTime.dwDay = 4;
        stuInfo.stuEndTime.dwHour = 12;
        stuInfo.stuEndTime.dwMinute = 22;
        stuInfo.stuEndTime.dwSecond = 12;

        stuInfo.stuStartTime.dwYear = 2019;
        stuInfo.stuStartTime.dwMonth = 12;
        stuInfo.stuStartTime.dwDay = 6;
        stuInfo.stuStartTime.dwHour = 12;
        stuInfo.stuStartTime.dwMinute = 22;
        stuInfo.stuStartTime.dwSecond = 12;

        memcpy(stuInfo.szHolidayName, "International Children's Day", sizeof(stuInfo.szHolidayName));
        memcpy(stuInfo.szHolidayNo, "12345", sizeof(stuInfo.szHolidayNo));

        stuParam.emType = NET_RECORD_ACCESSCTLHOLIDAY;
        stuParam.pBuf = (void*)&stuInfo;
        stuParam.nBufLen = sizeof(stuInfo);

        BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_UPDATE, &stuParam,
5000);
    }
    else{
        printf("CLIENT_QueryDevState failed!");
    }
```

2.3.10 Advanced Config of Door

2.3.10.1 Unlock at Designated Intervals and First Card Unlock

2.3.10.1.1 Introduction

For unlock at designated intervals and first card unlock, you can call SDK interface to get and set the config of unlock at designated intervals, first card unlock and first user unlock of the access control device.

2.3.10.1.2 Interface Overview

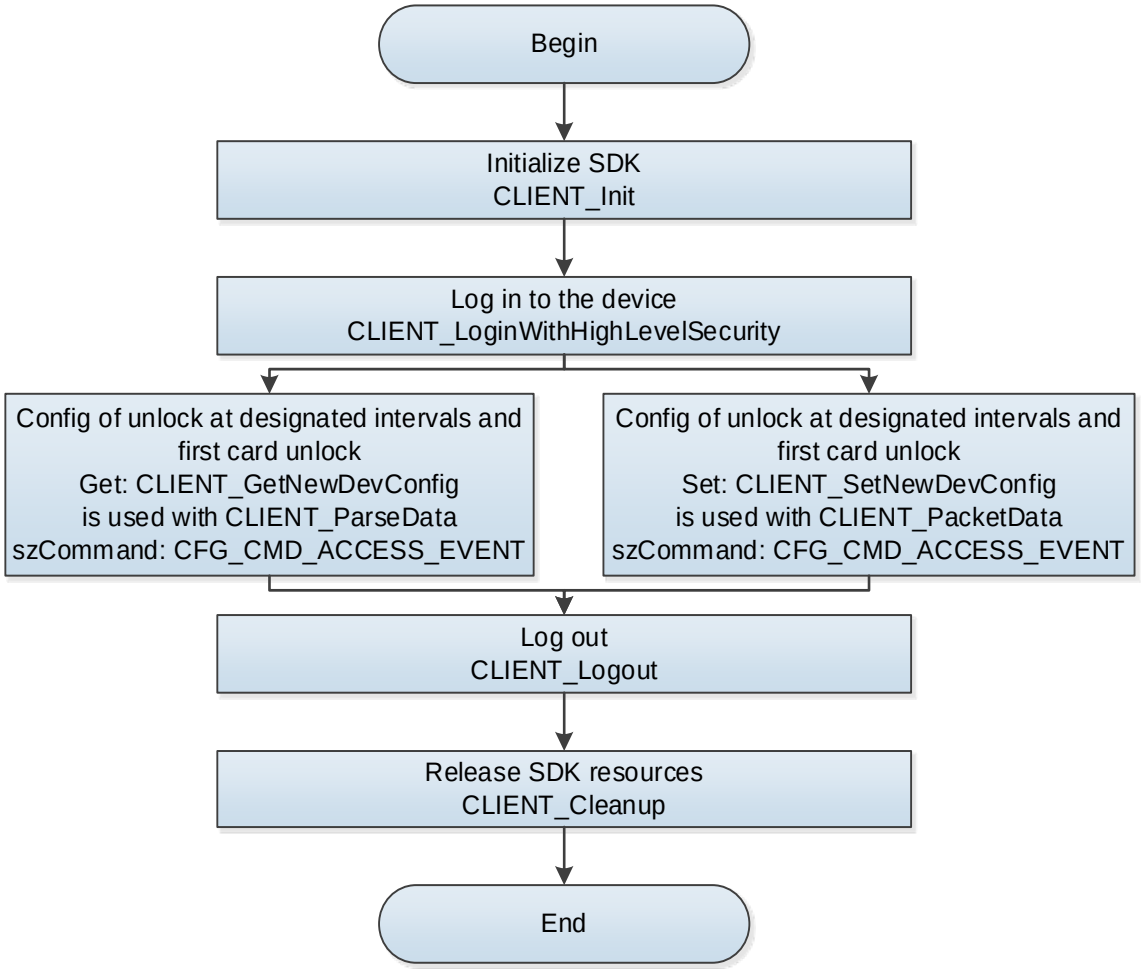
Table 2-43 Description of interfaces for unlock at designated intervals and first card unlock

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.

Interface	Description
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.10.1.3 Process Description

Figure 2-33 Unlock at designated intervals and first card unlock



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access info of unlock at designated intervals and first card unlock.

- szCommand: CFG_CMD_ACCESS_EVENT.
- pBuf: CFG_ACCESS_EVENT_INFO.

Table 2-44 Description of CFG_ACCESS_EVENT_INFO

CFG_ACCESS_EVENT_INFO	Description
stuDoorTimeSection	Config of unlock at designated intervals
stuFirstEnterInfo	First user/first card unlock config

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** in pairs to set the access info of unlock at designated intervals and first card unlock.

- szCommand: CFG_CMD_ACCESS_EVENT.
- pBuf: CFG_ACCESS_EVENT_INFO.

Table 2-45 Description of CFG_ACCESS_EVENT_INFO

CFG_ACCESS_EVENT_INFO	Description
stuDoorTimeSection	Config of unlock at designated intervals
stuFirstEnterInfo	First user/first card unlock config

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

Note

- User ID of first card refers to card number.
- To implement first card unlock function, add the person of the user ID to device and select the card as first card; otherwise, the first card unlock function cannot be used.

2.3.10.1.4 Example Code

```
//Get config information of unlock at designated intervals and first card/first user unlock
char * szOut1 = new char[1024*32];
    CFG_ACCESS_EVENT_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut1, 1024*32,
&nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESS_EVENT, szOut1, &stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), NULL);
        if (bRet1)
        {
            printf("whether it is first card/first user unlock: %d\n",stOut2.stuFirstEnterInfo.bEnable);
            printf("Access status after passing first card permission
verification: %d\n",stOut2.stuFirstEnterInfo.emStatus);
            printf("Periods that need first card verification: %d\n", stOut2.stuFirstEnterInfo.nTimeIndex);
        }
    }
    else{
        printf("parse failed!!!");
    }
    char * szOut = new char[1024*32];
    //First user/first card unlock config
    stOut2.stuFirstEnterInfo.bEnable = TRUE;
    stOut2.stuFirstEnterInfo.emStatus = ACCESS_FIRSTENTER_STATUS_KEEPOPEN;
    stOut2.stuFirstEnterInfo.nTimeIndex = 0;
    //Config of unlock at designated intervals
    stOut2.stuDoorTimeSection[0][0].emDoorOpenMethod = CFG_DOOR_OPEN_METHOD_PWD_ONLY;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuStartTime.dwHour = 9;
```

```
    stOut2.stuDoorTimeSection[0][0].stuTime.stuStartTime.dwMinute = 11;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuStartTime.dwSecond = 45;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuEndTime.dwHour = 19;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuEndTime.dwMinute = 11;
    stOut2.stuDoorTimeSection[0][0].stuTime.stuEndTime.dwSecond = 45;

    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESS_EVENT, (char *)&stOut2,
sizeof(CFG_ACCESS_EVENT_INFO), szOut, 1024*32);
    if(bRet2)
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_EVENT, 0, szOut,
1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.10.2 Combination Unlock by Multiple Persons

2.3.10.2.1 Introduction

For combination unlock by multiple persons, you can call SDK interface to get and set the config of combination unlock by multiple persons of the access control device.

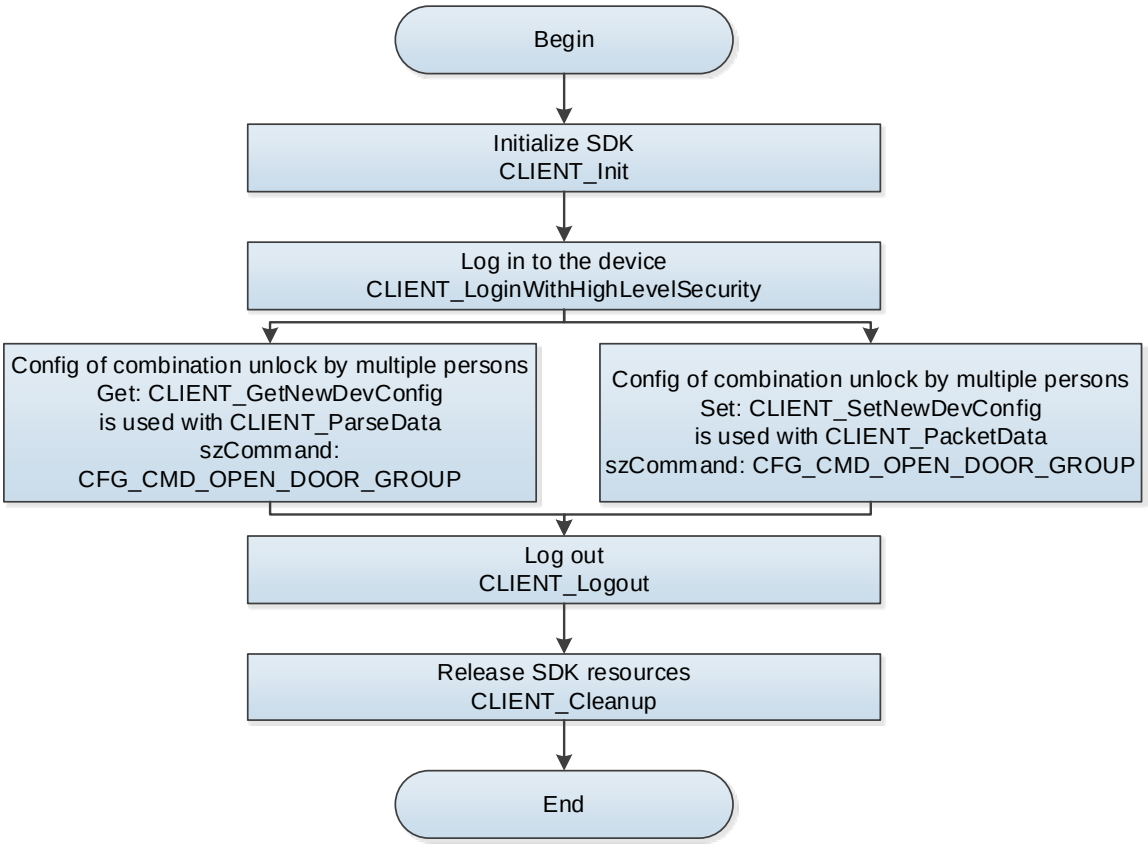
2.3.10.2.2 Interface Overview

Table 2-46 Description of interfaces for combination unlock by multiple persons

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.10.2.3 Process Description

Figure 2-34 Combination unlock by multiple persons



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access info of combination unlock by multiple persons

- szCommand: CFG_CMD_OPEN_DOOR_GROUP.
- pBuf: CFG_OPEN_DOOR_GROUP_INFO.

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** to set the access info of combination unlock by multiple persons.

- szCommand: CFG_CMD_OPEN_DOOR_GROUP.
- pBuf: CFG_OPEN_DOOR_GROUP_INFO.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

Note

- Before configuring combination unlock by multiple persons, add personnel to the device.
- Combination number: Group the personnel, and one door can configure up to 4 personnel groups.
- Personnel group: Person within the group and one group has up to 50 persons who should be added to device in advance.

- Number of valid persons: Should be less than or equal to the current number of persons in the group, and the total number of valid persons for one door is less than or equal to five persons.
- Set the unlock method for the personnel group: You can select from card or fingerprint.

2.3.10.2.4 Example Code

```
char * szOut1 = new char[1024*32];
    CFG_OPEN_DOOR_GROUP_INFO stOut2 = {sizeof(stOut2)};
    int nCount;
    CFG_OPEN_DOOR_GROUP_DETAIL* pstGroupDetail = new CFG_OPEN_DOOR_GROUP_DETAIL[nCount];
    if (NULL == pstGroupDetail)
    {
        return;
    }
    memset(pstGroupDetail, 0, sizeof(CFG_OPEN_DOOR_GROUP_DETAIL)*nCount);

    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_OPEN_DOOR_GROUP, 0, szOut1,
1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_OPEN_DOOR_GROUP, szOut1, &stOut2,
sizeof(CFG_OPEN_DOOR_GROUP_INFO), NULL);
        if (bRet1)
        {
            printf("number of valid combinations：%d\n",stOut2.nGroup);

            for (int i = 0; i < stOut2.nGroup; i++)
            {
                printf("[%d]group classification enabling:%d\n", i, stOut2.stuGroupInfo[i].bGroupDetailEx);
                printf("[%d]number of users: %d\n", i, stOut2.stuGroupInfo[i].nUserCount);
                printf("[%d]detailed maximum number of groups of combination unlock by multiple
persons: %d\n", i, stOut2.stuGroupInfo[i].nMaxGroupDetailNum);
                if (stOut2.stuGroupInfo[i].nMaxGroupDetailNum >
CFG_MAX_OPEN_DOOR_GROUP_DETAIL_NUM)
                {
                    for (int m = 0; m < stOut2.stuGroupInfo[i].nMaxGroupDetailNum; m++)
                    {
                        printf("[%d]-[%d]Method:%d\n", i, m,
stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].emMethod);
                        printf("[%d]-[%d]MethodExNum:%d\n", i, m,
stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].nMethodExNum);
```

```
                                for (int n = 0; n < stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].nMethodExNum;
n++)
                                {
                                        printf("[%d]-[%d]-[%d]MethodEx:%d\n", i, m, n,
stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].emMethodEx);
                                }
                                printf("[%d]-[%d]UserID:%s\n", i, m,
stOut2.stuGroupInfo[i].pstuGroupDetailEx[m].szUserID);
                            }
                        }
                        printf("[%d]GroupNum：%d\n", i, stOut2.stuGroupInfo[i].nGroupNum);
                        for (int j = 0; j < stOut2.stuGroupInfo[i].nGroupNum; j++)
                        {
                            printf("[%d],[%d]user ID：%s\n", i, j, stOut2.stuGroupInfo[i].stuGroupDetail[j].szUserID);
                        }

                    }
                }
            }
        else{
            printf("parse failed!!!");
        }
        char * szOut = new char[1024*32];

        stOut2.nGroup = 1;
        stOut2.stuGroupInfo[0].bGroupDetailEx = FALSE;
        stOut2.stuGroupInfo[0].nGroupNum = 1;
        stOut2.stuGroupInfo[0].stuGroupDetail[0].emMethod = EM_CFG_OPEN_DOOR_GROUP_METHOD_ANY;

        BOOL bRet2 = CLIENT_PacketData(CFG_CMD_OPEN_DOOR_GROUP, (char *)&stOut2,
sizeof(CFG_OPEN_DOOR_GROUP_INFO), szOut, 1024*32);
        if(bRet2)
        {
            BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_OPEN_DOOR_GROUP, 0, szOut,
1024*32, NULL, NULL, 3000);
            if (bRet3)
            {
                printf("CLIENT_SetNewDevConfig Success!\n");
            }
            else{
```

```
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.10.3 Inter-door Lock

2.3.10.3.1 Introduction

For inter-door lock config, you can call SDK interface to get and set the inter-door lock config of the access control device.

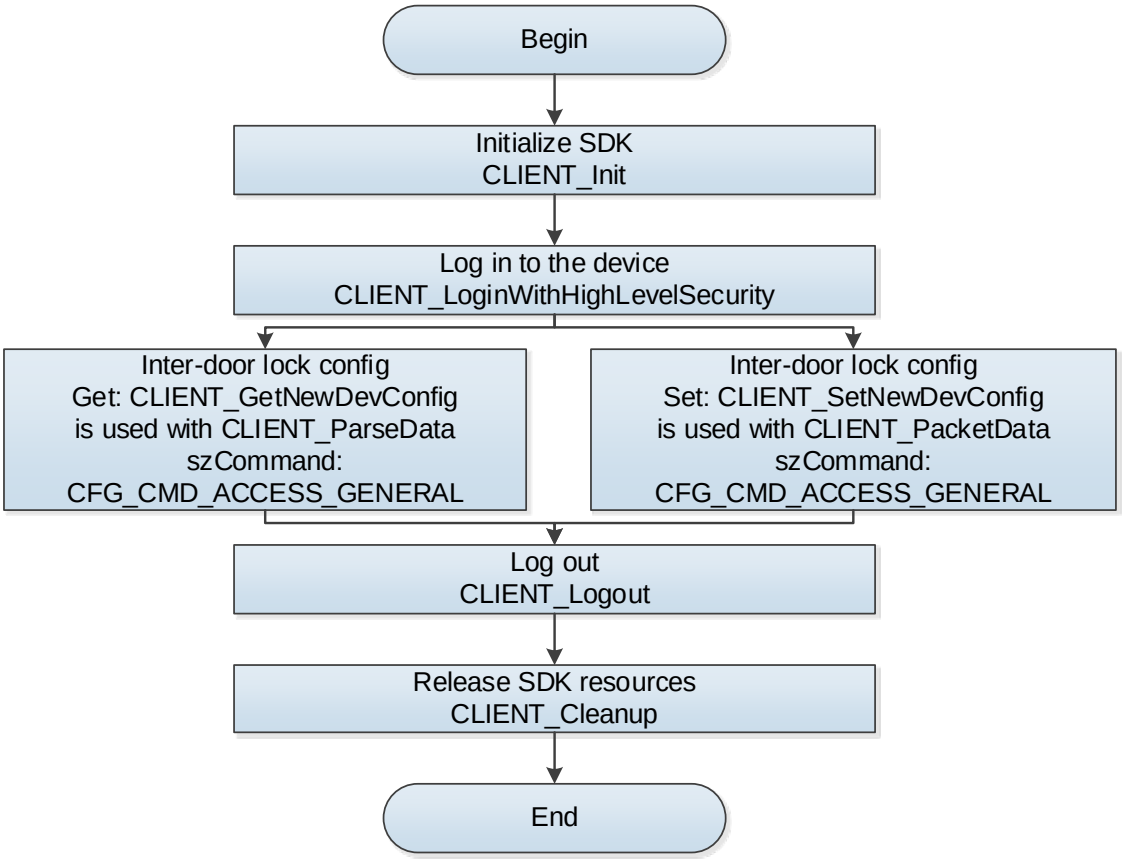
2.3.10.3.2 Interface Overview

Table 2-47 Description of inter-door lock interfaces

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.10.3.3 Process Description

Figure 2-35 Inter-door lock config



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access inter-door lock info.

- szCommand: CFG_CMD_ACCESS_GENERAL.
- pBuf: CFG_ACCESS_GENERAL_INFO.

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** to set the access inter-door lock info.

- szCommand: CFG_CMD_ACCESS_GENERAL.
- pBuf: CFG_ACCESS_GENERAL_INFO.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

Note

One device supports only one inter-door lock scheme.

2.3.10.3.4 Example Code

```
//Get inter-door lock config information
char * szOut1 = new char[1024*32];
```

```
    CFG_ACCESS_GENERAL_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_GENERAL, 0, szOut1,
1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_ACCESS_GENERAL, szOut1, &stOut2,
sizeof(CFG_ACCESS_GENERAL_INFO), NULL);
        if (bRet1)
        {
            printf("enabling: %d\n",stOut2.stuABLockInfo.bEnable);
            printf("number of valid interlock groups: %d\n",stOut2.stuABLockInfo.nDoors);

            for (int i = 0; i < stOut2.stuABLockInfo.nDoors; i++)
            {
                printf("[%d]number of valid interlock doors: %d\n", i,
stOut2.stuABLockInfo.stuDoors[i].nDoor);
                for (int j = 0; j < stOut2.stuABLockInfo.stuDoors[i].nDoor; j++)
                {
                    printf("[%d],[%d]channel number for interlock door: %d\n", i, j,
stOut2.stuABLockInfo.stuDoors[i].anDoor[j]);
                }
            }
        }
    }
    else{
        printf("parse failed!!!");
    }
//Set inter-door lock config information
    char * szOut = new char[1024*32];

    stOut2.stuABLockInfo.bEnable = TRUE;
    stOut2.stuABLockInfo.nDoors = 1;
    stOut2.stuABLockInfo.stuDoors[0].nDoor = 2;
    stOut2.stuABLockInfo.stuDoors[0].anDoor[0] = 0;
    stOut2.stuABLockInfo.stuDoors[0].anDoor[0] = 1;

    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_ACCESS_GENERAL, (char *)&stOut2,
sizeof(CFG_ACCESS_GENERAL_INFO), szOut, 1024*32);
    if(bRet2)
    {
```

```
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_ACCESS_GENERAL, 0, szOut,
1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.10.4 Anti-passback

2.3.10.4.1 Introduction

For anti-passback config, you can call SDK interface to get and set the anti-passback config of the access control device.

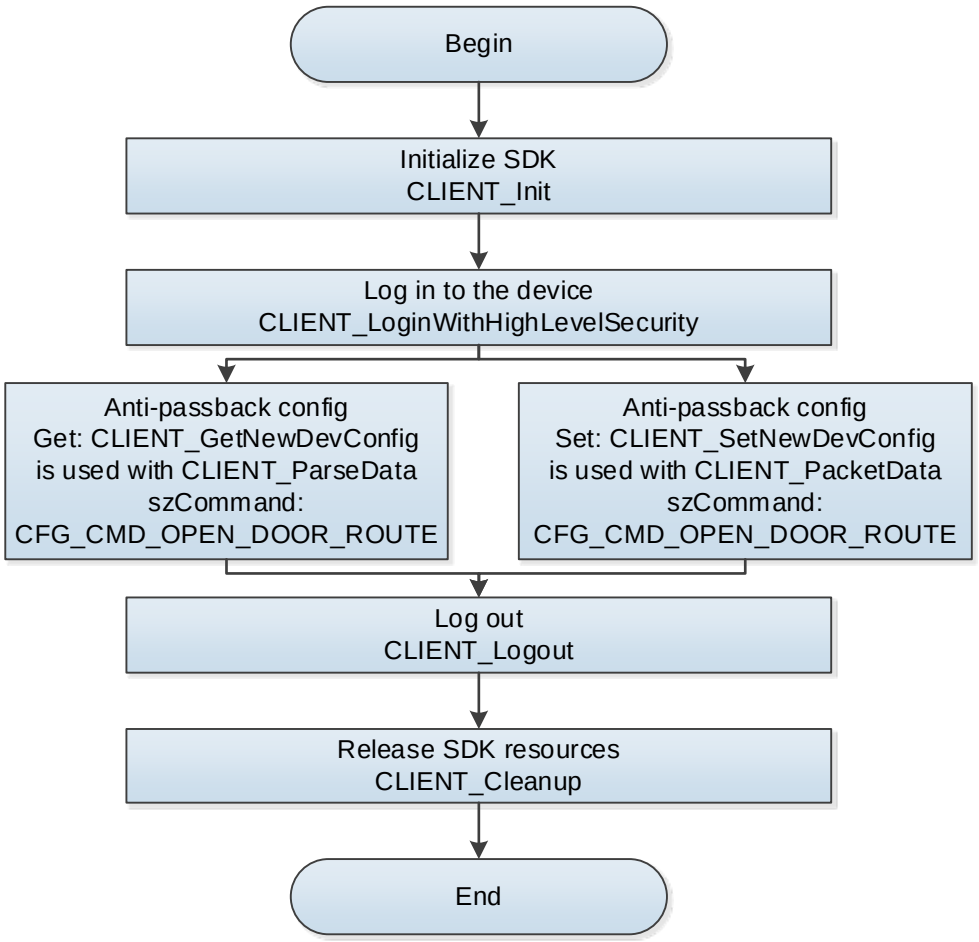
2.3.10.4.2 Interface Overview

Table 2-48 Description of anti-passback interfaces

Interface	Description
CLIENT_GetNewDevConfig	Query config information.
CLIENT_ParseData	Parse the queried config information.
CLIENT_SetNewDevConfig	Set config information.
CLIENT_PacketData	Pack the config information to be set into the string format.

2.3.10.4.3 Process Description

Figure 2-36 Anti-passback config



Process

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

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

Step 3 Call **CLIENT_GetNewDevConfig** and **CLIENT_ParseData** to query the access anti-passback info.

- szCommand: CFG_CMD_OPEN_DOOR_ROUTE.
- pBuf: CFG_OPEN_DOOR_ROUTE_INFO.

Step 4 Call **CLIENT_SetNewDevConfig** and **CLIENT_PacketData** to set the access anti-passback info.

- szCommand: CFG_CMD_OPEN_DOOR_ROUTE.
- pBuf: CFG_OPEN_DOOR_ROUTE_INFO.

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

Note

One device supports only one anti-passback scheme.

2.3.10.4.4 Example Code

```
//Get anti-passback config information
```

```
char * szOut1 = new char[1024*32];
    CFG_OPEN_DOOR_ROUTE_INFO stOut2 = {sizeof(stOut2)};
    int nError = 0;
    BOOL bRet = CLIENT_GetNewDevConfig(g_lLoginHandle, CFG_CMD_OPEN_DOOR_ROUTE, 0, szOut1,
1024*32, &nError, 3000);
    if(bRet){
        BOOL bRet1 = CLIENT_ParseData(CFG_CMD_OPEN_DOOR_ROUTE, szOut1, &stOut2,
sizeof(CFG_OPEN_DOOR_ROUTE_INFO), NULL);
        if (bRet1)
        {
            printf("passback reset time: %d\n",stOut2.nResetTime);
            printf("number of door lists: %d\n",stOut2.nDoorList);
            printf("period corresponding to passback path: %d\n",stOut2.nOpenAlwaysTimeIndex);

            for (int i = 0; i < stOut2.nDoorList; i++)
            {
                printf("[%d]passback reset time: %d\n", i, stOut2.stuDoorList[i].nResetTime);
                printf("[%d]number of valid nodes for unlock routes: %d\n", i, stOut2.stuDoorList[i].nDoors);
                for (int j = 0; j < stOut2.stuDoorList[i].nDoors; j++)
                {
                    printf("[%d],[%d]Card reader ID：  %s\n", i, j,
stOut2.stuDoorList[i].stuDoors[j].szReaderID);
                }
            }
        }
    }
    else{
        printf("parse failed!!!");
    }
//Configure anti-passback config information
    char * szOut = new char[1024*32];

    stOut2.nDoorList = 1;
    stOut2.nResetTime = 1;
    stOut2.nTimeSection = 2;
    stOut2.stuDoorList[0].nResetTime = 0;

    BOOL bRet2 = CLIENT_PacketData(CFG_CMD_OPEN_DOOR_ROUTE, (char *)&stOut2,
sizeof(CFG_OPEN_DOOR_ROUTE_INFO), szOut, 1024*32);
    if(bRet2)
```

```
    {
        BOOL bRet3 = CLIENT_SetNewDevConfig(g_lLoginHandle, CFG_CMD_OPEN_DOOR_ROUTE, 0, szOut,
1024*32, NULL, NULL, 3000);
        if (bRet3)
        {
            printf("CLIENT_SetNewDevConfig Success!\n");
        }
        else{
            printf("CLIENT_SetNewDevConfig failed! Last Error[%x]\n", CLIENT_GetLastError());
        }
    }
    else{
        printf("CLIENT_PacketData failed! Last Error[%x]\n", CLIENT_GetLastError());
    }
```

2.3.10.5 Unlock Password

2.3.10.5.1 Introduction

For unlock password, you can call SDK interface to add, delete, query and modify the unlock password of the access control device.

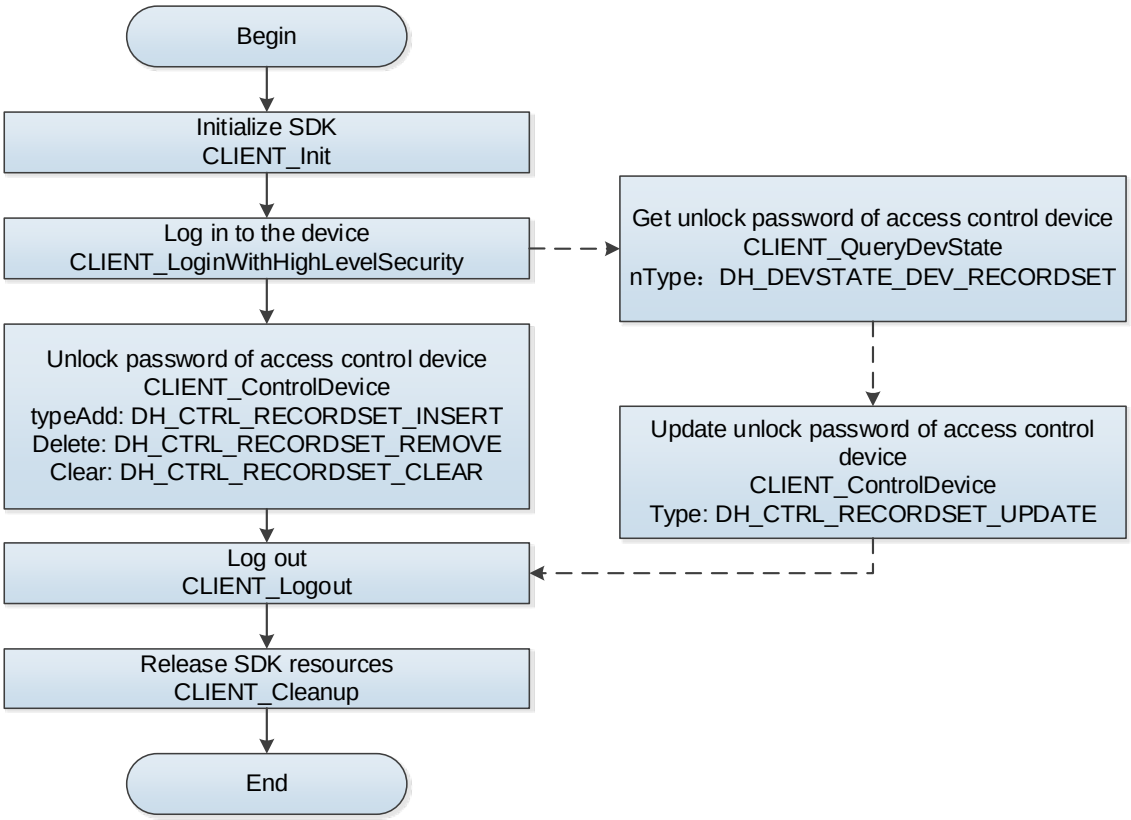
2.3.10.5.2 Interface Overview

Table 2-49 Description of unlock password interface

Interface	Description
CLIENT_ControlDevice	Device control.

2.3.10.5.3 Process Description

Figure 2-37 Unlock password config



Process

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

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

Step 3 Call the **CLIENT_ControlDevice** to operate unlock password information.

Table 2-50 Description and structure of type

Type	Description	emType	Param
DH_CTRL_RECORDSET_INSERT	Add unlock password	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_INSERT_PARA NET_RECORDSET_ACCESS_CTL_PWD
DH_CTRL_RECORDSET_REMOVE	Delete unlock password	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_PARAM NET_RECORDSET_ACCESS_CTL_PWD
DH_CTRL_RECORDSET_CLEAR	Clear unlock password	NET_RECORD_ACCESSCTLPWD	NET_CTRL_RECORDSET_PARAM

Step 4 Call the **CLIENT_QueryDevState** interface to get unlock password information.

Table 2-51 Description and structure of type

Type	Description	emType	Param
DH_DEVSTATE_DEV_RECORDSET	Get unlock password	NET_RECORD_ACCESSCTLPWD	• NET_CTRL_RECORDSET_PARAM • NET_RECORDSET_ACCESS_CTL_PWD

Step 5 Call the **CLIENT_ControlDevice** to update unlock password information.

Table 2-52 Description and structure of type

Type	Description	emType	Param
DH_CTRL_RECORDSET_UPDATE	Get unlock password	NET_RECORD_ACCESSCTLPWD	● NET_CTRL_RECORDSET_PARAM ● NET_RECORDSET_ACCESS_CTL_PWD

Step 6 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

Note

- Before configuring combination unlock by multiple persons, add personnel to the device.
- User number: Personnel card number.

2.3.10.5.4 Example Code

```
NET_RECORDSET_ACCESS_CTL_PWD stuInfo = {sizeof(stuInfo)};

    //Add
    stuInfo.bNewDoor = TRUE;
    stuInfo.nDoorNum = 2;
    stuInfo.sznDoors[0] = 1223;

    memcpy(stuInfo.szUserID, "11234", sizeof(stuInfo.szUserID));
    memcpy(stuInfo.szDoorOpenPwd, "12345", sizeof(stuInfo.szDoorOpenPwd));

    NET_CTRL_RECORDSET_INSERT_PARAM stuParam = {sizeof(stuParam)};
    stuParam.stuCtrlRecordSetInfo.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_IN);
    stuParam.stuCtrlRecordSetInfo.emType = NET_RECORD_ACCESSCTLPWD;
    stuParam.stuCtrlRecordSetInfo.pBuf = (void*)&stuInfo;
    stuParam.stuCtrlRecordSetInfo.nBufLen = sizeof(stuInfo);

    stuParam.stuCtrlRecordSetResult.dwSize = sizeof(NET_CTRL_RECORDSET_INSERT_OUT);

    BOOL bRet = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_INSERT, &stuParam, 5000);
    //Get and update
    stuInfo.nRecNo = 123456;
    NET_CTRL_RECORDSET_PARAM stuParam2 = {sizeof(stuParam2)};
    stuParam2.emType = NET_RECORD_ACCESSCTLPWD;
    stuParam2.pBuf = (void*)&stuInfo;
    stuParam2.nBufLen = sizeof(stuInfo);
```

```
    int nRet = 0;
    BOOL bRet1 = CLIENT_QueryDevState(g_lLoginHandle, DH_DEVSTATE_DEV_RECORDSET,
(char*)&stuParam2,
        sizeof(stuParam2), &nRet, 5000);
    if (bRet)
    {

        stuParam2.emType = NET_RECORD_ACCESSCTLPWD;
        stuParam2.pBuf = (void*)&stuInfo;
        stuParam2.nBufLen = sizeof(stuInfo);

        // update info
        BOOL bRet2 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_UPDATE, &stuParam2,
5000);
    }
    else{
        printf("CLIENT_QueryDevState failed!\n");
    }
    //Delete
    stuInfo.nRecNo = 123456;
    NET_CTRL_RECORDSET_PARAM stuParam3 = {sizeof(stuParam3)};
    stuParam3.emType = NET_RECORD_ACCESSCTLPWD;
    stuParam3.pBuf = (void*)&stuInfo.nRecNo;
    stuParam3.nBufLen = sizeof(stuInfo.nRecNo);

    BOOL bRet3 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_REMOVE, &stuParam3,
5000);
    //Clear
    NET_CTRL_RECORDSET_PARAM stuParam4 = {sizeof(stuParam4)};
    stuParam4.emType = NET_RECORD_ACCESSCTLPWD;
    BOOL bRet4 = CLIENT_ControlDevice(g_lLoginHandle, DH_CTRL_RECORDSET_CLEAR, &stuParam4, 5000);
```

2.3.11 Records Query

2.3.11.1 Unlock Records

2.3.11.1.1 Introduction

For unlock records query, you can call SDK interface to query the unlock records of the access control device. You can set query conditions and number of query entries.

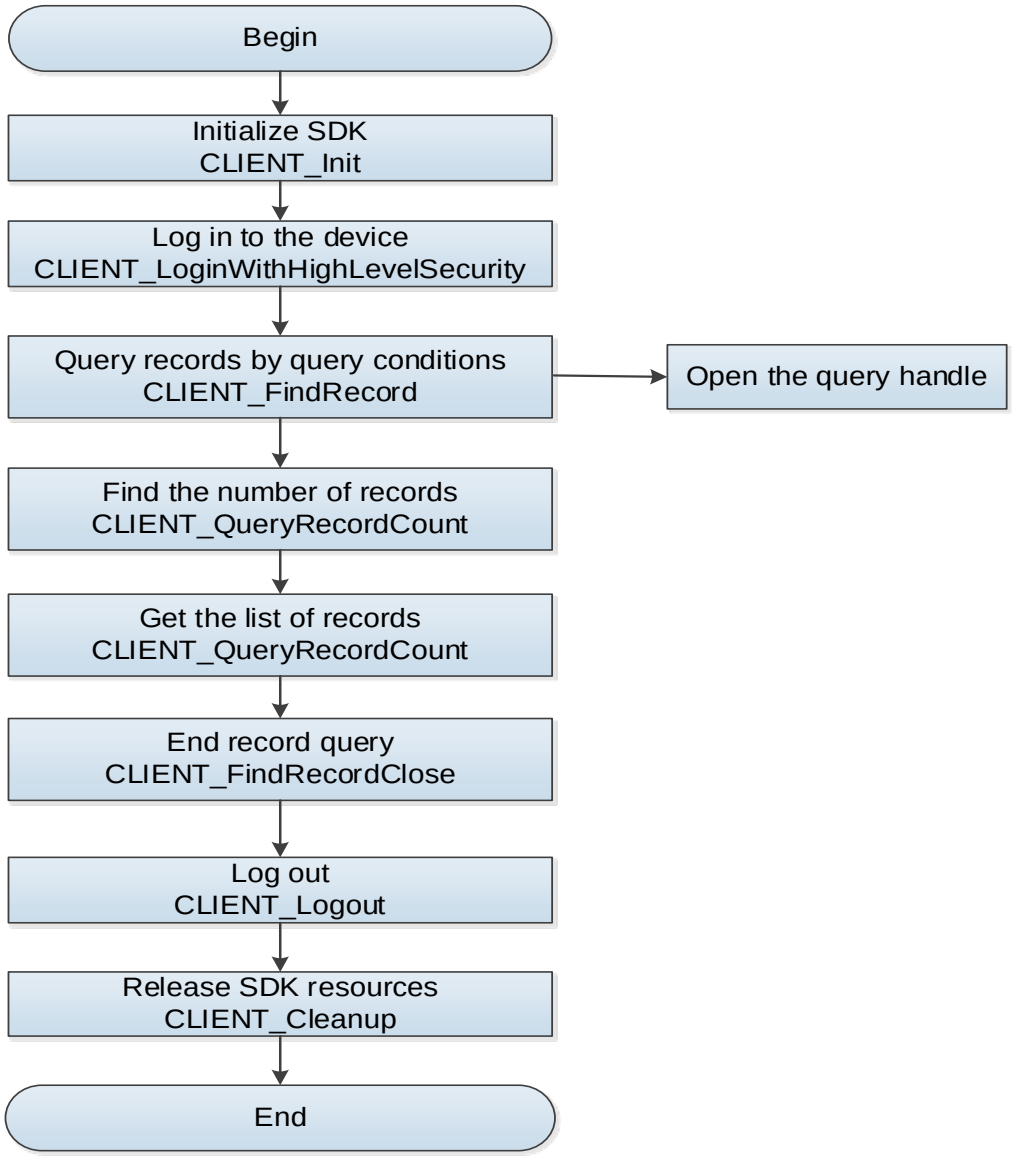
2.3.11.1.2 Interface Overview

Table 2-53 Description of record query interfaces

Interface	Description
CLIENT_QueryRecordCount	Find the count of records.
CLIENT_FindRecord	Query records by query conditions.
CLIENT_FindNextRecord	Find records: View the count of files to be required by nFilecount. When the return value is the count of media files and less than nFilecount, the query of files is completed within the corresponding period.
CLIENT_FindRecordClose	End record query.

2.3.11.1.3 Process Description

Figure 2-38 Record query



Process

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

Step 2 Call the **CLIENT_LoginWithHighLevelSecurity** to log in to the device.
Step 3 Call the **CLIENT_FindRecord** to get the query handle.
emType unlock record: NET_RECORD_ACCESSCTLCARDREC.
Step 4 Call the **CLIENT_QueryRecordCount** to find the count of records.
Step 5 Cal the **CLIENT_FindNextRecord** to get the list of records.
Step 6 After query, call **CLIENT_FindRecordClose** to close the query handle.
Step 7 After completing this process, call the **CLIENT_Logout** to log out of the device.
Step 8 After using all SDK functions, call the **CLIENT_Cleanup** to release SDK resources.

2.3.11.1.4 Example Code

```
NET_IN_FIND_RECORD_PARAM stuIn = {sizeof(stuIn)};
    NET_OUT_FIND_RECORD_PARAM stuOut = {sizeof(stuOut)};

    stuIn.emType = NET_RECORD_ACCESS_ALARMRECORD;

    if (CLIENT_FindRecord(g_lLoginHandle, &stuIn, &stuOut, 5000))
    {
        printf("CLIENT_FindRecord success!\n");
    }
    else{
        printf("CLIENT_FindRecord failed!\n");
    }

    NET_IN_QUEYT_RECORD_COUNT_PARAM stuInCount = {sizeof(stuInCount)};
    stuInCount.lFindeHandle = stuOut.lFindeHandle;
    NET_OUT_QUEYT_RECORD_COUNT_PARAM stuOutCount = {sizeof(stuOutCount)};
    if (CLIENT_QueryRecordCount(&stuInCount, &stuOutCount, 5000))
    {
        printf("CLIENT_QueryRecordCount success!\n");
    }
    else{
        printf("CLIENT_QueryRecordCount failed!\n");
    }

    int i = 0, j = 0;
    int nMaxNum = 10;
    NET_IN_FIND_NEXT_RECORD_PARAM stuIn1 = {sizeof(stuIn1)};
    stuIn1.lFindeHandle = stuOut.lFindeHandle;
    stuIn1.nFileCount = nMaxNum;

    NET_OUT_FIND_NEXT_RECORD_PARAM stuOut2 = {sizeof(stuOut2)};
```

```
    stuOut2.nMaxRecordNum = nMaxNum;

    NET_RECORDSET_ACCESS_CTL_CARD* pstuCard = new
NET_RECORDSET_ACCESS_CTL_CARD[nMaxNum];
    if (NULL == pstuCard)
    {
        return;
    }
    memset(pstuCard, 0, sizeof(NET_RECORDSET_ACCESS_CTL_CARD) * nMaxNum);

    for (i = 0; i < nMaxNum; i++)
    {
        pstuCard[i].dwSize = sizeof(NET_RECORDSET_ACCESS_CTL_CARD);
    }
    stuOut2.pRecordList = (void*)pstuCard;

    if (CLIENT_FindNextRecord(&stuIn1, &stuOut2, 5000) >= 0)
    {
        printf("CLIENT_FindNextRecord success!\n");
    }
    else
    {
        printf("CLIENT_FindNextRecord failed!\n");
    }
CLIENT_FindRecordClose(stuOut.lFindeHandle);
```

2.3.11.2 Device log

2.3.11.2.1 Introduction

For device log, you can call SDK interface to query the operation log of the access control device by specifying the log type or the number of queries, or query by pages.

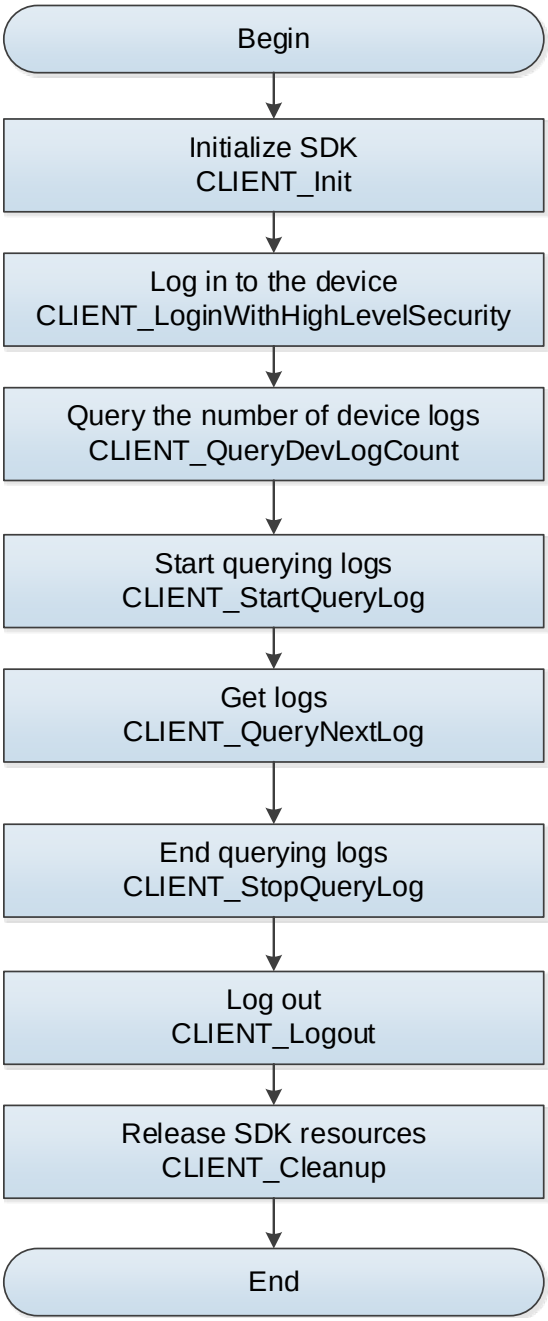
2.3.11.2.2 Interface Overview

Table 2-54 Description of device log interfaces

Interface	Description
CLIENT_QueryDevLogCount	Query the count of device logs.
CLIENT_StartQueryLog	Start querying logs.
CLIENT_QueryNextLog	Get logs.
CLIENT_StopQueryLog	Stop querying logs.

2.3.11.2.3 Process Description

Figure 2-39 Device log



Process

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

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

Step 3 Call the **CLIENT_QueryDevLogCount** to set the number of queried logs.

Step 4 Call the **CLIENT_StartQueryLog** to start querying log information.

- pInParam: NET_IN_START_QUERYLOG.
- pOutParam: NET_OUT_START_QUERYLOG.

Step 5 Call the **CLIENT_QueryNextLog** to get log information.

- pInParam: NET_IN_QUERYNEXTLOG.
- pOutParam: NET_OUT_QUERYNEXTLOG.

Step 6 Call the **CLIENT_StopQueryLog** to stop querying logs.

Step 7 After completing this process, call the **CLIENT_Logout** to log out of the device.

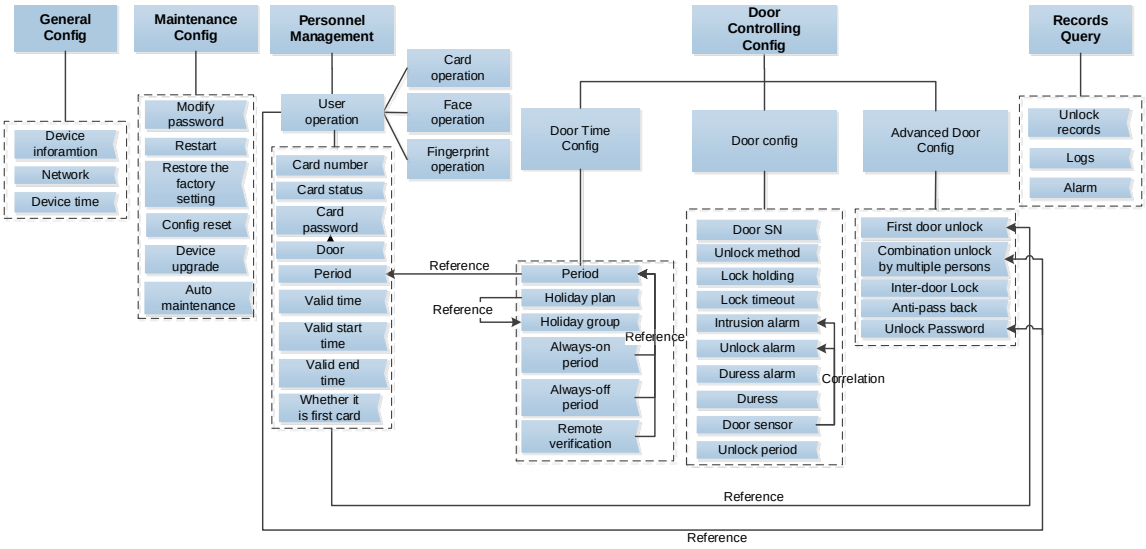
Step 8 After using all SDK functions, call the **CLIENT_Cleanup** to release SDK resources.

2.3.11.2.4 Example Code

```
//Start querying log information
NET_IN_START_QUERYLOG stuIn = {sizeof(stuIn)};
NET_OUT_START_QUERYLOG stuOut = {sizeof(stuOut)};
LLONG lLogID = CLIENT_StartQueryLog(m_lLoginId, &stuIn, &stuOut, 5000);
//Get log information
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 querying log information
BOOL bRet0 = CLIENT_StopQueryLog(m_lLogID);
```

2.4 Access Controller/All-in-one Face Machine (Second-Generation)

Figure 2-40 Function calling relationship



Here are the meanings of reference and correlation.

- Reference: The function pointed by the end point of the arrow refers to the function pointed by the start point of the arrow.
- Correlation: Whether the function started by the arrow can be used normally is related to the function configuration pointed by the end point of the arrow.

2.4.1 Access Control

See "2.3.1 Access Control."

2.4.2 Alarm Event

See "2.3.2 Alarm Event."

2.4.3 Viewing Device Information

2.4.3.1 Capability Set Query

2.4.3.1.1 Introduction

The process to view device information is that, you issue a command through SDK to the access control device, to get the capability of another device.

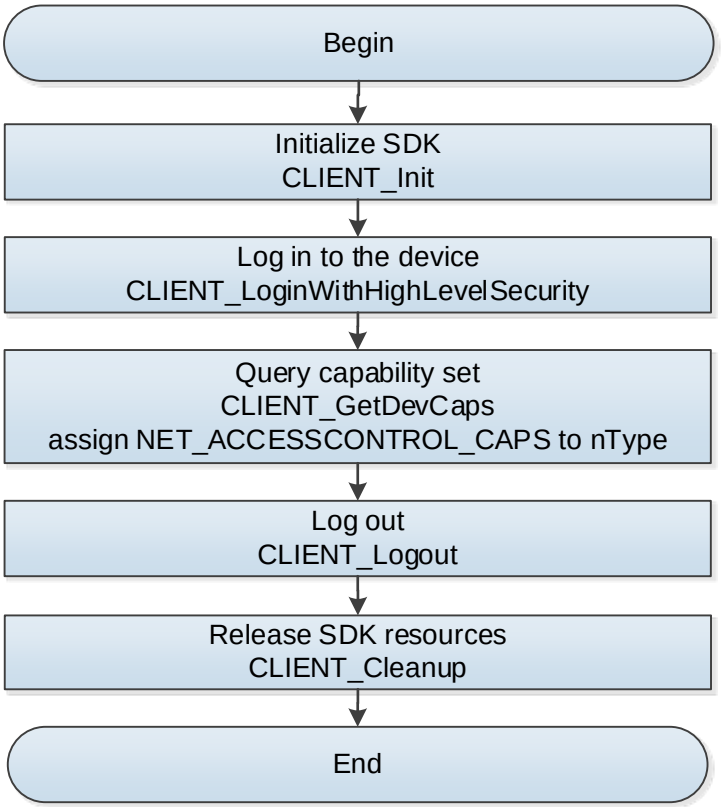
2.4.3.1.2 Interface Overview

Table 2-55 Description of capability set query interface

Interface	Description
CLIENT_GetDevCaps	Get the access control capability (sucha as access control, user, card, face, and fingerprint).

2.4.3.1.3 Process Description

Figure 2-41 Device information viewing



Process

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

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

Step 3 Call **CLIENT_GetDevCaps** and assign **NET_ACCESSCONTROL_CAPS** to nType, to get the access control.

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.4.3.1.4 Example Code

```
NET_IN_AC_CAPS stuIn = {sizeof(stuIn)};
    NET_OUT_AC_CAPS stuOut = {sizeof(stuOut)};
    BOOL bRet = CLIENT_GetDevCaps(m_lLoginID, NET_ACCESSCONTROL_CAPS,&stuIn, &stuOut,5000);
    if (bRet)
    {
        NET_ACCESS_USER_CAPS      stuUserCaps = stuOut.stuUserCaps;
```

```
}
else
{
        return FALSE;
}
```

2.4.3.2 Viewing Device Version and MAC

See "2.3.3.2 Viewing Device Version and MAC."

2.4.4 Network Setting

See "2.3.4 Network Setting."

2.4.5 Setting the Device Time

See "2.3.5 Device Time Setting."

2.4.6 Maintenance Config

See "2.3.6 Maintenance Config."

2.4.7 Personnel Management

2.4.7.1 User Management

2.4.7.1.1 Introduction

Call SDK to add, delete, and query the user info fields of the access controllers (including user ID, person name, type, status, ID card number, valid period, holiday plan, and user permission).

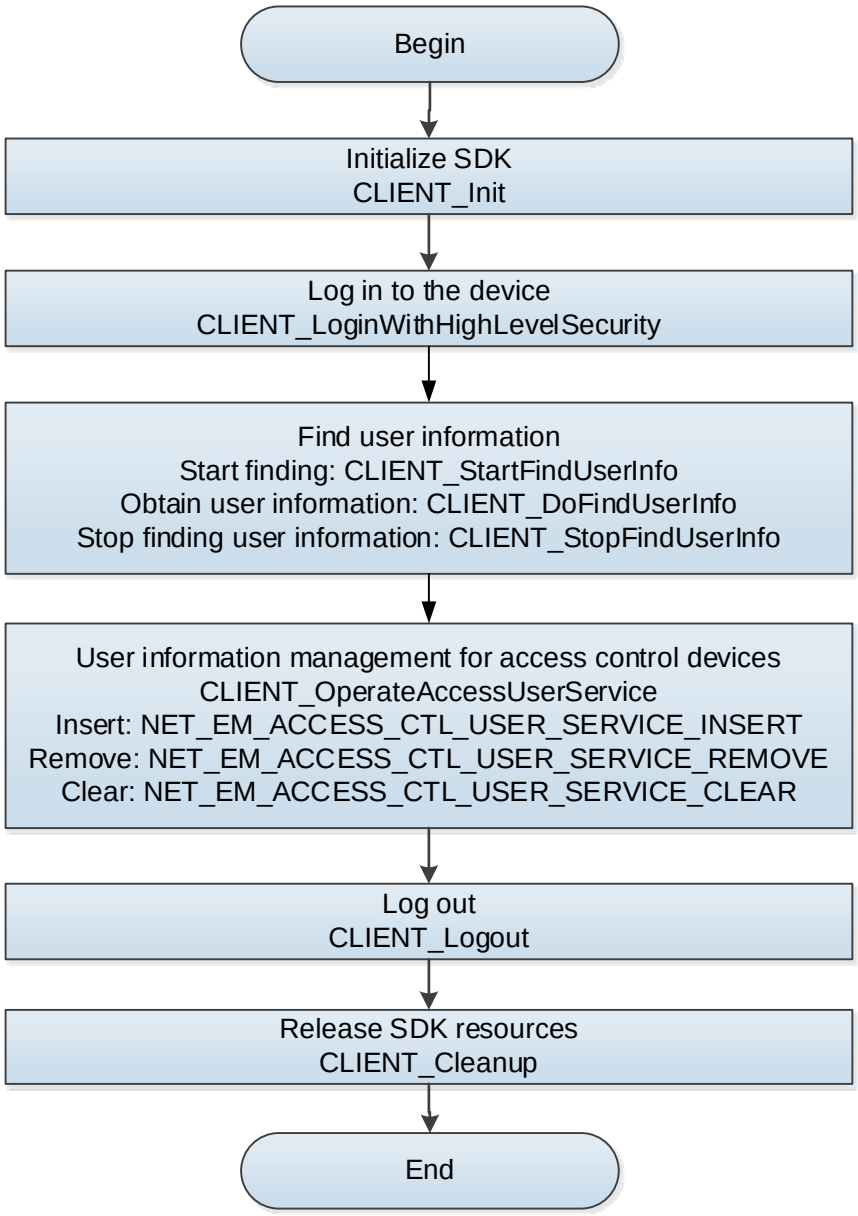
2.4.7.1.2 Interface Overview

Table 2-56 Description of user information interface

Interface	Description
CLIENT_OperateAccessUserService	User information management interface for access controllers.
CLIENT_StartFindUserInfo	Start to find the user information.
CLIENT_DoFindUserInfo	Obtain the user information.
CLIENT_StopFindUserInfo	Stop finding the user information.

2.4.7.1.3 Process Description

Figure 2-42 User info management



Process

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

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

Step 3 Call **CLIENT_StartFindUserInfo** to start finding the user information.

Step 4 Call **CLIENT_DoFindUserInfo** to obtain the user information.

Step 5 Call **CLIENT_StopFindUserInfo** to stop finding the user information.

Step 6 Call **CLIENT_OperateAccessUserService** to add, delete, and clear the user information

Step 7 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.4.7.1.4 Example Code

```
//Get
```

```
NET_IN_USERINFO_START_FIND stuStartIn = {sizeof(stuStartIn)};
    NET_OUT_USERINFO_START_FIND stuStartOut = {sizeof(stuStartOut)};
    stuStartOut.nTotalCount = 0;
    stuStartOut.nCapNum = 10;
    LLONG UserFindId = CLIENT_StartFindUserInfo(m_lLoginID, &stuStartIn, &stuStartOut,SDK_API_WAIT);
    if (UserFindId != NULL)
    {
        m_UserInfoVector.clear();
        //
        int nStartNo = 0;
        m_bIsDoFindNext = TRUE;
        while (m_bIsDoFindNext)
        {
            NET_ACCESS_USER_INFO* pUserInfo = new NET_ACCESS_USER_INFO[10];
            if (pUserInfo)
            {
                int nRecordNum = 0;
                NET_IN_USERINFO_DO_FIND stuFindIn = {sizeof(stuFindIn)};
                stuFindIn.nStartNo = nStartNo;
                stuFindIn.nCount = 10;
                NET_OUT_USERINFO_DO_FIND stuFindOut = {sizeof(stuFindOut)};
                stuFindOut.nMaxNum = 10;
                stuFindOut.pstuInfo = pstuAlarm;
                if (CLIENT_DoFindUserInfo(m_UserFindId, &stuFindIn, &stuFindOut, SDK_API_WAIT))
                {
                    if (stuFindOut.nRetNum > 0)
                    {
                        nRecordNum = stuFindOut.nRetNum;
                        m_bIsDoFindNext = TURE;
                    }
                }
                m_bIsDoFindNext = FALSE;
                for (int i=0;i<nRecordNum;i++)
                {
                    NET_ACCESS_USER_INFO stuUserInfo;
                    memset(&stuUserInfo,0,sizeof(NET_ACCESS_USER_INFO));
                    memcpy(&stuUserInfo,&pUserInfo[i],sizeof(NET_ACCESS_USER_INFO));
                    m_UserInfoVector.push_back(stuUserInfo);
                }
                nStartNo = nRecordNum;
```

```
                delete []pUserInfo;
                pUserInfo = NULL;
            }
            else
            {
                m_bIsDoFindNext = FALSE;
            }
        }
        CLIENT_StopFindUserInfo(m_UserFindId);
        return TRUE;
    }
    else
    {
        return FALSE;
    }

//Add
NET_ACCESS_USER_INFO stuUserInfo;
memset(&stuUserInfo,0,sizeof(NET_ACCESS_USER_INFO));
memcpy(&stuUserInfo,pstuUserInfo,sizeof(NET_ACCESS_USER_INFO));
NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
NET_IN_ACCESS_USER_SERVICE_INSERT stuUserInsertIn = {sizeof(stuUserInsertIn)};
    stuUserInsertIn.nInfoNum = nNum;
    stuUserInsertIn.pUserInfo = &stuUserInfo;
    NET_OUT_ACCESS_USER_SERVICE_INSERT stuUserINsertOut = {sizeof(stuUserINsertOut)};
    stuUserINsertOut.nMaxRetNum = nNum;
    stuUserINsertOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessUserService(m_lLoginID,
NET_EM_ACCESS_CTL_USER_SERVICE_INSERT, &stuUserInsertIn, &stuUserINsertOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//Delete
NET_IN_ACCESS_USER_SERVICE_REMOVE stuUserRemoveIn = {sizeof(stuUserRemoveIn)};
    stuUserRemoveIn.nUserNum = 1;
    NET_ACCESS_USER_INFO stuUserInfo;
    memset(&stuUserInfo,0,sizeof(NET_ACCESS_USER_INFO));
```

```
    memcpy(&stuUserInfo,&m_UserInfoVector[m_nUserInfoIndex],sizeof(NET_ACCESS_USER_INFO));
    strncpy(stuUserRemoveIn.szUserID[0],stuUserInfo.szUserID,DH_MAX_USERID_LEN-1);

    NET_OUT_ACCESS_USER_SERVICE_REMOVE stuUserRemoveOut = {sizeof(stuUserRemoveOut)};
    NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
    stuUserRemoveOut.nMaxRetNum = 1;
    stuUserRemoveOut.pFailCode = &stuFailCodeR;
BOOL bRet = CLIENT_OperateAccessUserService(m_lLoginID, NET_EM_ACCESS_CTL_USER_SERVICE_REMOVE,
stuUserRemoveIn, &stuUserRemoveOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
//Clear
NET_IN_ACCESS_USER_SERVICE_CLEAR stuUserClearIn = {sizeof(stuUserClearIn)};
    NET_OUT_ACCESS_USER_SERVICE_CLEAR stuUserClearOut = {sizeof(stuUserClearIn)};
    BOOL bRet = CLIENT_OperateAccessUserService(m_lLoginID,
NET_EM_ACCESS_CTL_USER_SERVICE_CLEAR, &stuUserClearIn, &stuUserClearOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
```

2.4.7.2 Card Management

2.4.7.2.1 Introduction

Call SDK to add, delete, query, and modify the card information fields of the access control device (including card number, user ID, and card type).

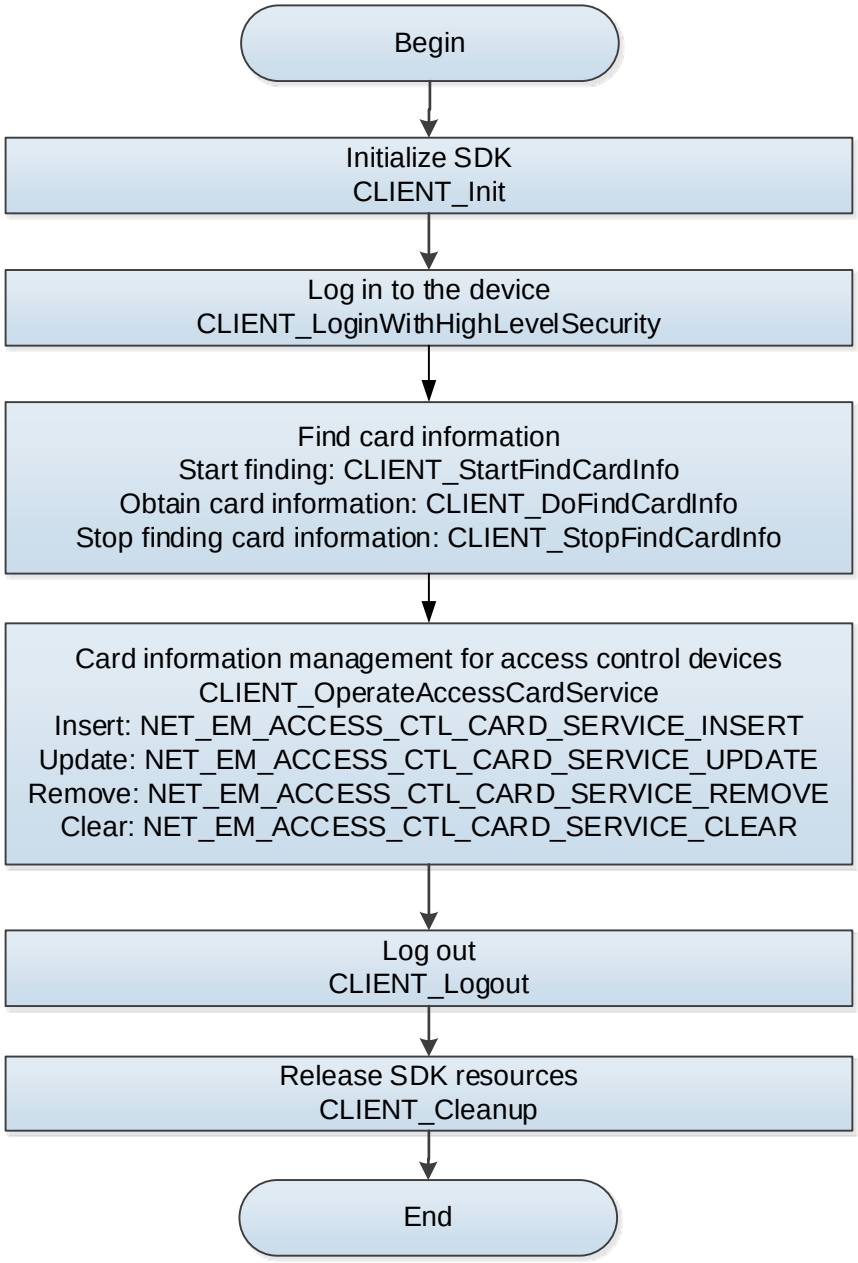
2.4.7.2.2 Interface Overview

Table 2-57 Description of card information interface

Interface	Description
CLIENT_OperateAccessCardService	Card information management interface for access control devices
CLIENT_StartFindCardInfo	Start to find the card information
CLIENT_DoFindCardInfo	Obtain the card information
CLIENT_StopFindCardInfo	Stop finding the card information

2.4.7.2.3 Process Description

Figure 2-43 Management of card information



Process

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

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

Step 3 Call **CLIENT_StartFindCardInfo** to start finding the card information.

Step 4 Call **CLIENT_DoFindCardInfo** to obtain the card information.

Step 5 Call **CLIENT_StopFindCardInfo** to stop finding the card information.

Step 6 Call **CLIENT_OperateAccessCardService** to add, update, delete, and clear the card information.

Step 7 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.4.7.2.4 Example Code

```
//Get
NET_IN_CARDINFO _START_FIND stuStartIn = {sizeof(stuStartIn)};
    NET_OUT_CARDINFO _START_FIND stuStartOut = {sizeof(stuStartOut)};
    stuStartOut.nTotalCount = 0;
    stuStartOut.nCapNum = 10;
    LLONG CardFindId = CLIENT_StartFindCardInfo(m_lLoginID, &stuStartIn, &stuStartOut,SDK_API_WAIT);
    if (CardFindId != NULL)
    {
        m_CardInfoVector.clear();
        //
        int nStartNo = 0;
        m_bIsDoFindNext = TRUE;
        while (m_bIsDoFindNext)
        {
            NET_ACCESS_CARD_INFO* pCardInfo = new NET_ACCESS_CARD_INFO[10];
            if (pCardInfo)
            {
                int nRecordNum = 0;
                NET_IN_CARDINFO_DO_FIND stuFindIn = {sizeof(stuFindIn)};
                stuFindIn.nStartNo = nStartNo;
                stuFindIn.nCount = 10;
                NET_OUT_CARDINFO_DO_FIND stuFindOut = {sizeof(stuFindOut)};
                stuFindOut.nMaxNum = 10;
                stuFindOut.pstuInfo = pstuAlarm;
                if (CLIENT_DoFindCardInfo(m_CardFindId, &stuFindIn, &stuFindOut, SDK_API_WAIT))
                {
                    if (stuFindOut.nRetNum > 0)
                    {
                        nRecordNum = stuFindOut.nRetNum;
                        m_bIsDoFindNext = TURE;
                    }
                }
                m_bIsDoFindNext = FALSE;
                for (int i=0;i<nRecordNum;i++)
                {
                    NET_ACCESS_CARD_INFO stuCardInfo;
                        memset(&stuCardInfo,0,sizeof(NET_ACCESS_CARD_INFO));
                        memcpy(&stuCardInfo,&pCardInfo[i],sizeof(NET_ACCESS_CARD_INFO));
```

```
                    m_CardInfoVector.push_back(stuCardInfo);
                }
                nStartNo = nRecordNum;
                delete []pCardInfo;
                pCardInfo = NULL;
            }
            else
            {
                m_bIsDoFindNext = FALSE;
            }
        }
        CLIENT_StopFindCardInfo (m_CardFindId);
        return TRUE;
    }
    else
    {
        return FALSE;
    }

//Add
NET_ACCESS_CARD_INFO stuCardInfo;
    memset(&stuCardInfo,0,sizeof(stuCardInfo));
    memcpy(&stuCardInfo, pstuCardInfo, sizeof(stuCardInfo));
    memcpy(stuCardInfo.szUserID,m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
    NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
    NET_IN_ACCESS_CARD_SERVICE_INSERT stuCardInsertIn = {sizeof(stuCardInsertIn)};
    stuCardInsertIn.nInfoNum = nNum;
    stuCardInsertIn.pCardInfo = &stuCardInfo;
    NET_OUT_ACCESS_CARD_SERVICE_INSERT stuCardINsertOut = {sizeof(stuCardINsertOut)};
    stuCardINsertOut.nMaxRetNum = nNum;
    stuCardINsertOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessCardService(m_lLoginID,
NET_EM_ACCESS_CTL_CARD_SERVICE_INSERT, &stuCardInsertIn, &stuCardINsertOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//Update
```

```
NET_ACCESS_CARD_INFO stuCardInfo;
	memset(&stuCardInfo,0,sizeof(stuCardInfo));
	memcpy(&stuCardInfo, pstuCardInfo, sizeof(stuCardInfo));
	memcpy(stuCardInfo.szUserID,m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
	NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
NET_IN_ACCESS_CARD_SERVICE_UPDATE stuCardUpdateIn = {sizeof(stuCardUpdateIn)};
	stuCardUpdateIn.nInfoNum = nNum;
	stuCardUpdateIn.pCardInfo = &stuCardInfo;
	NET_OUT_ACCESS_CARD_SERVICE_UPDATE stuCardUpdateOut = {sizeof(stuCardUpdateOut)};
	stuCardUpdateOut.nMaxRetNum = nNum;
	stuCardUpdateOut.pFailCode = &stuFailCode;
	BOOL bRet = CLIENT_OperateAccessCardService(m_lLoginID,
NET_EM_ACCESS_CTL_CARD_SERVICE_UPDATE, &stuCardUpdateIn, &stuCardUpdateOut, SDK_API_WAIT);
	if (bRet)
	{
		return TRUE;
	}
	return FALSE;

//Delete
NET_IN_ACCESS_CARD_SERVICE_REMOVE stuCardRemoveIn = {sizeof(stuCardRemoveIn)};
	stuCardRemoveIn.nCardNum = 1;
	NET_ACCESS_CARD_INFO stuCardInfo = m_CardInfoVector[m_nCardIndex];
	memcpy(&stuCardRemoveIn.szCardNo[0], stuCardInfo.szCardNo, sizeof(stuCardInfo.szCardNo));
	NET_OUT_ACCESS_CARD_SERVICE_REMOVE stuCardRemoveOut = {sizeof(stuCardRemoveOut)};
	stuCardRemoveOut.nMaxRetNum = 1;
	NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
	stuCardRemoveOut.pFailCode = &stuFailCode;
BOOL bRet = CLIENT_OperateAccessCardService(m_lLoginID, NET_EM_ACCESS_CTL_CARD_SERVICE_REMOVE,
&stuCardRemoveIn, &stuCardRemoveOut, SDK_API_WAIT);
	if (bRet)
	{
		return TRUE;
	}
	return FALSE;

//Clear
NET_IN_ACCESS_CARD_SERVICE_CLEAR stuCardClearIn = {sizeof(stuCardClearIn)};
	NET_OUT_ACCESS_CARD_SERVICE_CLEAR stuCardClearOut = {sizeof(stuCardClearOut)};
```

```
    BOOL bRet = CLIENT_OperateAccessCardService(m_lLoginID,
NET_EM_ACCESS_CTL_CARD_SERVICE_CLEAR, &stuCardClearIn, &stuCardClearOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
```

2.4.7.3 Face Management

2.4.7.3.1 Introduction

Call SDK to add, delete, query, and modify the face information fields of the access control device (including user ID and face picture).

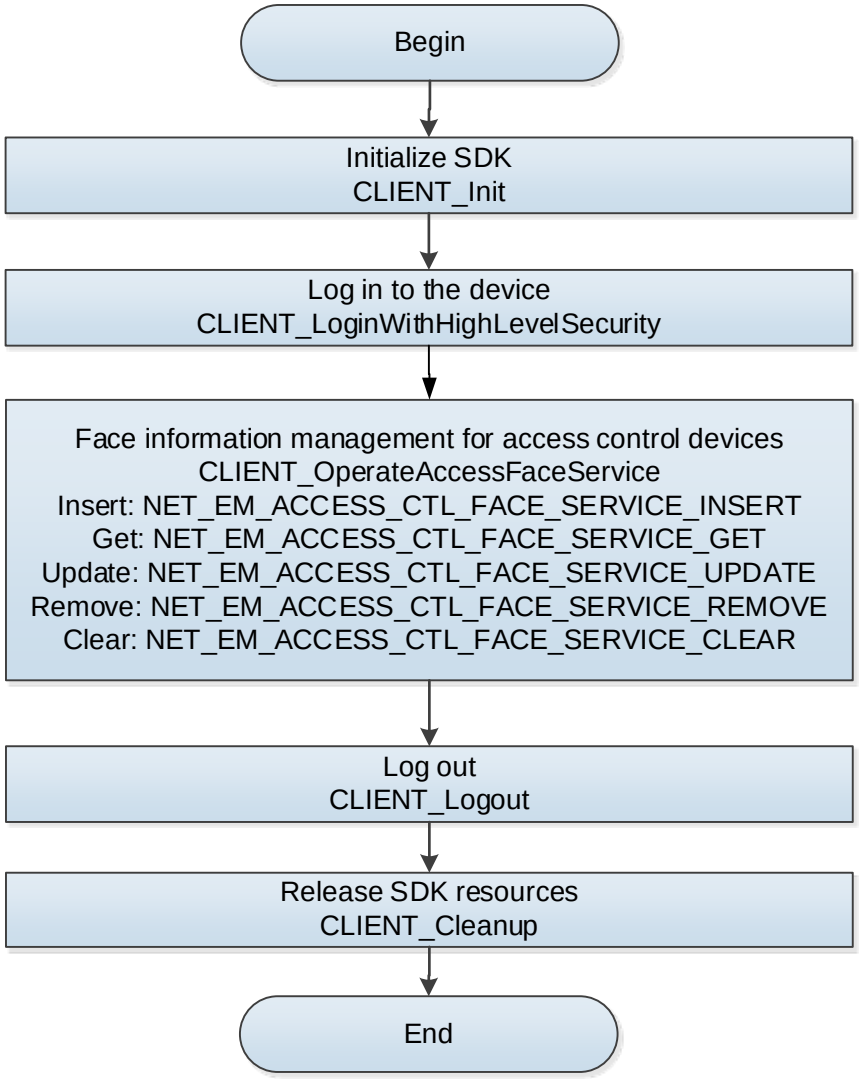
2.4.7.3.2 Interface Overview

Table 2-58 Description of face information interface

Interface	Description
CLIENT_OperateAccessFaceService	Face information management interface for access control devices

2.4.7.3.3 Process Description

Figure 2-44 Management of face information



Process

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

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

Step 3 Call **CLIENT_OperateAccessFaceService** to add, obtain, update, and delete the face information.

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.4.7.3.4 Example Code

```
//Add
NET_IN_ACCESS_FACE_SERVICE_INSERT    stuFaceInsertIn = {sizeof(stuFaceInsertIn)};
    stuFaceInsertIn.nFaceInfoNum = 1;
    NET_ACCESS_FACE_INFO stuFaceInfo;
    memset(&stuFaceInfo,0,sizeof(stuFaceInfo));
    memcpy(&stuFaceInfo.szUserID, &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
```

```
    stuFaceInfo.nFacePhoto = 1;

    FILE *fPic = fopen(m_szFilePath, "rb");
    if (fPic == NULL)
    {
        FaceUIState(TRUE);
        MessageBox(ConvertString("Open picture fail"), ConvertString("Prompt"));
        return;
    }

    fseek(fPic, 0, SEEK_END);
    int nLength = ftell(fPic);
    if (nLength <= 0)
    {
        goto FREE_RETURN;
    }
    rewind(fPic);

    stuFaceInfo.nInFacePhotoLen[0]= nLength;
    stuFaceInfo.nOutFacePhotoLen[0] = nLength;
    stuFaceInfo.pFacePhoto[0] = new char[nLength];
    memset(stuFaceInfo.pFacePhoto[0], 0, nLength);
    int nReadLen = fread(stuFaceInfo.pFacePhoto[0], 1, nLength, fPic);
    fclose(fPic);
    fPic = NULL;
    if (nReadLen <= 0)
    {
        goto FREE_RETURN;
    }
    stuFaceInsertIn.pFaceInfo = &stuFaceInfo;

    NET_OUT_ACCESS_FACE_SERVICE_INSERT    stuFaceInsertOut = {sizeof(stuFaceInsertOut)};
    stuFaceInsertOut.nMaxRetNum = 1;
    NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
    stuFaceInsertOut.pFailCode = &stuFailCodeR;
NET_IN_ACCESS_FACE_SERVICE_INSERT stuFaceInsertIn = {sizeof(stuFaceInsertIn)};
    stuFaceInsertIn.nFaceInfoNum = nNum;
    stuFaceInsertIn.pFaceInfo = &stuFaceInfo;
    NET_OUT_ACCESS_FACE_SERVICE_INSERT stuFaceINsertOut = {sizeof(stuFaceINsertOut)};
    stuFaceINsertOut.nMaxRetNum = nNum;
```

```
	stuFaceINsertOut.pFailCode = &stuFailCode;
	BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID,
NET_EM_ACCESS_CTL_FACE_SERVICE_INSERT, &stuFaceInsertIn, &stuFaceINsertOut, SDK_API_WAIT);
	if (bRet)
	{
		return TRUE;
	}
	return FALSE;

//Get
NET_IN_ACCESS_FACE_SERVICE_GET stuFaceGetIn = {sizeof(stuFaceGetIn)};
	stuFaceGetIn.nUserNum = 1;
	memcpy(&stuFaceGetIn.szUserID[0], &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));

	NET_OUT_ACCESS_FACE_SERVICE_GET stuFaceGetOut = {sizeof(stuFaceGetOut)};
	stuFaceGetOut.nMaxRetNum = 1;
	NET_ACCESS_FACE_INFO stuFaceInfo;
	memset(&stuFaceInfo,0,sizeof(stuFaceInfo));
	for (int i=0;i<5;i++)
	{
		stuFaceInfo.nInFacePhotoLen[i] = 100*1024;
		stuFaceInfo.pFacePhoto[i] = new char[100*1024];
		memset(stuFaceInfo.pFacePhoto[i],0,100*1024);
	}
	stuFaceGetOut.pFaceInfo = &stuFaceInfo;

	NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
	stuFaceGetOut.pFailCode = &stuFailCodeR;
BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID, NET_EM_ACCESS_CTL_FACE_SERVICE_GET,
&stuFaceGetIn, &stuFaceGetOut, SDK_API_WAIT);
	if (bRet)
	{
		return TRUE;
	}
	return FALSE;

//Update
NET_IN_ACCESS_FACE_SERVICE_INSERT	stuFaceInsertIn = {sizeof(stuFaceInsertIn)};
	stuFaceInsertIn.nFaceInfoNum = 1;
	NET_ACCESS_FACE_INFO stuFaceInfo;
```

```
    memset(&stuFaceInfo,0,sizeof(stuFaceInfo));
    memcpy(&stuFaceInfo.szUserID, &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
    stuFaceInfo.nFacePhoto = 1;

    FILE *fPic = fopen(m_szFilePath, "rb");
    if (fPic == NULL)
    {
        FaceUIState(TRUE);
        MessageBox(ConvertString("Open picture fail"), ConvertString("Prompt"));
        return;
    }

    fseek(fPic, 0, SEEK_END);
    int nLength = ftell(fPic);
    if (nLength <= 0)
    {
        goto FREE_RETURN;
    }
    rewind(fPic);

    stuFaceInfo.nInFacePhotoLen[0]= nLength;
    stuFaceInfo.nOutFacePhotoLen[0] = nLength;
    stuFaceInfo.pFacePhoto[0] = new char[nLength];
    memset(stuFaceInfo.pFacePhoto[0], 0, nLength);
    int nReadLen = fread(stuFaceInfo.pFacePhoto[0], 1, nLength, fPic);
    fclose(fPic);
    fPic = NULL;
    if (nReadLen <= 0)
    {
        goto FREE_RETURN;
    }
    stuFaceInsertIn.pFaceInfo = &stuFaceInfo;

    NET_OUT_ACCESS_FACE_SERVICE_INSERT    stuFaceInsertOut = {sizeof(stuFaceInsertOut)};
    stuFaceInsertOut.nMaxRetNum = 1;
    NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
    stuFaceInsertOut.pFailCode = &stuFailCodeR;
NET_IN_ACCESS_FACE_SERVICE_UPDATE stuFaceUpdateIn = {sizeof(stuFaceUpdateIn)};
    stuFaceUpdateIn.nFaceInfoNum = nNum;
    stuFaceUpdateIn.pFaceInfo = &stuFaceInfo;
```

```
	NET_OUT_ACCESS_FACE_SERVICE_UPDATE stuFaceUpdateOut = {sizeof(stuFaceUpdateOut)};
	stuFaceUpdateOut.nMaxRetNum = nNum;
	stuFaceUpdateOut.pFailCode = &stuFailCode;
	BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID,
NET_EM_ACCESS_CTL_FACE_SERVICE_UPDATE, &stuFaceUpdateIn, &stuFaceUpdateOut, SDK_API_WAIT);
	if (bRet)
	{
		return TRUE;
	}
	return FALSE;

//Delete
NET_IN_ACCESS_FACE_SERVICE_REMOVE stuFaceRIn = {sizeof(stuFaceRIn)};
	stuFaceRIn.nUserNum = 1;
	memcpy(&stuFaceRIn.szUserID[0], &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
	NET_OUT_ACCESS_FACE_SERVICE_REMOVE stuFaceROut = {sizeof(stuFaceROut)};
	stuFaceROut.nMaxRetNum = 1;
	NET_EM_FAILCODE stuFailCodeR = NET_EM_FAILCODE_NOERROR;
	stuFaceROut.pFailCode = &stuFailCodeR;
BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID, NET_EM_ACCESS_CTL_FACE_SERVICE_REMOVE,
&stuFaceRIn, &stuFaceROut, SDK_API_WAIT);
	if (bRet)
	{
		return TRUE;
	}
	return FALSE;

//Clear
NET_IN_ACCESS_FACE_SERVICE_CLEAR stuFaceClearIn = {sizeof(stuFaceClearIn)};
	NET_OUT_ACCESS_FACE_SERVICE_CLEAR stuFaceClearOut = {sizeof(stuFaceClearOut)};
	BOOL bRet = CLIENT_OperateAccessFaceService(m_lLoginID,
NET_EM_ACCESS_CTL_FACE_SERVICE_CLEAR, &stuFaceClearIn, &stuFaceClearOut, SDK_API_WAIT);
	if (bRet)
	{
		return TRUE;
	}
	return FALSE;
```

2.4.7.4 Fingerprint Management

2.4.7.4.1 Introduction

Call SDK to add, delete, query, and modify the fingerprint information fields of the access control device (including user ID, fingerprint data packet, and duress fingerprint number).

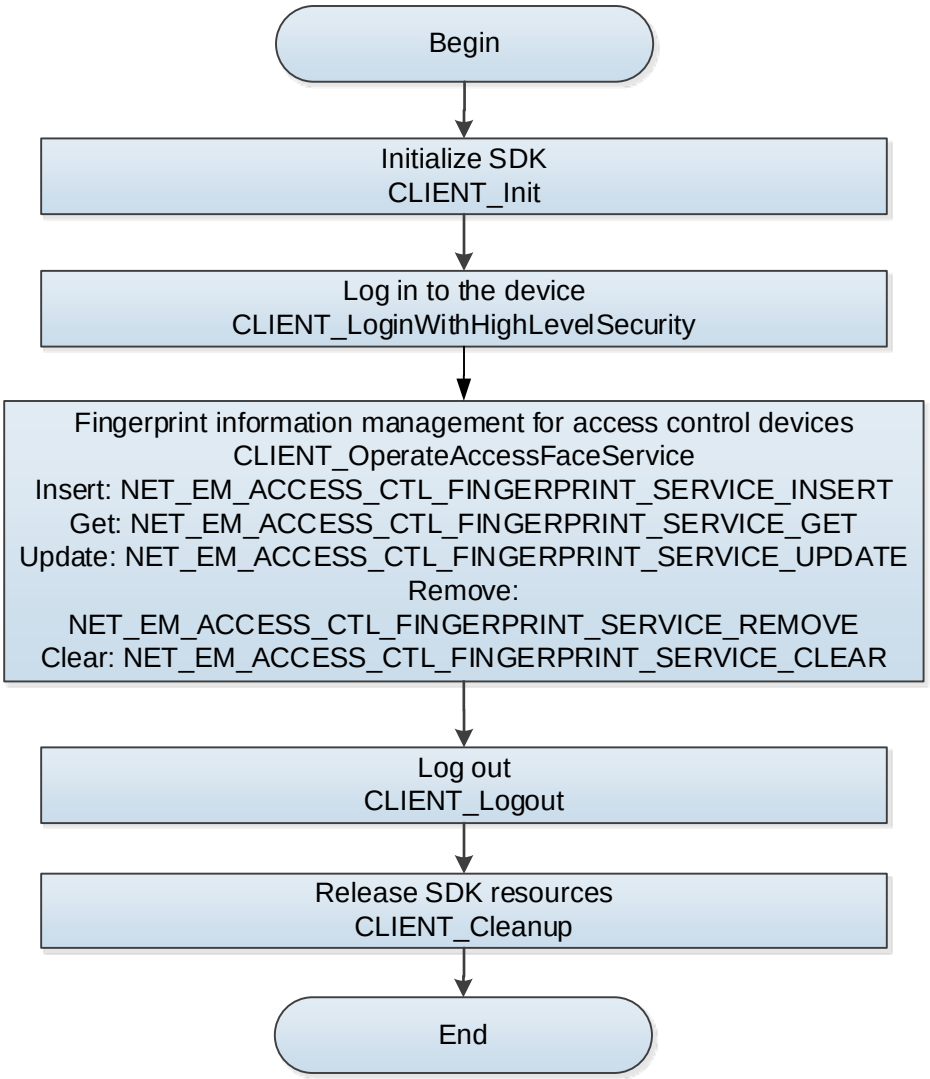
2.4.7.4.2 Interface Overview

Table 2-59 Description of fingerprint information interface

Interface	Description
CLIENT_OperateAccessFingerprintService	Fingerprint information management interface

2.4.7.4.3 Process Description

Figure 2-45 Management of fingerprint information



Process

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

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

Step 3 Call **CLIENT_OperateAccessFingerprintService** to add, obtain, update, delete, and clear the fingerprint information.

Step 4 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.4.7.4.4 Example Code

```
//Add
NET_ACCESS_FINGERPRINT_INFO stuFingerPrintInfo;
    memset(&stuFingerPrintInfo,0,sizeof(stuFingerPrintInfo));
    memcpy(&stuFingerPrintInfo.szUserID, &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
    stuFingerPrintInfo.nPacketLen = m_nFingerprintLen;
    stuFingerPrintInfo.nPacketNum = 1;
    stuFingerPrintInfo.szFingerPrintInfo = new char[m_nFingerprintLen];
    memset(stuFingerPrintInfo.szFingerPrintInfo, 0, m_nFingerprintLen);
    memcpy(stuFingerPrintInfo.szFingerPrintInfo, m_byFingerprintData, m_nFingerprintLen);
    if (bDuress)
    {
        stuFingerPrintInfo.nDuressIndex = 1;
    }
    //stuFingerPrintInfo.nDuressIndex
    NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
    NET_IN_ACCESS_FINGERPRINT_SERVICE_INSERT stuFingerPrintInsertIn = {sizeof(stuFingerPrintInsertIn)};
    stuFingerPrintInsertIn.nFpNum = nNum;
    stuFingerPrintInsertIn.pFingerPrintInfo = &stuFingerPrintInfo;
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_INSERT stuFingerPrintINsertOut =
{sizeof(stuFingerPrintINsertOut)};
    stuFingerPrintINsertOut.nMaxRetNum = nNum;
    stuFingerPrintINsertOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_INSERT, &stuFingerPrintInsertIn, &stuFingerPrintINsertOut,
SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
    if (stuFingerPrintInfo.szFingerPrintInfo != NULL)
    {
        delete[] stuFingerPrintInfo.szFingerPrintInfo;
        stuFingerPrintInfo.szFingerPrintInfo = NULL;
    }
```

```
//Get
NET_IN_ACCESS_FINGERPRINT_SERVICE_GET stuFingerPrintGetIn = {sizeof(stuFingerPrintGetIn)};
	memcpy(&stuFingerPrintGetIn.szUserID[0], &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
	BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_GET, &stuFingerprintGetIn, &stuFingerprintGetOut,
SDK_API_WAIT);
	if (bRet)
	{
		return TRUE;
	}
	return FALSE;

//Update
NET_ACCESS_FINGERPRINT_INFO stuFingerPrintInfo;
	memset(&stuFingerPrintInfo,0,sizeof(stuFingerPrintInfo));
	memcpy(&stuFingerPrintInfo.szUserID, &m_stuUserInfo.szUserID, sizeof(m_stuUserInfo.szUserID));
	if (m_nFingerprintLen != m_stuFingerprint.nSinglePacketLength)
	{
		MessageBox(ConvertString("FingerprintLen error"), ConvertString("Prompt"));
		FingerPrintUIState(TRUE);
		return;
	}
	stuFingerPrintInfo.nPacketLen = m_nFingerprintLen;
	stuFingerPrintInfo.nPacketNum = m_stuFingerprint.nRetFingerPrintCount;
	stuFingerPrintInfo.szFingerPrintInfo = new char[m_nFingerprintLen *
m_stuFingerprint.nRetFingerPrintCount];
	memset(stuFingerPrintInfo.szFingerPrintInfo, 0, m_nFingerprintLen *
m_stuFingerprint.nRetFingerPrintCount);
	memcpy(stuFingerPrintInfo.szFingerPrintInfo, m_stuFingerprint.pbyFingerData,
m_stuFingerprint.nRetFingerDataLength);
	memcpy(stuFingerPrintInfo.szFingerPrintInfo + m_nFingerprintIndex * m_nFingerprintLen,
m_byFingerprintData, m_nFingerprintLen);
	if (bDuress)
	{
		stuFingerPrintInfo.nDuressIndex = m_nFingerprintIndex + 1;
	}
	//stuFingerPrintInfo.nDuressIndex
	NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
	NET_IN_ACCESS_FINGERPRINT_SERVICE_UPDATE stuFingerprintUpdateIn =
{sizeof(stuFingerprintUpdateIn)};
```

```
    stuFingerprintUpdateIn.nFpNum = nNum;
    stuFingerprintUpdateIn.pFingerPrintInfo = &stuFingerPrintInfo;
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_UPDATE stuFingerprintUpdateOut =
{sizeof(stuFingerprintUpdateOut)};
    stuFingerprintUpdateOut.nMaxRetNum = nNum;
    stuFingerprintUpdateOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_UPDATE, &stuFingerprintUpdateIn, &stuFingerprintUpdateOut,
SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
    if (stuFingerPrintInfo.szFingerPrintInfo != NULL)
    {
        delete[] stuFingerPrintInfo.szFingerPrintInfo;
        stuFingerPrintInfo.szFingerPrintInfo = NULL;
    }

//Delete
NET_IN_ACCESS_FINGERPRINT_SERVICE_REMOVE stuFingerPrintRemoveIn = {sizeof(stuFingerPrintRemoveIn)};
    stuFingerPrintRemoveIn.nUserNum = 1;
    memcpy(&stuFingerPrintRemoveIn.szUserID[0], &m_stuUserInfo.szUserID,
sizeof(m_stuUserInfo.szUserID));
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_REMOVE stuFingerPrintRemoveOut =
{sizeof(stuFingerPrintRemoveOut)};
    stuFingerPrintRemoveOut.nMaxRetNum = 1;
    NET_EM_FAILCODE stuFailCode = NET_EM_FAILCODE_NOERROR;
    stuFingerPrintRemoveOut.pFailCode = &stuFailCode;
    BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_REMOVE, &stuFingerprintRemoveIn,
&stuFingerprintRemoveOut, SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;

//Clear
NET_IN_ACCESS_FINGERPRINT_SERVICE_CLEAR stuFingerprintClearIn = {sizeof(stuFingerprintClearIn)};
```

```
    NET_OUT_ACCESS_FINGERPRINT_SERVICE_CLEAR stuFingerprintClearOut =
{sizeof(stuFingerprintClearOut)};
    BOOL bRet = CLIENT_OperateAccessFingerprintService(m_lLoginID,
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_CLEAR, &stuFingerprintClearIn, &stuFingerprintClearOut,
SDK_API_WAIT);
    if (bRet)
    {
        return TRUE;
    }
    return FALSE;
```

2.4.8 Door Config

See "2.3.8 Door Config."

2.4.9 Door Time Config

2.4.9.1 Period Config

See "2.3.9.1 Period Config."

2.4.9.2 Always Open and Always Closed Period Config

See "2.3.9.2 Always Open and Always Closed Period Config."

2.4.9.3 Holiday Group

2.4.9.3.1 Introduction

Configure the holiday group of the device through SDK, including the holiday group name, the start and end time, and group enabling.

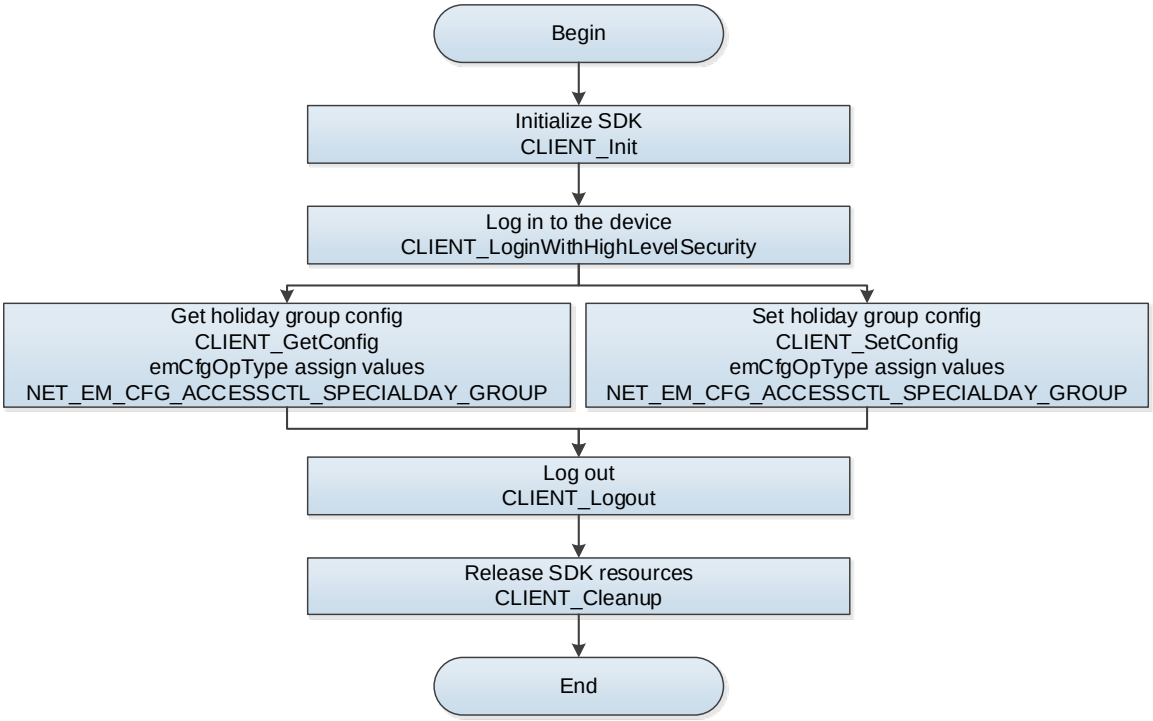
2.4.9.3.2 Interface Overview

Table 2-60 Description of holiday group interface

Interface	Description
CLIENT_GetConfig	Query config information.
CLIENT_SetConfig	Set config information.

2.4.9.3.3 Process Description

Figure 2-46 Holiday group



Process

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

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

Step 3 Call **CLIENT_GetConfig** to query the holiday group config info for the access control device.

Table 2-61 Description of emCfgOpType

emCfgOpType	Description	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP	Get holiday info	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO	Structure size of NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO

Step 4 Call **CLIENT_SetConfig** to set the holiday group config info for the access control device.

Table 2-62 Description of emCfgOpType

emCfgOpType	Description	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP	Set holiday info	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO	Structure size of NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.4.9.3.4 Example Code

```
//Get
```

```
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO stuIn = {sizeof(stuIn)};
    BOOL bret = CLIENT_GetConfig((LLONG)m_lLoginID, NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP,
nId,&stuSpecialdayGroup, sizeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO));
    if (!bret)
    {
        return FALSE;
    }
    else
    {
        return TRUE;
    }

//Set
NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO stuSpecialdayGroup;
stuSpecialdayGroup.bGroupEnable = m_chkGroupEnable.GetCheck() ? TRUE : FALSE;
BOOL bret = CLIENT_SetConfig((LLONG)m_lLoginID, NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP, nId,
&stuSpecialdayGroup, sizeof(NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO));// The channel is the holiday
group number 0-127.
    if (!bret)
    {
        return FALSE;
    }
    else
    {
        return TRUE;
}
```

2.4.9.4 Holiday Plan

2.4.9.4.1 Introduction

Configure the holiday plan of the device through SDK, including the holiday plan name, enabling, period, and valid door channel.

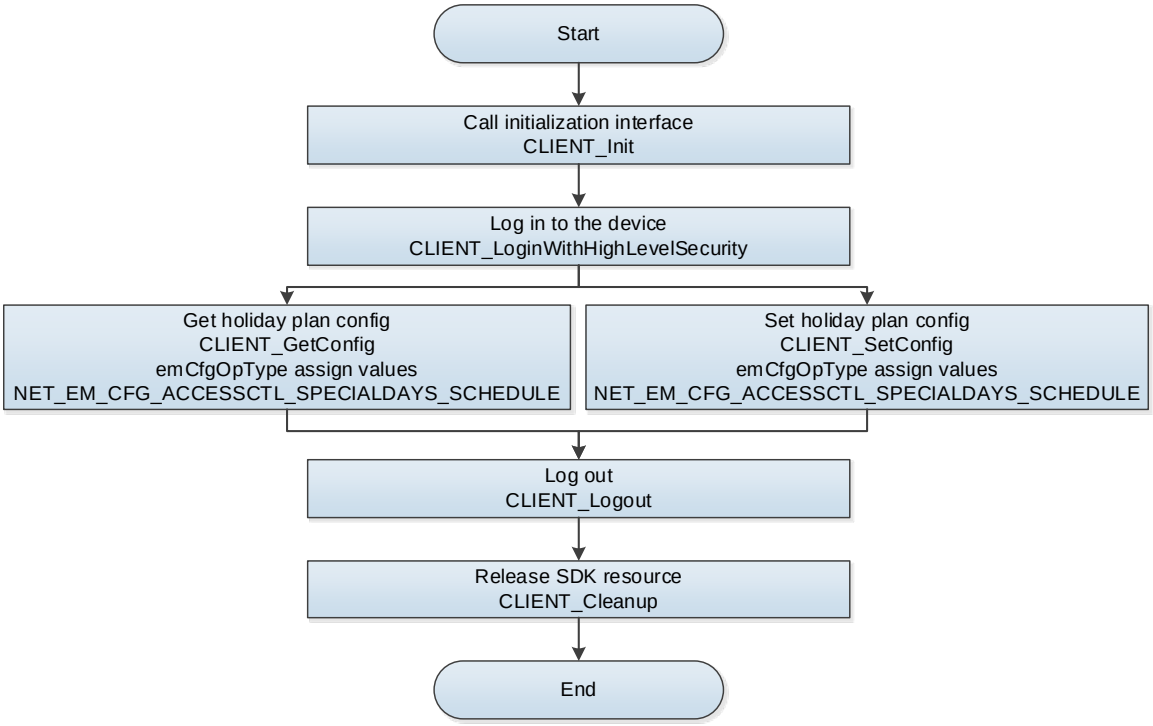
2.4.9.4.2 Interface Overview

Table 2-63 Description of holiday plan interface

Interface	Description
CLIENT_GetConfig	Query config information.
CLIENT_SetConfig	Set config information.

2.4.9.4.3 Process Description

Figure 2-47 Holiday plan



Process

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

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

Step 3 Call **CLIENT_GetConfig** to query the holiday plan config info for the access control device.

Table 2-64 Description of emCfgOpType

emCfgOpType	Description	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE	Get holiday info	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO	Structure size of NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO

Step 4 Call **CLIENT_SetConfig** to set the the holiday plan config info for the access control device.

Table 2-65 Description of emCfgOpType

emCfgOpType	Description	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE	Set holiday info	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO	Structure size of NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO

Step 5 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.4.9.4.4 Example Code

```
//Get
```

```
NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO stuSpecialdaySchedule =
{sizeof(stuSpecialdaySchedule)};
	BOOL bret = CLIENT_GetConfig((LLONG)m_lLoginID,
NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE, nId,& stuSpecialdaySchedule,
sizeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO));
	if (!bret)
	{
		return FALSE;
	}
	else
	{
		return TRUE;
	}

//Set
NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO stuSpecialdaySchedule;
stuSpecialdaySchedule.bSchdule = m_chkSpeciadayEnable.GetCheck() ? TRUE : FALSE;
stuSpecialdaySchedule.nGroupNo = 1;
BOOL bret = CLIENT_SetConfig((LLONG)m_lLoginID, NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE, nId,
& stuSpecialdaySchedule, sizeof(NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO));
	if (!bret)
	{
		return FALSE;
	}
	else
	{
		return TRUE;
	 }
```

2.4.10 Advanced Config of Door

See "2.3.10 Advanced Config of Door."

2.4.11 Records Query

2.4.11.1 Unlock Records

See "2.3.11.1 Unlock Records."

2.4.11.2 Device Log

See "2.3.11.2 Device log."

2.4.11.3 Alarm Records

2.4.11.3.1 Introduction

Query the alarm records of the access control device through the SDK interface.

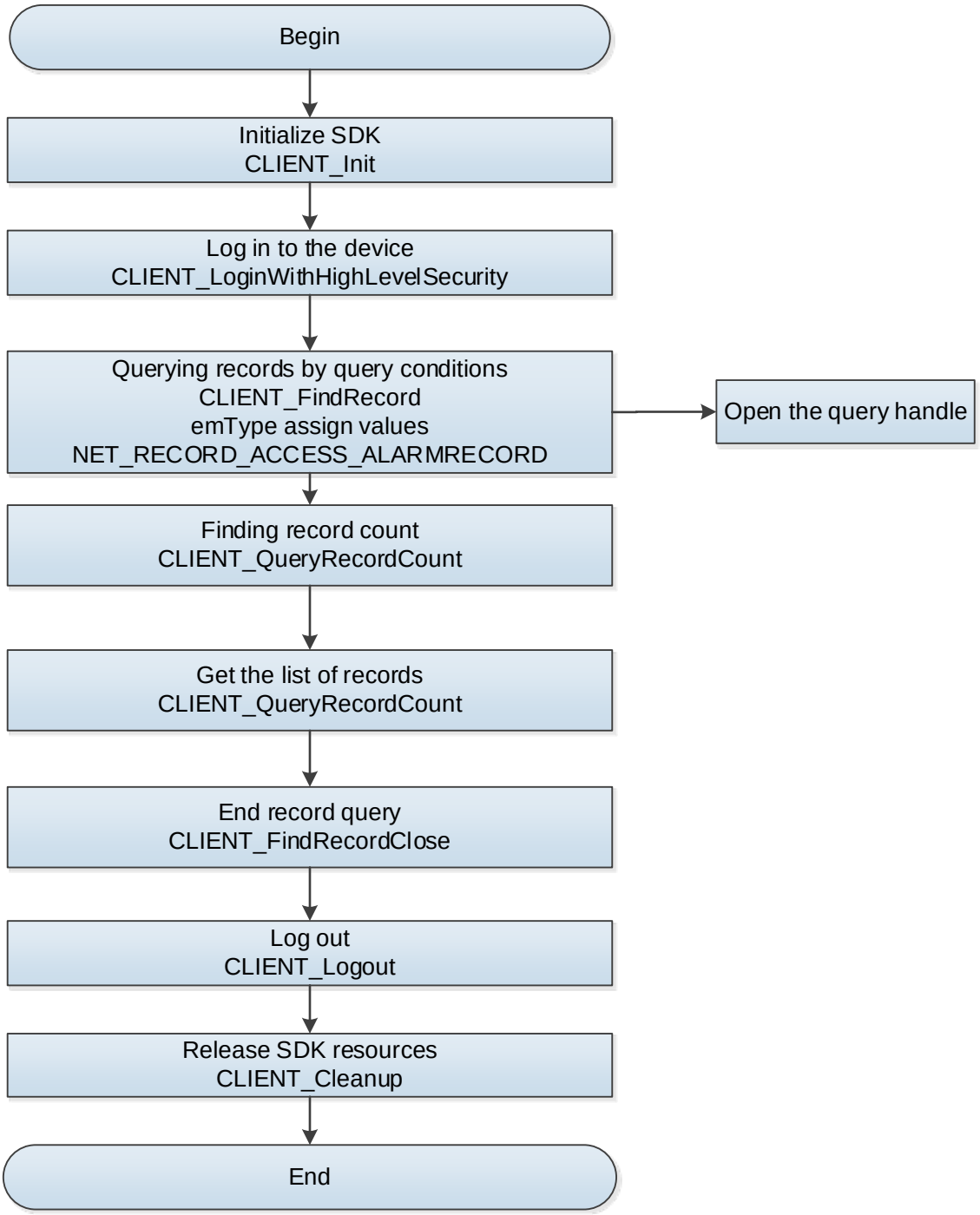
2.4.11.3.2 Interface Overview

Table 2-66 Description of record query interfaces

Interface	Description
CLIENT_QueryRecordCount	Find the count of records
CLIENT_FindRecord	Query records by query conditions
CLIENT_FindNextRecord	Find records: nFilecount: count of files to be queried. When the return value is the count of media files and less than nFilecount, the query of files is completed within the corresponding period
CLIENT_FindRecordClose	End record query

2.4.11.3.3 Process Description

Figure 2-48 Record query



Process

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

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

Step 3 Call the **CLIENT_FindRecord** to get the query handle.
Assign NET_RECORD_ACCESS_ALARMRECORD to emType in pInParam.

Step 4 Call the **CLIENT_QueryRecordCount** to find the count of records.

Step 5 Call the **CLIENT_FindNextRecord** to get the list of records.

Step 6 Call **CLIENT_FindRecordClose** to close the query handle.

Step 7 After completing this process, call the **CLIENT_Logout** to log out of the device.

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

2.4.11.3.4 Example Code

```
//Start finding
NET_IN_FIND_RECORD_PARAM stuIn = {sizeof(stuIn)};
    NET_OUT_FIND_RECORD_PARAM stuOut = {sizeof(stuOut)};
    stuIn.emType = NET_RECORD_ACCESS_ALARMRECORD;
    if (CLIENT_FindRecord(m_lLoginID, &stuIn, &stuOut, SDK_API_WAIT))
    {
        m_AlarmFindId = stuOut.lFindeHandle;
        return TRUE;
    }
    else
    {
        return FALSE;
}

//Find the number of records
NET_IN_QUEYT_RECORD_COUNT_PARAM stuIn = {sizeof(stuIn)};
    stuIn.lFindeHandle = m_AlarmFindId;
    NET_OUT_QUEYT_RECORD_COUNT_PARAM stuOut = {sizeof(stuOut)};
    if (CLIENT_QueryRecordCount(&stuIn, &stuOut, SDK_API_WAIT))
    {
        nCount = stuOut.nRecordCount;
        return TRUE;
    }
    else
    {
        return FALSE;
    }

//Find the records
    NET_RECORD_ACCESS_ALARMRECORD_INFO pstuAlarm;
NET_IN_FIND_NEXT_RECORD_PARAM stuIn = {sizeof(stuIn)};
    stuIn.lFindeHandle = m_AlarmFindId;
    stuIn.nFileCount = nMaxNum;

    NET_OUT_FIND_NEXT_RECORD_PARAM stuOut = {sizeof(stuOut)};
    stuOut.nMaxRecordNum = nMaxNum;
    stuOut.pRecordList = (void*)pstuAlarm;
```

```
        if (CLIENT_FindNextRecord(&stuIn, &stuOut, SDK_API_WAIT) >= 0)
        {
                if (stuOut.nRetRecordNum > 0)
                {
                        nRecordNum = stuOut.nRetRecordNum;
                        return TRUE;
                }
        }
        return FALSE;
//End record query
BOOL bret = CLIENT_FindRecordClose(m_AlarmFindId);
        if (bret)
        {
                m_AlarmFindId = NULL;
                return TRUE;
        }
        else
        {
                return FALSE;
        }
```

3 Interface Function

3.1 Common Interface

3.1.1 SDK Initialization

3.1.1.1 SDK Initialization CLIENT_Init

Table 3-1 SDK initialization description

Item	Description	
Description	Initialize the SDK.	
Function	BOOL CLIENT_Init(● fDisConnect cbDisConnect, ● LDWORD dwUser);	
Parameter	[in]cbDisConnect	Disconnection callback.
	[in]dwUser	User parameters for disconnection callback.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	● Prerequisite for calling other functions of the NetSDK. ● When the callback is set as NULL, the device will not be sent to the user after disconnection.	

3.1.1.2 SDK Cleaning up CLIENT_Cleanup

Table 3-2 Description of SDK cleaning up

Item	Description
Description	Clean up SDK.
Function	void CLIENT_Cleanup()
Parameter	None.
Return Value	None.
Note	SDK cleaning up interface is finally called before the end.

3.1.1.3 Setting Reconnection Callback CLIENT_SetAutoReconnect

Table 3-3 Description of setting 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 parameters for reconnection callback.
Return Value	None.	
Note	Set reconnection callback interface. If the callback is set as NULL, the device will not be reconnected automatically.	

3.1.1.4 Setting Network Parameter CLIENT_SetNetworkParam

Table 3-4 Description of device network parameter

Item	Description	
Description	Set network parameters.	
Function	void CLIENT_SetNetworkParam(● NET_PARAM *pNetParam);	
Parameter	[in]pNetParam	Network delay, number of reconnections, buffer size and other parameters.
Return Value	None.	
Note	You can adjust parameters according to the actual network environment.	

3.1.2 Device Initialization

3.1.2.1 Searching Device CLIENT_StartSearchDevicesEx

Table 3-5 Description of searching device

Item	Description	
Description	Search device information.	
Function	● LLONG CLIENT_StartSearchDevicesEx (● NET_IN_STARTSERACH_DEVICE* pInBuf, ● NET_OUT_STARTSERACH_DEVICE* pOutBuf ●);	
Parameter	[in] pInBuf	Input parameter of async searching. Refer to NET_IN_STARTSERACH_DEVICE
	[out] pOutBuf	Output parameter of async searching. Refer to NET_OUT_STARTSERACH_DEVICE
Return Value	Search handle.	
Note	Multi-thread calling is not supported.	

3.1.2.2 Device Initialization CLIENT_InitDevAccount

Table 3-6 Description of device initialization

Item	Description	
Description	Initialize 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	Input parameter, corresponding to NET_IN_INIT_DEVICE_ACCOUNT structure.
	[out]pInitAccountOut	Output parameter, corresponding to NET_OUT_INIT_DEVICE_ACCOUNT structure.
	[in]dwWaitTime	Timeout period.
	[in]szLocalIp	● In the case of single network adapter, szLocalIp can be left empty. ● In the case of multiple network adapters, fill the host IP in szLocalIp.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	None.	

3.1.2.3 Getting Password Reset Information CLIENT_GetDescriptionForResetPwd

Table 3-7 Description of getting password reset information

Item	Description	
Description	Get password reset information.	
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	Input parameter, corresponding to NET_IN_DESCRIPTION_FOR_RESET_PWD structure.
	[out]pDescriptionOut	Output parameter, corresponding to NET_OUT_DESCRIPTION_FOR_RESET_PWD structure.
	[in]dwWaitTime	Timeout period.

Item	Description	
	[in]szLocalIp	● In the case of single network adapter, the last parameter can be left empty. ● In the case of multiple network adapters, please fill the host IP in the last parameter.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	None	

3.1.2.4 Checking the Validity of Security Code CLIENT_CheckAuthCode

Table 3-8 Description of checking 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	Input parameter, corresponding to NET_IN_CHECK_AUTHCODE structure.
	[out]pCheckAuthCodeOut	Output parameter, corresponding to NET_OUT_CHECK_AUTHCODE structure.
	[in]dwWaitTime	Timeout period.
	[in]szLocalIp	● In the case of single network adapter, the last parameter can be left empty. ● In the case of multiple network adapters, please fill the host IP in the last parameter.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	● None.	

3.1.2.5 Resetting Password CLIENT_ResetPwd

Table 3-9 Description of resetting 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);

Item	Description	
Parameter	[in]pResetPwdIn	Input parameter, corresponding to NET_IN_RESET_PWD structure.
	[out]pResetPwdOut	Output parameter, corresponding to NET_OUT_RESET_PWD structure.
	[in]dwWaitTime	Timeout period.
	[in]szLocalIp	● In the case of single network adapter, the last parameter can be left empty. ● In the case of multiple network adapters, please fill the host IP in the last parameter.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	● None.	

3.1.2.6 Getting Password Rules CLIENT_GetPwdSpecification

Table 3-10 Description of getting 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	Input parameter, corresponding to NET_IN_PWD_SPECI structure.
	[out]pPwdSpeciOut	Output parameter, corresponding to NET_OUT_PWD_SPECI structure.
	[in]dwWaitTime	Timeout period.
	[in]szLocalIp	● In the case of single network adapter, the last parameter can be left empty. ● In the case of multiple network adapters, please fill the host IP in the last parameter.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	● None.	

3.1.2.7 Stopping Searching Device CLIENT_StopSearchDevices

Table 3-11 Description of stopping searching device

Item	Description
Description	Stop searching device information.

Item	Description	
Function	BOOL CLIENT_StopSearchDevices (● LLONG lSearchHandle);	
Parameter	[in] lSearchHandle	Input parameter, search handle.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	Multi-thread calling is not supported.	

3.1.3 Device Login

3.1.3.1 Logging in to the Device CLIENT_LoginWithHighLevelSecurity

Table 3-12 Description of user logging in to the device

Item	Description	
Description	Log in to the device.	
Function	● LLONG CLIENT_LoginWithHighLevelSecurity (● NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY* pstInParam, ● NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY* pstOutParam ●);	
Parameter	[in] pstInParam	Login parameters include IP, port, user name, password, login mode.
	[out] pstOutParam	Device login output parameters include device information, error code.
Return Value	● Success: Non-0 ● Failure: 0	
Note	High security level login interface. You can still use CLINET_LoginEx2, but there is a security risk. Therefore, it is highly recommended to use the latest interface CLIENT_LoginWithHighLevelSecurity to log in to the device.	

Table 3-13 Error codes and meanings of errors in the parameter

Error code	Corresponding meanings
1	Incorrect password.
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 restricted.
7	Out of resources, the system is busy.
8	Sub-connection failed.
9	Primary connection failed.
10	Exceeded the maximum number of user connections.
11	Lack of avnetsdk or avnetsdk dependent library.

Error code	Corresponding meanings
12	USB flash drive is not inserted into device, or the USB flash disk information error.
13	The client IP address is not authorized with login.

3.1.3.2 User Logging Out of the Device CLIENT_Logout

Table 3-14 Description of user logging out of the device

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

3.1.4 Realtime Monitor

3.1.4.1 Opening the Monitoring CLIENT_RealPlayEx

Table 3-15 Description of opening the 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, an integer increasing from 0.
	[in]hWnd	Window handle, only valid in Windows system.
	[in]rType	Live view type.
Return Value	● Success: Non-0 ● Failure: 0	
Note	In Windows environment: ● When hWnd is valid, the picture is displayed in the corresponding window. ● When hWnd is NULL, the way of getting stream is to get video data by setting callback function, and then submit the data to users for processing.	

Table 3-16 Description of live view types

Live view type	Meanings
DH_RType_Realplay	Live View
DH_RType_Multiplay	Zero-Ch Encode
DH_RType_Realplay_0	Real-time monitoring—main stream, equivalent to DH_RType_Realplay
DH_RType_Realplay_1	Real-time monitoring—sub stream 1
DH_RType_Realplay_2	Real-time monitoring—sub stream 2
DH_RType_Realplay_3	Real-time monitoring—sub stream 3
DH_RType_Multiplay_1	Multi-picture preview—1 picture
DH_RType_Multiplay_4	Multi-picture preview—4 pictures
DH_RType_Multiplay_8	Multi-picture preview—8 pictures
DH_RType_Multiplay_9	Multi-picture preview—9 pictures
DH_RType_Multiplay_16	Multi-picture preview—16 pictures
DH_RType_Multiplay_6	Multi-picture preview—6 pictures
DH_RType_Multiplay_12	Multi-picture preview—12 pictures
DH_RType_Multiplay_25	Multi-picture preview—25 pictures
DH_RType_Multiplay_36	Multi-picture preview—36 pictures

3.1.4.2 Closing the Monitoring CLIENT_StopRealPlayEx

Table 3-17 Description of closing the monitoring

Item	Description	
Description	Close 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 Saving the Monitoring Data CLIENT_SaveRealData

Table 3-18 Description of saving the monitoring data

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

3.1.4.4 Stopping Saving the Monitoring Data CLIENT_StopSaveRealData

Table 3-19 Description of stopping saving the monitoring data

Item	Description	
Description	Stop saving the real-time monitoring data as a 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 Setting Monitoring Data Callback CLIENT_SetRealDataCallBackEx2

Table 3-20 Description of setting monitoring data callback

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

Table 3-21 dwFlag types and meanings

dwFlag	Meanings
REALDATA_FLAG_RAW_DATA	Flag of raw data
REALDATA_FLAG_DATA_WITH_FRAME_INFO	Flag of data with frame information
REALDATA_FLAG_YUV_DATA	Flag of YUV data
REALDATA_FLAG_PCM_AUDIO_DATA	Flag of PCM audio data

3.1.5 Device Control

3.1.5.1 Device Controlling CLIENT_ControlDeviceEx

Table 3-22 Device control description

Item	Description	
Description	Device control.	
Function	BOOL CLIENT_ControlDeviceEx(● LLONG lLoginID, ● CtrlType emType, ● Void *pInBuf, ● Void *pOutBuf = NULL, ● int nWaitTime = 1000);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in]emType	Control type.
	[in]pInBuf	Input parameters, which vary by emType.
	[out]pOutBuf	Output parameters, NULL by default; for some emTypes, there are corresponding output structures.
	[in]waittime	Timeout period, 1000 ms by default, which can be set as needed.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	None.	

Table 3-23 Comparison of emType, pInBuf and pOutBuf

emType	Description	pInBuf	pOutBuf
DH_CTRL_ARMED_EX	Arming and Disarming	CTRL_ARM_DISARM_PARAM	NULL
DH_CTRL_SET_BYPASS	Bypass setting function	NET_CTRL_SET_BYPASS	NULL
DH_CTRL_ACCESS_OPEN	Access control—open	NET_CTRL_ACCESS_OPEN	NULL
DH_CTRL_ACCESS_CLOSE	Access control—close	NET_CTRL_ACCESS_CLOSE	NULL

3.1.6 Alarm Listening

3.1.6.1 Setting Alarm Callback Function DPSDK_SetEventCallBack

Table 3-24 Description of setting alarm callback function

Item	Description
Description	Set alarm callback function.

Item	Description	
Function	void CLIENT_SetDVRMessCallBack(● fMessCallBack cbMessage, ● LDWORD dwUser);	
Parameter	[in]cbMessage	Message callback function ● Status in which devices can be called back, such as alarm status. ● When the value is set as 0, it means callback is forbidden.
	[in]dwUser	User-defined data.
Return Value	None	
Note	● Set device message callback function to get the current device status information; this function is independent of the calling sequence, and the SDK is not called back by default. ● The callback function fMessCallBack must call the alarm message subscription interface CLIENT_StartListenEx first before it takes effect.	

3.1.6.2 Subscribing to Alarm CLIENT_StartListenEx

Table 3-25 Description of subscribing to alarm

Item	Description	
Description	Subscribing to alarms.	
Function	BOOL CLIENT_StartListenEx(● LLONG lLoginID);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	Subscribe to device message, and the message received is called back from the set value of CLIENT_SetDVRMessCallBack.	

3.1.6.3 Stopping Subscribing to Alarm CLIENT_StopListen

Table 3-26 Description of stopping subscribing to alarm

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

3.1.7 Subscribing to Intelligent Event

3.1.7.1 CLIENT_RealLoadPictureEx

Table 3-27 Subscribe intelligent event interface

Item	Description	
Name	Subscribe intelligent event interface.	
Function	LLONG CLIENT_RealLoadPictureEx(LLONG lLoginID, int nChannelID, DWORD dwAlarmType, BOOL bNeedPicFile, fAnalyzerDataCallBack cbAnalyzerData, LDWORD dwUser, void* Reserved);	
Parameter	[in] lLoginID	Login handle.
	[in] nChannelID	Channel number.
	[in] dwAlarmType	Alarm type.
	[in] bNeedPicFile	Whether to subscribe picture file, 1-yes, return intelligent picture information in the callback function; 0-no, do not return intelligent picture information(in this case, it can reduce the network flow).
	[in] cbAnalyzerData	Intelligent data analysis callback function.
	[in] dwUser	The user parameters.
	[in] Reserved	Reserve parameter.
Return value	Success: the subscribe handle of LLONG type. Failure: 0.	
Note	If the interface return failed, call CLIENT_GetLastError to get error code.	

Table 3-28 Preview type and meaning

Preview Type	Meaning
EVENT_IVS_CITIZEN_PICTURE_COMPARE	Event of comparison with ID and card picture.
EVENT_IVS_ACCESS_CTL	Access control event.

3.1.7.2 CLIENT_StopLoadPic

Table 3-29 Stop subscribing intelligent event.

Item	Description	
Name	Stop subscribing intelligent event.	
Function	BOOL CLIENT_StopLoadPic(LLONG lAnalyzerHandle);	
Parameter	[in] lAnalyzerHandle	Event subscribing handle.
Return value	Success: TRUE. Failure: FALSE.	

Item	Description
Note	None.

3.1.8 Getting Device Status

3.1.8.1 Getting Device Status CLIENT_Querydevstate

Table 3-30 Description of getting device status

Item	Description	
Description	Directly get the connection status of remote devices.	
Function	BOOL CLIENT_QueryDevState(● LLONG lLoginID, ● int nType, ● char *pBuf, ● int nBufLen, ● int *pRetLen, ● int waittime=1000);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in]nType	Query information type.
	[out]pBuf	To receive the returned data buffer in query. Based on different query types, the structures of returned data are also different. See Table 3-28 for details.
	[in]nBufLen	Buffer length, in bytes.
	[out]pRetLen	Length of data actually returned, in bytes.
	[in]waittime	Waiting time in query status, 1000 ms by default, which can be set as needed.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	None.	

Table 3-31 Correspondence between query information type and structure

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

3.1.9 Voice Talk

3.1.9.1 Getting Talk Type Supported by the Device CLIENT_GetDevProtocolType

Table 3-32 Description of getting talk type supported by the device

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

3.1.9.2 Setting Voice Talk Mode CLIENT_Setdevicemode

Table 3-33 Description of setting device voice talk mode

Item	Description	
Description	Set device voice talk mode.	
Function	BOOL CLIENT_SetDeviceMode(● LLONG lLoginID, ● EM_USEDEV_MODE emType, ● void *pValue);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[in]emType	Enumerated value.
	[in]pValue	For structure data pointers corresponding to the enumerated values, see Table 3-31.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	None.	

Table 3-34 Comparison of emType and pValue

emType	Description	pValue
DH_TALK_ENCODE_TYPE	Talk in a specified format.	DHDEV_TALKDECODE_INFO
DH_TALK_CLIENT_MODE	Set voice talk client mode.	None
DH_TALK_SPEAK_PARAM	Set speak parameters for voice talk.	NET_SPEAK_PARAM

emType	Description	pValue
DH_TALK_MODE3	Set voice talk parameters for third-generation devices.	NET_TALK_EX

3.1.9.3 Starting Talk CLIENT_Starttalkex

Table 3-35 Description of starting 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 function.
	[in]dwUser	Parameters of audio data callback function.
Return Value	● Success: Non-0 ● Failure: 0	
Note	None.	

3.1.9.4 Stopping Talk CLIENT_StopTalkEx

Table 3-36 Description of stopping 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.	

3.1.9.5 Opening the Recording CLIENT_RecordStartEx

Table 3-37 Description of opening the recording

Item	Description	
Description	Open the local recording.	
Function	BOOL CLIENT_RecordStartEx(● LLONG lLoginID);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
Return Value	● Success: TRUE ● Failure: FALSE	

Item	Description
Note	This interface is only valid in Windows.

3.1.9.6 Stoping the Recording CLIENT_RecordStopEx

Table 3-38 Description of closing the recording

Item	Description	
Description	Stop the local recording.	
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.	

3.1.9.7 Sending Voice CLIENT_TalkSendData

Table 3-39 Description of sending voice

Item	Description	
Description	Send audio data to the device.	
Function	LONG CLIENT_TalkSendData(● LLONG lTalkHandle, ● char *pSendBuf, ● DWORD dwBufSize);	
Parameter	[in]lTalkHandle	Return value of CLIENT_StartTalkEx.
	[in]pSendBuf	Pointer of audio data blocks to be sent.
	[in]dwBufSize	Length of audio data blocks to be sent, in bytes.
Return Value	● Length of audio data blocks successfully returned. ● Return -1 if failed.	
Note	None.	

3.1.9.8 Decoding Voice CLIENT_AudioDecEx

Table 3-40 Description of decoding voice

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	Pointer of audio data blocks to be decoded.

Item	Description	
	[in]dwBufSize	Length of audio data blocks to be decoded, in bytes.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	None.	

3.2 Alarm Host

3.3 Access Controller/ All-in-one Fingerprint Machine (First-generation)

3.3.1 Access Control

For details of the door control interface, see "3.1.5.1 Device Controlling CLIENT_ControlDeviceEx."

For details of the door sensor status interface, see "3.3.3.4 Querying Device Status CLIENT_QueryDevState."

3.3.2 Alarm Event

See "3.1.6 Alarm Listening."

3.3.3 Viewing Device Information

3.3.3.1 Querying System Capability Information CLIENT_QueryNewSystemInfo

Table 3-41 Description of querying system capability information

Item	Description	
Description	Query system capability information in string format.	
Function	BOOL CLIENT_QueryNewSystemInfo (● LLONG lLoginID, ● char *szCommand, ● int nChannelID, ● char *szOutBuffer, ● DWORD dwOutBufferSize, ● int *error, ● int nWaitTime = 1000);	
Parameter	[in]lLoginID	Return value of CLIENT_Login or CLIENT_LoginEx.

Item	Description	
	[in] szCommand	Command parameter. See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData" for details.
	[in] nChannelID	Channel.
	[out] szOutBuffer	Received protocol buffer.
	[in] dwOutBufferSize	Total number of bytes received (in bytes).
	[out] error	Error number.
	[in]waittime	Timeout period, 1000ms by default, which can be set as needed.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	The information got is in string format, and information contained in each string is parsed by CLIENT_ParseData.	

Table 3-42 Error codes and meanings of errors in the parameter

Error code	Corresponding meanings
0	Successful
1	Failed
2	Illegal data
3	Cannot be set for now
4	Permission denied

3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData

Table 3-43 Description of parsing the queried config information

Item	Description	
Description	Parse the queried config information.	
Function	BOOL CLIENT_ParseData (● char *szCommand, ● char *szInBuffer, ● LPVOID lpOutBuffer, ● DWORD dwOutBufferSize, ● int *pReserved);	
Parameter	[in] szCommand	Command parameter. See Table 3-41 for details.
	[in] szInBuffer	Input buffer, character config buffer.
	[out] lpOutBuffer	Output buffer. For structure types, see Table 3-41.
	[in] dwOutBufferSize	Output buffer size.
	[in] pReserved	Reserved parameter.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	None.	

Table 3-44 Comparison of szCommand, query type and corresponding structure

szCommand	Query type	Corresponding structure
CFG_CAP_CMD_ACCESSCONTROLMANAGER	Access control capability	CFG_CAP_ACCESSCONTROL
CFG_CMD_NETWORK	IP config	CFG_NETWORK_INFO
CFG_CMD_DVRIP	Auto register config	CFG_DVRIP_INFO
CFG_CMD_NTP	NTP time sync	CFG_NTP_INFO
CFG_CMD_ACCESS_EVENT	Access control event config (door config information, always open and always closed period config, unlock at designated intervals, first card unlocking config)	CFG_ACCESS_EVENT_INFO
CFG_CMD_ACCESSTIMESCHEDULE	Card swiping period for access control (period config)	CFG_ACCESS_TIMESCHEDULE_INFO
CFG_CMD_OPEN_DOOR_GROUP	Combination unlocking by multiple persons config	CFG_OPEN_DOOR_GROUP_INFO
CFG_CMD_ACCESS_GENERAL	Basic config for access control (inter-door lock)	CFG_ACCESS_GENERAL_INFO
CFG_CMD_OPEN_DOOR_ROUTE	Collection of routes to open the door, also called anti-passback route config	CFG_OPEN_DOOR_ROUTE_INFO

3.3.3.3 Getting Device Capabilities CLIENT_ GetDevCaps

Table 3-45 Description of getting device capabilities

Item	Description	
Description	Get device capabilities.	
Function	BOOL CLIENT_GetDevCaps (● LLONG lLoginID, ● int nType, ● void* pInBuf, ● void* pOutBuf, ● int nWaitTime);	
Parameter	[in] lLoginID	Login handle, return value of CLIENT_LoginWithHighLevelSecurity.
	[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 period.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	None.	

Table 3-46 Comparison of nType, pInBuf and pOutBuf

nType	Description	pInBuf	pOutBuf
NET_FACEINFO_CAPS	Obtain the capability set for face access controller	NET_IN_GET_FACEINFO_CAPS	NET_OUT_GET_FACEINFO_CAPS

3.3.3.4 Querying Device Status CLIENT_QueryDevState

Table 3-47 Description of querying device status

Item	Description	
Description	Get the current working status of the front-end device.	
Function	BOOL CLIENT_QueryDevState (● LLONG lLoginID, ● int nType, ● char *pBuf, ● int nBufLen, ● int *pRetLen, ● int waittime=1000);	
Parameter	[in] lLoginID	Login handle, return value of CLIENT_LoginWithHighLevelSecurity.
	[in] nType	Device type. Control parameters vary by type.
	[out] pBuf	Output parameter, used to receive the returned data buffer in query. Based on different query types, the structures of returned data are also different.
	[in] nBufLen	Buffer length, in bytes.
	[in] waittime	Timeout period.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	None.	

Table 3-48 Correspondence between nType, query type and structure

nType	Description	pBuf
DH_DEVSTATE_SOFTWARE	Query device software version information	DHDEV_VERSION_INFO
DH_DEVSTATE_NETINTERFACE	Query network port information	DHDEV_NETINTERFACE_INFO
DH_DEVSTATE_DEV_RECORDSET	Query device record set information	NET_CTRL_RECORDSET_PARAM
DH_DEVSTATE_DOOR_STATE	Query access control status (door sensor)	NET_DOOR_STATUS_INFO

3.3.4 Network Setting

3.3.4.1 IP Settings

3.3.4.1.1 Parsing Config Information CLIENT_GetNewDevConfig

For details about CLIENT_ParseData, see "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig

Table 3-49 Description of querying config information

Item	Description	
Description	Get config in string format.	
Function	BOOL CLIENT_GetNewDevConfig (● LLONG lLoginID, ● char *szCommand, ● int nChannelID, ● char *szOutBuffer, ● DWORD dwOutBufferSize, ● int *error, ● int waittime =500);	
Parameter	[in] lLoginID	Login handle, return value of CLIENT_LoginWithHighLevelSecurity.
	[in] szCommand	Command parameter. See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."
	[in] nChannelID	Channel.
	[out]szOutBuffer	Output buffer.
	[in] dwOutBufferSize	Output buffer size.
	[out] error	Error Code.
	[in] waittime	Timeout period for waiting.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	Get config in string format, and information contained in each string is parsed by CLIENT_ParseData.	

Table 3-50 Description of error codes and meanings of the parameter error

Error code	Corresponding meanings
0	Successful
1	Failed
2	Illegal data
3	Cannot be set for now
4	Permission denied

3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig

Table 3-51 Description of setting config information

Item	Description	
Description	Get config 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, return value of CLIENT_LoginWithHighLevelSecurity.
	[in] szCommand	Command parameter information. See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."
	[in] nChannelID	Channel.
	[in] szInBuffer	Output buffer.
	[in] dwInBufferSize	Output buffer size.
	[out] error	Error Code.
	[out] restart	Whether the device is required to restart after the config is set. 1 means required; 0 means not required.
	[in] waittime	Timeout period for waiting.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	Set config in string format, and information contained in each string is packed by CLIENT_PacketData.	

Table 3-52 Description of error codes and meanings of the parameter error

Error code	Corresponding meanings
0	Successful
1	Failed
2	Illegal data
3	Cannot be set for now
4	Permission denied

3.3.4.1.4 Packing into String Format CLIENT_PacketData

Table 3-53 Description of packing into string format

Item	Description
Description	Pack the config information to be set into the string format.

Item	Description	
Function	BOOL CLIENT_PacketData (● char *szCommand, ● LPVOID lpInBuffer, ● DWORD dwInBufferSize, ● char *szOutBuffer, ● DWORD dwOutBufferSize);	
Parameter	[in] szCommand	Command parameter. See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData" for details.
	[in] lpInBuffer	Input buffer. For structure types, see "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."
	[in] dwInBufferSize	Input buffer size.
	[out] szOutBuffer	Output buffer.
	[in] dwOutBufferSize	Output buffer size.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	This interface is used with CLIENT_SetNewDevConfig. After using CLIENT_PacketData, set the packed information onto the device by CLIENT_SetNewDevConfig.	

3.3.4.2 Auto Register Config

3.3.4.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.4.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.4.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.4.2.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.5 Time Settings

3.3.5.1 Time Settings

Table 3-54 Description of time settings

Item	Description	
Description	Set the current time of the device.	
Function	BOOL CLIENT_SetupDeviceTime (● LLONG lLoginID, ● LPNET_TIME pDeviceTime,);	
Parameter	[in] lLoginID	Login handle, return value of CLIENT_LoginWithHighLevelSecurity.
	[in] pDeviceTime	Set device time pointer.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	When it is applied in system time sync, change the current system time of the front-end device to be synchronized with the local system time.	

3.3.5.2 NTP Time Sync, Time Zone Config

3.3.5.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.5.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.5.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.5.2.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.5.3 DST Setting

3.3.5.3.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.5.3.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.5.3.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.5.3.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.6 Maintenance Config

3.3.6.1 Modifying Login Password

3.3.6.1.1 Operating Device User CLIENT_OperateUserInfoNew

Table 3-55 Description of operating device user

Item	Description	
Description	Operate device user, supporting up to 64-channel device.	
Function	BOOL CLIENT_OperateUserInfoNew (● LLONG lLoginID, ● int nOperateType, ● void *opParam, ● void *subParam, ● void* pReserved, ● int nWaitTime = 1000);	
Parameter	[in]lLoginID	Return value of CLIENT_Login or CLIENT_LoginEx.
	[in] nOperateType	For operation types, see Table 3-53 for details.
	[in] opParam	Set the input buffer for user information. See Table 3-53 for details.
	[in] subParam	Set the auxiliary input buffer for user information. When the set type is modified information, part of the original user information shall be passed in here. See Table 3-53 for details.
	[in] pReserved	Reserved.
	[in]waittime	Timeout period, 1000ms by default, which can be set as needed.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	To implement the required function, set user information for changed devices.	

Table 3-56 Correspondence between nOperateType, opParam and subParam

nOperateType	opParam	subParam
6	USER_INFO_NEW	USER_INFO_NEW

3.3.6.2 Restart

3.3.6.2.1 Device Control CLIENT_ControlDevice

Table 3-57 Device control description

Item	Description
Description	Device control.

Item	Description	
Function	BOOL CLIENT_ControlDevice(● LLONG lLoginID, ● CtrlType type, ● void *param, ● int nWaitTime = 1000);	
Parameter	[in]lLoginID	Return value of CLIENT_Login or CLIENT_LoginEx.
	[in]type	Control type.
	[in]param	Control parameters vary by type.
	[in]waittime	Timeout period, 1000ms by default, which can be set as needed.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	None.	

Table 3-58 Comparison of type and param

Type	Description	Param
DH_CTRL_REBOOT	Restart	None
DH_CTRL_RECORDSET_INSERT	Add records to get the record set number	NET_CTRL_RECORDSET_INSERT_PARAM
DH_CTRL_RECORDSET_INSERTEX	Add fingerprint records to get the record set number	NET_CTRL_RECORDSET_INSERT_PARAM
DH_CTRL_RECORDSET_REMOVE	Delete a record according to the record set number	NET_CTRL_RECORDSET_PARAM
DH_CTRL_RECORDSET_CLEAR	Clear information of all record sets	NET_CTRL_RECORDSET_PARAM
DH_CTRL_RECORDSET_UPDATE	Update records of a record set number	NET_CTRL_RECORDSET_PARAM
DH_CTRL_RECORDSET_UPDATEEX	Update records of a fingerprint record set number	NET_CTRL_RECORDSET_PARAM
DH_CTRL_ACCESS_OPEN	Access control—open	CTRL_ARM_DISARM_PARAM
DH_CTRL_RESTOREDEFAULT	Restore the device to factory default	DH_RESTORE_COMMON

3.3.6.3 Restoring to Factory Defaults

3.3.6.3.1 Restoring to Factory Defaults CLIENT_ControlDevice, CLIENT_ResetSystem

- For details of CLIENT_ControlDevice, see "3.3.6.2.1 Device Control CLIENT_ControlDevice."
- For details of CLIENT_ResetSystem, see Table 3-56.

Table 3-59 Description of restoring to factory defaults

Item	Description
Description	Restoring to factory defaults.

Item	Description	
Function	BOOL CLIENT_ResetSystem (● LLONG lLoginID, ● const NET_IN_RESET_SYSTEM* pstInParam, ● NET_OUT_RESET_SYSTEM* pstOutParam, int nWaitTime);	
Parameter	[in]lLoginID	Return value of CLIENT_Login or CLIENT_LoginEx.
	[in] pstInParam	Input parameter for restoring to factory defaults.
	[out] pstOutParam	Output parameter for restoring to factory defaults.
	[in] nWaitTime	Timeout period.
Return Value	● Success: TRUE ● Failure: FALSE	

3.3.6.4 Device Upgrade

3.3.6.4.1 Starting Upgrading Device Program CLIENT_StartUpgradeEx

Table 3-60 Description of start upgrading device program

Item	Description	
Description	Start upgrading device program—extension.	
Function	LLONG CLIENT_StartUpgradeEx (● LLONG lLoginID, ● EM_UPGRADE_TYPE emType ● char *pchFileName, ● fUpgradeCallBack cbUpgrade, ● LDWORD dwUser);	
Parameter	[in]lLoginID	Return value of CLIENT_Login or CLIENT_LoginEx.
	[in] emType	Enumerated value. See Table 3-58 for details.
	[in] pchFileName	Name of file to be upgraded.
	[in] cbUpgrade	Upgrade progress callback function. See "4.8 Upgrade Progress Callback fUpgradeCallBackEx" for details.
	[in] dwUser	User-defined data.
Return Value	● Success: Upgrade handle ID ● Failure: 0	
Note	Set the upgrade of remote programs to return the program upgrade handle. Calling this interface has not sent upgrade program data, which will be sent by calling the CLIENT_SendUpgrade interface.	

Table 3-61 Enumerated value

emType	Meanings
DH_UPGRADE_BIOS_TYPE	BIOS upgrade
DH_UPGRADE_WEB_TYPE	WEB upgrade
DH_UPGRADE_BOOT_YPE	BOOT upgrade

emType	Meanings
DH_UPGRADE_CHARACTER_TYPE	Chinese character library
DH_UPGRADE_LOGO_TYPE	LOGO
DH_UPGRADE_EXE_TYPE	EXE, such as player
DH_UPGRADE_DEVCONSTINFO_TYPE	Inherent device information settings (such as hardware ID, MAC, SN)
DH_UPGRADE_PERIPHERAL_TYPE	Peripheral access sub chip (such as vehicle chip)
DH_UPGRADE_GEOINFO_TYPE	Geographic information positioning chip
DH_UPGRADE_MENU	Menu (pictures in the device operating interface)
DH_UPGRADE_ROUTE	Route file (such as bus routes)
DH_UPGRADE_ROUTE_STATE_AUTO	Bus stop announcement audio (matching with routes)
DH_UPGRADE_SCREEN	Dispatch screen (such as bus operating screen)

3.3.6.4.2 Starting Sending Upgrade File CLIENT_SendUpgrade

Table 3-62 Description of starting sending upgrade file

Item	Description	
Description	Start sending upgrade file.	
Function	BOOL CLIENT_SendUpgrade (● LLONG lUpgradeID);	
Parameter	[in] lUpgradeID	Upgrade handle ID.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	Send upgrade program data.	

3.3.6.4.3 Stop Upgrading CLIENT_StopUpgrade

Table 3-63 Description of stopping upgrading

Item	Description	
Description	Start sending upgrade file.	
Function	BOOL CLIENT_StopUpgrade (● LLONG lUpgradeID);	
Parameter	[in] lUpgradeID	Upgrade handle ID.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	Do not call this interface in callback function.	

3.3.6.5 Auto Maintenance

3.3.6.5.1 Querying Config Information CLIENT_NewDevConfig

Table 3-64 Description of querying config information

Item	Description
Description	Read device config information.

Item	Description	
Function	BOOL CLIENT_GetDevConfig (● LLONG lLoginID, ● DWORD dwCommand, ● LONG lChannel, ● LPVOID lpOutBuffer, ● DWORD dwOutBufferSize, ● LPDWORD lpBytesReturned, ● int waittime =500);	
Parameter	[in] lLoginID	Device login handle.
	[in] dwCommand	For device config commands, see Table 3-62 for details. Different dwCommand and lpOutBuffer correspond to different structures. See Table 3-62 for details.
	[in] lChannel	Channel number. If all channel data obtained is 0xFFFFFFFF and the command does not require channel number, this parameter is invalid.
	[out] lpOutBuffer	Pointer of received data buffer.
	[in] dwOutBufferSize	Length of received data buffer (in bytes).
	[out] lpBytesReturned	Length of data actually received.
	[in] waittime	Timeout period for waiting.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	None.	

Table 3-65 Correspondence between dwCommand and lpOutBuffer

dwCommand	Query type	Corresponding structure lpOutBuffer
DH_DEV_DST_CFG	DST configuration	CFG_NTP_INFO
DH_DEV_AUTOMTCFG	Auto maintenance config	DHDEV_AUTOMT_CFG

3.3.6.5.2 Setting Config Information CLIENT_SetDevConfig

Table 3-66 Description of setting config information

Item	Description	
Description	Set device config information.	
Function	BOOL CLIENT_SetDevConfig (● LLONG lLoginID, ● DWORD dwCommand, ● LONG lChannel, ● LPVOID lpInBuffer, ● DWORD dwInBufferSize, ● int waittime =500);	
Parameter	[in] lLoginID	Device login handle.

Item	Description	
	[in] dwCommand	For device config commands, see Table 3-62 for details. Different dwCommand and lpInBuffer correspond to different structures. See Table 3-62 for details.
	[in] lChannel	Channel number. If all channel data obtained is 0xFFFFFFFF and the command does not require channel number, this parameter is invalid.
	[in] lpInBuffer	Data buffer pointer.
	[in] dwInBufferSize	Data buffer length (in bytes).
	[in] waittime	Timeout period for waiting.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	None.	

3.3.7 Personnel Management

3.3.7.1 Collection of Personnel Information Fields

See "3.3.6.2.1 Device Control CLIENT_ControlDevice" and "3.3.3.4 Querying Device Status CLIENT_QueryDevState."

3.3.8 Door Config

3.3.8.1 Door Config Information

3.3.8.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.8.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.8.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.8.1.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.9 Door Time Config

3.3.9.1 Period Config

3.3.9.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.9.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.9.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.9.1.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.9.2 Always Open and Always Closed Period Config

3.3.9.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.9.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.9.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.9.2.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.9.3 Holiday Config

See "3.3.6.2.1 Device Control CLIENT_ControlDevice" and "3.3.3.4 Querying Device Status CLIENT_QueryDevState."

3.3.10 Advanced Config of Door

3.3.10.1 Unlock at Designated Intervals and First Card Unlock

3.3.10.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.10.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.10.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.10.1.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.10.2 Combination Unlock by Multiple Persons

3.3.10.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.10.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.10.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.10.2.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.10.3 Inter-door Lock

3.3.10.3.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.10.3.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.10.3.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.10.3.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.10.4 Anti-passback

3.3.10.4.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.3.10.4.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.3.10.4.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.3.10.4.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.3.10.5 Unlock Password

See "3.3.6.2.1 Device Control CLIENT_ControlDevice."

3.3.10.6 Device Log

3.3.10.6.1 Querying the Count of Device Logs CLIENT_QueryDevLogCount

Table 3-67 Description of querying the count of device logs

Item	Description	
Description	Query the count of device logs.	
Function	int CLIENT_QueryDevLogCount (● LLONG lLoginID, ● NET_IN_GETCOUNT_LOG_PARAM* pInParam, ● NET_OUT_GETCOUNT_LOG_PARAM* pOutParam, ● int waittime);	
Parameter	[in]lLoginID	Device login handle, return value of CLIENT_LoginWithHighLevelSecurity.
	[in] pInParam	Parameter for querying logs. See NET_IN_GETCOUNT_LOG_PARAM for details.

Item	Description	
	[out] pOutParam	Returned log count. See NET_OUT_GETCOUNT_LOG_PARAM for details.
	[in] waittime	Timeout period in query.
Return Value	● Return the queried log count.	
Note	None.	

3.3.10.6.2 Starting Querying Logs CLIENT_StartQueryLog

Table 3-68 Description of starting querying logs

Item	Description	
Description	Start querying device logs.	
Function	LLONG CLIENT_StartQueryLog (● LLONG lLoginID, ● const NET_IN_START_QUERYLOG* pInParam, ● NET_OUT_START_QUERYLOG* pOutParam, ● int nWaitTime);	
Parameter	[in]lLoginID	Device login handle, return value of CLIENT_LoginWithHighLevelSecurity.
	[in] pInParam	Parameter for starting querying logs. See NET_IN_START_QUERYLOG for details.
	[out] pOutParam	Output parameter for starting querying logs. See NET_OUT_START_QUERYLOG for details.
	[in] nWaitTime	Timeout period in query.
Return Value	● Success: Query handle ● Failure: 0	
Note	None.	

3.3.10.6.3 Getting Logs CLIENT_QueryNextLog

Table 3-69 Description of getting logs

Item	Description	
Description	Get logs.	
Function	BOOL CLIENT_QueryNextLog (● LLONG lLogID, ● NET_IN_QUERYNEXTLOG* pInParam, ● NET_OUT_QUERYNEXTLOG* pOutParam, ● int nWaitTime);	
Parameter	[in] lLogID	Query log handle.
	[in] pInParam	Input parameter for getting logs. See NET_IN_QUERYNEXTLOG for details.
	[out] pOutParam	Output parameter for getting logs. See NET_OUT_QUERYNEXTLOG for details.
	[in] nWaitTime	Timeout period in query.

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

3.3.10.6.4 Ending Querying Logs CLIENT_StopQueryLog

Table 3-70 Description of ending querying logs

Item	Description	
Description	Stop querying device logs.	
Function	BOOL CLIENT_StopQueryLog (● LLONG lLogID,);	
Parameter	[in] lLogID	Query log handle.
Return Value	● Success: TRUE, ● Failure: FALSE	
Description	None.	

3.3.11 Records Query

3.3.11.1 Unlock Records

3.3.11.1.1 Querying Record Count CLIENT_QueryRecordCount

Table 3-71 Description of querying record count

Item	Description	
Description	Query the count of records.	
Function	BOOL CLIENT_QueryRecordCount (● NET_IN_QUEYT_RECORD_COUNT_PARAM* pInParam, ● NET_OUT_QUEYT_RECORD_COUNT_PARAM* pOutParam, ● int waittime);	
Parameter	[in] pInParam	Input parameter for querying record count. The pInParam > lFindeHandle is pOutParam > lFindeHandle of CLIENT_FindRecord.
	[out] pOutParam	Output parameter for querying record count. The pOutParam > nRecordC is the record count.
	[in] waittime	Timeout period in query.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	Before calling this interface, you should call CLIENT_FindRecord first to open the query handle.	

3.3.11.1.2 Querying Records by Query Conditions CLIENT_FindRecord

Table 3-72 Description of querying records by query conditions

Item	Description	
Description	Query records by query conditions.	
Function	BOOL CLIENT_FindRecord (LLONG lLoginID, NET_IN_FIND_RECORD_PARAM* pInParam, NET_OUT_FIND_RECORD_PARAM* pOutParam , int waittime=3000);	
Parameter	[in]lLoginID	Device login handle.
	[in] pInParam	Input parameter for querying records.
	[out] pOutParam	Output parameter for querying records. Return to the query handle.
	[in] waittime	Timeout period for waiting.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	You can call this interface first to get the query handle, then call the CLIENT_FindNextRecord function to get the list of records. After the query is completed, you can call CLIENT_FindRecordClose to close the query handle.	

Table 3-73 Description of pInParam

pInParam Structure	Value Assignment	Description
emType	NET_RECORD_ACCESSCTLCARDREC_EX	Query door unlook records.

3.3.11.1.3 Querying Records CLIENT_FindNextRecord

Table 3-74 Description of querying records

Item	Description	
Description	Query records: nFilecount: count of files to be queried. When the return value is the count of media files and less than nFilecount, the query of files is completed within the corresponding period.	
Function	int CLIENT_FindNextRecord (● NET_IN_FIND_NEXT_RECORD_PARAM* pInParam, ● NET_OUT_FIND_NEXT_RECORD_PARAM* pOutParam, ● int waittime);	
Parameter	[in] pInParam	Input parameter for querying records. The pInParam > lFindeHandle is pOutParam > lFindeHandle of CLIENT_FindRecord.
	[out] pOutParam	Output parameter for querying records. Return to recods info.
	[in] waittime	Timeout period for waiting.
Return Value	● 1: Successfully get one record. ● 0: All records are got. ● -1: Parameter error.	
Note	None.	

Table 3-75 Description of pOutParam

pOutParam Structure	Value Assignment	Description
pRecordList	NET_RECORDSET_ACCESS_CTL_CARDREC	Query door unlook records.

3.3.11.1.4 Ending Record Query CLIENT_FindRecordClose

Table 3-76 Description of ending record query

Item	Description	
Description	Stop record query.	
Function	BOOL CLIENT_FindRecordClose (● LLONG lFindHandle,);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindRecord.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	Call CLIENT_FindRecord to open the query handle; after the query is completed, you should call this function to close the query handle.	

3.4 Access Controller/All-in-one Face Machine (Second-Generation)

3.4.1 Access Control

For details of the door control interface, see "3.1.5.1 Device Controlling CLIENT_ControlDeviceEx."

For details of the door contact status interface, see 3.3.3.4 Querying Device Status CLIENT_QueryDevState."

3.4.2 Alarm Event

See "3.1.6 Alarm Listening."

3.4.3 Viewing Device Information

3.4.3.1 Getting Device Capabilities CLIENT_QueryDevState

Table 3-77 Description of getting 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 period.
Return value	● Success: TRUE ● Failure: FALSE	
Description	None.	

Table 3-78 Comparison of nType, pInBuf and pOutBuf

nType	Description	pInBuf	pOutBuf
NET_ACCESSCONTROL_CAPS	Get the access control capability	NET_IN_AC_CAPS	NET_OUT_AC_CAPS

3.4.3.2 Querying Device Status CLIENT_QueryDevState

For details about CLIENT_QueryDevState, see "3.3.3.4 Querying Device Status CLIENT_QueryDevState."

3.4.4 Network Setting

See "3.3.4 Network Setting."

3.4.5 Time Settings

See "3.3.5 Time Settings."

3.4.6 Maintenance Config

See "3.3.6 Maintenance Config."

3.4.7 Personnel Management

3.4.7.1 User Management

3.4.7.1.1 User Information Management Interface for Access Control Devices CLIENT_OperateAccessUserService

Table 3-79 Description of user information management interface for access control devices

Item	Description	
Description	Personnel information management interface for access control devices.	
Function	BOOL CLIENT_OperateAccessUserService (LLONG lLoginID, NET_EM_ACCESS_CTL_USER_SERVICE emtype, void* pstInParam, void* pstOutParam, int nWaitTime);	
Parameter	[in] lLoginID	Login handle.
	[in] emtype	User information operation type.
	[in] pInBuf	User information management (input parameter).
	[out] pOutBuf	User information management (output parameter).
	[in] nWaitTime	Timeout period.
Return value	● Success: TRUE ● Failure: FALSE	
Description	None.	

Table 3-80 Comparison of nType, pInBuf and pOutBuf

emtype	Description	pInBuf	pOutBuf
NET_EM_ACCESS_CTL_USER_SERVICE_INSERT	Add user info	NET_IN_ACCESS_USER_SERVICE_INSERT	NET_OUT_ACCESS_USER_SERVICE_INSERT
NET_EM_ACCESS_CTL_USER_SERVICE_REMOVE	Delete user info	NET_IN_ACCESS_USER_SERVICE_REMOVE	NET_OUT_ACCESS_USER_SERVICE_REMOVE
NET_EM_ACCESS_CTL_USER_SERVICE_CLEAR	Clear all user information	NET_IN_ACCESS_USER_SERVICE_CLEAR	NET_OUT_ACCESS_USER_SERVICE_CLEAR

3.4.7.1.2 Starting to Find the Personnel Information CLIENT_StartFindUserInfo

Table 3-81 Description of starting to find the personnel information interface

Item	Description
Description	Starting to find the personnel information interface.

Item	Description	
Function	LLONG CLIENT_StartFindUserInfo (LLONG lLoginID, NET_IN_USERINFO_START_FIND* pstIn, NET_OUT_USERINFO_START_FIND* pstOut, int nWaitTime);	
Parameter	[in] lLoginID	Login handle.
	[in] pstIn	Starting to find the personnel information interface (input parameter).
	[out] pstOut	Starting to find the personnel information interface (output parameter).
	[in] nWaitTime	Timeout period.
Return value	● Success: Search handle ● Failure: 0	
Description	None	

3.4.7.1.3 Finding the Personnel Information Interface CLIENT_DoFindUserInfo

Table 3-82 Description of finding the personnel information interface

Item	Description	
Description	Finding the personnel information interface.	
Function	BOOL CLIENT_DoFindUserInfo (LLONG lFindHandle, NET_IN_USERINFO_DO_FIND* pstIn, NET_OUT_USERINFO_DO_FIND* pstOut, int nWaitTime);	
Parameter	[in] lFindHandle	Return value of CLIENT_StartFindUserInfo.
	[in] pstIn	Finding the personnel information interface (input parameter).
	[out] pstOut	Finding the personnel information interface (output parameter).
	[in] nWaitTime	Timeout period.
Return value	● Success: TRUE ● Failure: FALSE	
Description	None.	

3.4.7.1.4 Stopping Finding the Personnel Information Interface CLIENT_StartFindUserInfo

Table 3-83 Stopping finding the personnel information interface

Item	Description	
Description	Stopping finding the personnel information interface.	
Function	BOOL CLIENT_StopFindUserInfo (LLONG lFindHandle);	
Parameter	[in] lFindHandle	CLIENT_StartFindUserInfo return value.

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

3.4.7.2 Card Management

3.4.7.2.1 Card Information Management Interface for Access Control Devices CLIENT_OperateAccessCardService

Table 3-84 Description of card information management interface for access control devices

Item	Description	
Description	Card information management interface for access control devices.	
Function	BOOL CLIENT_OperateAccessCardService (● LLONG lLoginID, ● NET_EM_ACCESS_CTL_CARD_SERVICE emtype, ● void* pstInParam, ● void* pstOutParam, ● int nWaitTime);	
Parameter	[in] lLoginID	Login handle.
	[in] emtype	Card information operation type.
	[in] pInBuf	Card information management (input parameter).
	[out] pOutBuf	Card information management (output parameter).
	[in] nWaitTime	Timeout period.
Return value	● Success: TRUE ● Failure: FALSE	
Description	None	

Table 3-85 Comparison of nType, pInBuf and pOutBuf

emtype	Description	pInBuf	pOutBuf
NET_EM_ACCESS_CTL_CARD_SERVICE_INSERT	Add the card information	NET_IN_ACCESS_CARD_SERVICE_INSERT	NET_OUT_ACCESS_CARD_SERVICE_INSERT
NET_EM_ACCESS_CTL_CARD_SERVICE_REMOVE	Delete the card information	NET_IN_ACCESS_CARD_SERVICE_REMOVE	NET_OUT_ACCESS_CARD_SERVICE_REMOVE
NET_EM_ACCESS_CTL_CARD_SERVICE_CLEAR	Clear all card information	NET_IN_ACCESS_CARD_SERVICE_CLEAR	NET_OUT_ACCESS_CARD_SERVICE_CLEAR

3.4.7.2.2 Starting to Find the Card Information Interface CLIENT_StartFindCardInfo

Table 3-86 Description of starting to find the card information interface

Item	Description
Description	Starting to find the card information interface.

Item	Description	
Function	LLONG CLIENT_StartFindCardInfo (LLONG lLoginID, NET_IN_CARDINFO_START_FIND* pstIn, NET_OUT_CARDINFO_START_FIND* pstOut, int nWaitTime);	
Parameter	[in] lLoginID	Login handle.
	[in] pstIn	Starting to find the card information interface (input parameter).
	[out] pstOut	Starting to find the card information interface (output parameter).
	[in] nWaitTime	Timeout period.
Return value	● Success: Search handle ● Failure: 0	
Description	None.	

3.4.7.2.3 Finding the Card Information Interface CLIENT_DoFindCardInf

Table 3-87 Description of finding the card information interface

Item	Description	
Description	Finding the card information interface.	
Function	BOOL CLIENT_DoFindCardInfo (LLONG lFindHandle, NET_IN_CARDINFO_DO_FIND* pstIn, NET_OUT_CARDINFO_DO_FIND* pstOut, int nWaitTime);	
Parameter	[in] lFindHandle	Return value of CLIENT_StartFindCardInfo.
	[in] pstIn	Finding the card information interface (input parameter).
	[out] pstOut	Finding the card information interface (output parameter).
	[in] nWaitTime	Timeout period.
Return value	● Success: TRUE ● Failure: FALSE	
Description	None.	

3.4.7.2.4 Stopping Finding the Card Information Interface CLIENT_StopFindUserInfo

Table 3-88 Description of stopping finding the card information interface

Item	Description	
Description	Stopping finding the card information interface.	
Function	BOOL CLIENT_StopFindCardInfo (LLONG lFindHandle);	
Parameter	[in] lFindHandle	Return value of CLIENT_StartFindCardInf.

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

3.4.7.3 Face Management

3.4.7.3.1 Face Information Management Interface for Access Control Devices CLIENT_OperateAccessFaceService

Table 3-89 Description of face information management interface for access control devices

Item	Description	
Description	Face information management interface for access control devices.	
Function	BOOL CLIENT_OperateAccessFaceService (LLONG lLoginID, NET_EM_ACCESS_CTL_FACE_SERVICE emtype, void* pstInParam, void* pstOutParam, int nWaitTime);	
Parameter	[in] lLoginID	Login handle.
	[in] emtype	Face information operation type.
	[in] pInBuf	Face information management (input parameter).
	[out] pOutBuf	Face information management (output parameter).
	[in] nWaitTime	Timeout period.
Return value	● Success: TRUE ● Failure: FALSE	
Description	None.	

Table 3-90 Comparison of nType, pInBuf and pOutBuf

emtype	Description	pInBuf	pOutBuf
NET_EM_ACCESS_CTL_FACE_SERVICE_INSERT	Add the face information	NET_IN_ACCESS_FACE_SERVICE_INSERT	NET_OUT_ACCESS_FACE_SERVICE_INSERT
NET_EM_ACCESS_CTL_FACE_SERVICE_GET	Find the face information	NET_IN_ACCESS_FACE_SERVICE_GET	NET_OUT_ACCESS_FACE_SERVICE_GET
NET_EM_ACCESS_CTL_FACE_SERVICE_UPDATE	Update the face information	NET_IN_ACCESS_FACE_SERVICE_UPDATE	NET_OUT_ACCESS_FACE_SERVICE_UPDATE
NET_EM_ACCESS_CTL_FACE_SERVICE_REMOVE	Delete the face information	NET_IN_ACCESS_FACE_SERVICE_REMOVE	NET_OUT_ACCESS_FACE_SERVICE_REMOVE
NET_EM_ACCESS_CTL_FACE_SERVICE_CLEAR	Clear the face information	NET_IN_ACCESS_FACE_SERVICE_CLEAR	NET_OUT_ACCESS_FACE_SERVICE_CLEAR

3.4.7.4 Fingerprint Management

3.4.7.4.1 Fingerprint Information Management Interface for Access Control Devices CLIENT_OperateAccessFingerprintService

Table 3-91 Description of fingerprint information management interface for access control devices

Item	Description	
Description	Fingerprint information management interface for access control devices.	
Function	BOOL CLIENT_OperateAccessFingerprintService (LLONG lLoginID, NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE emtype, void* pstInParam, void* pstOutParam, int nWaitTime);	
Parameter	[in] lLoginID	Login handle.
	[in] emtype	Fingerprint information operation type.
	[in] pInBuf	Fingerprint information management (input parameter).
	[out] pOutBuf	Fingerprint information management (output parameter).
	[in] nWaitTime	Timeout period.
Return value	● Success: TRUE ● Failure: FALSE	
Description	None.	

Table 3-92 Comparison of nType, pInBuf and pOutBuf

emtype	Description	pInBuf	pOutBuf
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_INSERT	Add the fingerprint information	NET_IN_ACCESS_FINGERPRINT_SERVICE_INSERT	NET_OUT_ACCESS_FINGERPRINT_SERVICE_INSERT
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_GET	Find the fingerprint information	NET_IN_ACCESS_FINGERPRINT_SERVICE_GET	NET_OUT_ACCESS_FINGERPRINT_SERVICE_GET
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_UPDATE	Update the fingerprint information	NET_IN_ACCESS_FINGERPRINT_SERVICE_UPDATE	NET_OUT_ACCESS_FINGERPRINT_SERVICE_UPDATE
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_REMOVE	Delete the fingerprint information	NET_IN_ACCESS_FINGERPRINT_SERVICE_REMOVE	NET_OUT_ACCESS_FINGERPRINT_SERVICE_REMOVE
NET_EM_ACCESS_CTL_FINGERPRINT_SERVICE_CLEAR	Clear the fingerprint information	NET_IN_ACCESS_FINGERPRINT_SERVICE_CLEAR	NET_OUT_ACCESS_FINGERPRINT_SERVICE_CLEAR

3.4.8 Door Config

3.4.8.1 Door Config Information

3.4.8.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.4.8.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.4.8.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.4.8.1.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.4.9 Door Time Config

3.4.9.1 Period Config

3.4.9.1.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.4.9.1.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.4.9.1.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.4.9.1.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.4.9.2 Always open and always closed period config

3.4.9.2.1 Parsing Config Information CLIENT_GetNewDevConfig

See "3.3.3.2 Parsing the Queried Config Information CLIENT_ParseData."

3.4.9.2.2 Querying Config Information CLIENT_GetNewDevConfig

See "3.3.4.1.2 Querying Config Information CLIENT_GetNewDevConfig."

3.4.9.2.3 Setting Config Information CLIENT_SetNewDevConfig

See "3.3.4.1.3 Setting Config Information CLIENT_SetNewDevConfig."

3.4.9.2.4 Packing into String Format CLIENT_PacketData

See "3.3.4.1.4 Packing into String Format CLIENT_PacketData."

3.4.9.3 Holiday group

3.4.9.3.1 Getting the Holiday Group Interface CLIENT_GetConfig

Table 3-93 Description of getting the holiday group interface

Item	Description	
Description	Getting the holiday group interface.	
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	Login handle.
	[in] emCfgOpType	Set the type of configuration info. Holiday group config: NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP.
	[in] nChannelID	Channel.
	[out] szOutBuffer	Get the buffer address of configuration info.
	[in] dwOutBufferSize	Buffer address size.
	[in] waittime	Timeout period.
	[in] reserve	Reserved parameter.
Return value	● Success: TRUE ● Failure: FALSE	
Description	None.	

Table 3-94 Description of emCfgOpType

emCfgOpType	Description	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP	Get the holiday group info	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO structure dimension

3.4.9.3.2 Setting the Holiday Group Interface CLIENT_SetConfig

Table 3-95 Description of setting the holiday group interface

Item	Description	
Description	Setting the holiday group interface.	
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);	
Parameter	[in] lLoginID	Login handle.
	[in] emCfgOpType	Set the configuration type. Holiday group config: NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP.
	[in] nChannelID	Channel.
	[in] szInBuffer	Configured buffer address.
	[in] dwInBufferSize	Buffer address size.
	[in] waittime	Timeout period.
	[in] restart	Whether to restart.
	[in] reserve	Reserved parameter.
Return value	● Success: TRUE ● Failure: FALSE	
Description	None.	

Table 3-96 Description of emCfgOpType

emCfgOpType	Description	szInBuffer	dwInBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAY_GROUP	Setting the holiday group info	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO	NET_CFG_ACCESSCTL_SPECIALDAY_GROUP_INFO structure dimension

3.4.9.4 Holiday plan

For details, see "3.4.9.3 Holiday group."

Table 3-97 Description of emCfgOpType

emCfgOpType	Description	szOutBuffer	dwOutBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE	Set the holiday plan info	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO structure dimension

Table 3-98 Description of emCfgOpType

emCfgOpType	Description	szInBuffer	dwInBufferSize

emCfgOpType	Description	szInBuffer	dwInBufferSize
NET_EM_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE	Set the holiday plan info	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO	NET_CFG_ACCESSCTL_SPECIALDAYS_SCHEDULE_INFO structure dimension

3.4.10 Advanced Config of Door

See "3.3.10 Advanced Config of Door."

3.4.11 Records Query

3.4.11.1 Unlock Records

See "3.3.11.1 Unlock Records."

3.4.11.2 Alarm Records

3.4.11.2.1 Querying Record Count CLIENT_QueryRecordCount

Table 3-99 Description of querying record count

Item	Description	
Description	Query the count of records.	
Function	BOOL CLIENT_QueryRecordCount (NET_IN_QUEYT_RECORD_COUNT_PARAM* pInParam, NET_OUT_QUEYT_RECORD_COUNT_PARAM* pOutParam, int waittime);	
Parameter	[in] pInParam	Input parameter for querying record count. The pInParam > lFindeHandle is pOutParam > lFindeHandle of CLIENT_FindRecord.
	[out] pOutParam	Output parameter for querying record count. The pOutParam > nRecordCount is the record count.
	[in] waittime	Timeout period in query.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	Before calling this interface, you should call CLIENT_FindRecord first to open the query handle.	

3.4.11.2.2 Querying Records by Query Conditions CLIENT_FindRecord

Table 3-100 Description of querying records by query conditions

Item	Description
Description	Query records by query conditions.

Item	Description	
Function	BOOL CLIENT_FindRecord (LLONG lLoginID, NET_IN_FIND_RECORD_PARAM* pInParam, NET_OUT_FIND_RECORD_PARAM* pOutParam , int waittime=3000);	
Parameter	[in]lLoginID	Device login handle.
	[in] pInParam	Input parameter for querying records.
	[out] pOutParam	Output parameter for querying records. Return to the query handle.
	[in] waittime	Timeout period for waiting.
Return Value	● Success: TRUE, ● Failure: FALSE	
Note	You can call this interface first to get the query handle, then call the CLIENT_FindNextRecord function to get the list of records. After the query is completed, you can call CLIENT_FindRecordClose to close the query handle.	

Table 3-101 Description of pInParam

pInParam Structure	Value Assignment	Description
emType	NET_RECORD_ACCESS_ALARMRECORD	Query alarm records.

3.4.11.2.3 Querying Records CLIENT_FindNextRecord

Table 3-102 Description of querying records

Item	Description	
Description	Query records: nFilecount: count of files to be queried. When the return value is the count of media files and less than nFilecount, the query of files is completed within the corresponding period.	
Function	int CLIENT_FindNextRecord (● NET_IN_FIND_NEXT_RECORD_PARAM* pInParam, ● NET_OUT_FIND_NEXT_RECORD_PARAM* pOutParam, ● int waittime);	
Parameter	[in] pInParam	Input parameter for querying records. The pInParam > lFindeHandle is pOutParam > lFindeHandle of CLIENT_FindRecord.
	[out] pOutParam	Output parameter for querying records. Return to recods info.
	[in] waittime	Timeout period for waiting.
Return Value	● 1: Successfully get one record. ● 0: All records are got. ● -1: Parameter error.	
Note	None.	

Table 3-103 Description of pOutParam

pOutParam Structure	Value Assignment	Description
pRecordList	NET_RECORD_ACCESS_ALARMRECORD_INFO	Query alarm records.

3.4.11.2.4 Ending Record Query CLIENT_FindRecordClose

Table 3-104 Description of ending record query

Item	Description	
Description	Stop record query.	
Function	BOOL CLIENT_FindRecordClose (● LLONG lFindHandle,);	
Parameter	[in] lFindHandle	Return value of CLIENT_FindRecord.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	Call CLIENT_FindRecord to open the query handle; after the query is completed, you should call this function to close the query handle.	

4 Callback Function

4.1 Device Searching Callback fSearchDevicesCB

Table 4-1 Description of callback function for searching device

Item	Description	
Description	Callback function for searching device.	
Function	typedef void(CALLBACK *fSearchDevicesCB)(● DEVICE_NET_INFO_EX * pDevNetInfo, ● void* pUserData);	
Parameter	[out]pDevNetInfo	Searched device information.
	[out]pUserData	User data.
Return Value	None.	
Note	None.	

4.2 Device Searching Callback fSearchDevicesCBEx

Table 4-2 Callback of searching devices

Item	Description	
Name	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 Disconnection Callback fDisConnect

Table 4-3 Description of disconnecting callback function

Item	Description
Description	Disconnection callback.

Item	Description	
Function	typedef void (CALLBACK *fDisConnect)(● LLONG lLoginID, ● char *pchDVRIP, ● LONG nDVRPort, ● LDWORD dwUser);	
Parameter	[out]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity.
	[out]pchDVRIP	Disconnected device IP.
	[out]nDVRPort	Disconnected device port.
	[out]dwUser	User parameters for callback function.
Return Value	None.	
Note	None.	

4.4 Reconnection Callback fHaveReConnect

Table 4-4 Description of reconnecting callback function

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	Reconnected device IP.
	[out]nDVRPort	Reconnected device port.
	[out]dwUser	User parameters for callback function.
Return Value	None.	
Note	None.	

4.5 Callback for Real-time Monitoring Data fRealDataCallBackEx2

Table 4-5 Description of callback function for real-time monitoring data

Item	Description
Description	Callback function for real-time monitoring data.

Item	Description	
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 means raw data ● 1 means data with frame information ● 2 means YUV data ● 3 means PCM audio data
	[out]pBuffer	Monitoring data block address.
	[out]dwBufSize	Length of monitoring data block, in bytes.
	[out]param	Parameter structure for callback data. The type is different if the dwDataType value is different. ● When dwDataType is 0, param is null pointer. ● When dwDataType is 1, param is the structure pointer tagVideoFrameParam. ● When dwDataType is 2, param is the structure pointer tagCBYUVDataParam. ● When dwDataType is 3, param is the structure pointer tagCBPCMDataParam.
	[out]dwUser	User parameters for callback function.
Return Value	None.	
Note	None.	

4.6 Audio Data Callback pfAudioDataCallBack

Table 4-6 Description of audio data callback function

Item	Description	
Description	Audio data callback for voice talk.	
Function	typedef void (CALLBACK *pfAudioDataCallBack)(● LLONG lTalkHandle, ● char *pDataBuf, ● DWORD dwBufSize, ● BYTE byAudioFlag, ● LDWORD dwUser);	
Parameter	[out]lTalkHandle	Return value of CLIENT_StartTalkEx.
	[out]pDataBuf	Audio data block address.
	[out]dwBufSize	Length of audio data block, in bytes.

Item	Description	
	[out]byAudioFlag	Flag of data type ● 0 means that the data is locally collected. ● 1 means that the data is sent from the device.
	[out]dwUser	User parameters for callback function.
Return Value	None.	
Note	None.	

4.7 Alarm Callback fMessCallBack

Table 4-7 Description of alarm callback function

Item	Description	
Description	Alarm callback function.	
Function	BOOL (CALLBACK *fMessCallBack)(● LONG lCommand, ● LLONG lLoginID, ● char *pBuf, ● DWORD dwBufLen, ● char *pchDVRIP, ● LONG nDVRPort, ● LDWORD dwUser);	
Parameter	[out]lCommand	Alarm type. See Table 4-8 for details.
	[out]lLoginID	Return value of login interface.
	[out]pBuf	Buffer that receives alarm data, which is filled with different data according to different listening interfaces called and lCommand values.
	[out]dwBufLen	Length of pBuf, in bytes.
	[out]pchDVRIP	Device IP.
	[out]nDVRPort	Port.
	[out]dwUser	User-defined data.
Return Value	● Success: TRUE ● Failure: FALSE	
Note	Usually, call the set callback function during application initialization, and process properly in the callback function according to different device ID and command values.	

Table 4-8 Correspondence between alarm type and structure

Alarm business	Alarm type name	lCommand	pBuf
Alarm host	Local alarm event	DH_ALARM_ALARM_EX2	ALARM_ALARM_INFO_EX2
	Power failure event	DH_ALARM_POWERFAULT	ALARM_POWERFAULT_INFO

Alarm business	Alarm type name	lCommand	pBuf
	Dismantlement prevention event	DH_ALARM_CHASSISINTRUDED	ALARM_CHASSISINTRUDED_INFO
	Extended alarm input channel event	DH_ALARM_ALARMEXTENDED	ALARM_ALARMEXTENDED_INFO
	Emergency event	DH_URGENCY_ALARM_EX	The data is a 16-byte array, and each byte represents a channel status ● 1: With alarms ● 0: Without alarms
	Low battery voltage event	DH_ALARM_BATTERYLOWPOWER	ALARM_BATTERYLOWPOWER_INFO
	Device inviting platform to talk event	DH_ALARM_TALKING_INVITE	ALARM_TALKING_INVITE_INFO
	Device arming mode change event	DH_ALARM_ARMMODE_CHANGE_EVENT	ALARM_ARMMODE_CHANGE_INFO
	Protection zone bypass status change event	DH_ALARM_BYPASSMODE_CHANGE_EVENT	ALARM_BYPASSMODE_CHANGE_INFO
	Alarm input source signal event	DH_ALARM_INPUT_SOURCE_SIGNAL	ALARM_INPUT_SOURCE_SIGNAL_INFO
	Alarm clearing event	DH_ALARM_ALARMCLEAR	ALARM_ALARMCLEAR_INFO
	Sub-system status change event	DH_ALARM_SUBSYSTEM_STATE_CHANGE	ALARM_SUBSYSTEM_STATE_CHANGE_INFO
	Extension module offline event	DH_ALARM_MODULE_LOST	ALARM_MODULE_LOST_INFO
	PSTN offline event	DH_ALARM_PSTN_BREAK_LINE	ALARM_PSTN_BREAK_LINE_INFO
	Analog quantity alarm event	DH_ALARM_ANALOG_PULSE	ALARM_ANALOGPULSE_INFO
	Alarm transmission event	DH_ALARM_PROFILE_ALARM_TRANSMIT	ALARM_PROFILE_ALARM_TRANSMIT_INFO

Alarm business	Alarm type name	lCommand	pBuf
	Wireless device low battery alarm event	DH_ALARM_WIRELESSDEV_LOWPOWER	ALARM_WIRELESSDEV_LOWPOWER_INFO
	Protection zone arming and disarming status change event	DH_ALARM_DEFENCE_ARMMODE_CHANGE	ALARM_DEFENCE_ARMMODECHANGE_INFO
	Sub-system arming and disarming status change event	DH_ALARM_SUBSYSTEM_ARMMODE_CHANGE	ALARM_SUBSYSTEM_ARMMODECHANGE_INFO
	Detector abnormality alarm	DH_ALARM_SENSOR_ABNORMAL	ALARM_SENSOR_ABNORMAL_INFO
	Patient activity status alarm event	DH_ALARM_PATIENTDETECTION	ALARM_PATIENTDETECTION_INFO
Access Control	Access control event	DH_ALARM_ACCESS_CTL_EVENT	ALARM_ACCESS_CTL_EVENT_INFO
	Details of access control unlocking event	DH_ALARM_ACCESS_CTL_NOT_CLOSE	ALARM_ACCESS_CTL_NOT_CLOSE_INFO
	Details of intrusion event	DH_ALARM_ACCESS_CTL_BREAK_IN	ALARM_ACCESS_CTL_BREAK_IN_INFO
	Details of repeated entry event	DH_ALARM_ACCESS_CTL_REPEAT_ENTER	ALARM_ACCESS_CTL_REPEAT_ENTER_INFO
	Malicious unlocking event	DH_ALARM_ACCESS_CTL_MALICIOUS	ALARM_ACCESS_CTL_MALICIOUS
	Details of forced card swiping event	DH_ALARM_ACCESS_CTL_DURESS	ALARM_ACCESS_CTL_DURESS_INFO
	Combination unlocking by multiple persons event	DH_ALARM_OPENDOORGROUP	ALARM_OPEN_DOOR_GROUP_INFO

Alarm business	Alarm type name	lCommand	pBuf
	Dismantlement prevention event	DH_ALARM_CHASSISINTRUDED	ALARM_CHASSISINTRUDED_INFO
	Local alarm event	DH_ALARM_ALARM_EX2	ALARM_ALARM_INFO_EX2
	Access control status event	DH_ALARM_ACCESS_CTL_STATUS	ALARM_ACCESS_CTL_STATUS_INFO
	Bolt alarm	DH_ALARM_ACCESS_CTL_STATUS	ALARM_ACCESS_CTL_STATUS_INFO
	Fingerprint acquisition event	DH_ALARM_FINGER_PRINT	ALARM_CAPTURE_FINGER_PRINT_INFO
Video Intercom	No response to the call in direct connection event	DH_ALARM_CALL_NO_ANSWERED	NET_ALARM_CALL_NO_ANSWERED_INFO
	Mobile phone number report event	DH_ALARM_TELEPHONE_CHECK	ALARM_TELEPHONE_CHECK_INFO
	VTS status report	DH_ALARM_VTSTATE_UPDATE	ALARM_VTSTATE_UPDATE_INFO
	VTO object recognition	DH_ALARM_ACCESSIDENTIFY	NET_ALARM_ACCESSIDENTIFY
	Device inviting another device to start talk event	DH_ALARM_TALKING_INVITE	ALARM_TALKING_INVITE_INFO
	Device canceling talk request event	DH_ALARM_TALKING_IGNORE_INVITE	ALARM_TALKING_IGNORE_INVITE_INFO
	Device actively hanging up talk event	DH_ALARM_TALKING_HANGUP	ALARM_TALKING_HANGUP_INFO
	Radar monitoring overspeed alarm event	DH_ALARM_RADAR_HIGH_SPEED	ALARM_RADAR_HIGH_SPEED_INFO

4.8 Upgrade Progress Callback fUpgradeCallBackEx

Table 4-9 Description of upgrade progress callback function

Item	Description	
Description	Upgrade progress callback function.	
Function	void (CALLBACK *fUpgradeCallBackEx)(● LLONG lLoginID, ● LLONG lUpgradechannel, ● INT64 nTotalSize, ● INT64 nSendSize, ● LDWORD dwUser);	
Parameter	[out]lLoginID	Return value of login interface.
	[out] lUpgradechannel	Upgrade handle ID returned by CLIENT_StartUpgradeEx2.
	[out] nTotalSize	Total length of upgrade file, in bytes.
	[out] nSendSize	Sent file length, in bytes; when it is -1, it means the sending of upgrade file has ended.
	[out]dwUser	User-defined data.
Return Value	None.	
Description	Device upgrade program callback function prototype supports upgrade files above G. nTotalSize = 0, nSendSize = -1 means that upgrade is completed. nTotalSize = 0, nSendSize = -2 means upgrade error. nTotalSize = 0, nSendSize = -3 means that the user has no upgrade permission. nTotalSize = 0, nSendSize = -4 means that the upgrade program version is too low. nTotalSize = -1, nSendSize = XX means upgrade progress. nTotalSize = XX, nSendSize = XX means the progress of sending upgrade files.	

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 Allowlist**

   We suggest you to enable allowlist 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 allowlist.

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 blocklist and allowlist feature to reduce the risk that your device might be attacked.